Maximizing Correctness with Minimal User Effort to Learn Data Transformations

Bo Wu
Computer Science Department
University of Southern California
4676 Admiralty Way
Marina del Rey,CA
bowu365@gmail.com

Craig A. Knoblock
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA
knoblock@isi.edu

ABSTRACT

Data transformation often requires users to write many trivial and task-dependent programs to transform thousands of records. Recently, programming-by-example approaches enable users to transform data without coding. A key challenge of these PBE approaches is to deliver correctly transformed results on large datasets, as these transformation programs are likely to be generated by non-expert users. To address this challenge, existing approaches aim to identify a small set of potentially incorrect records and ask users to examine these records instead of the entire dataset. However, as the transformation scenarios are highly task-dependent, existing approaches cannot capture the incorrect records for various scenarios. In this paper, our approach learns from past transformation scenarios to generate a meta-classifier to identify the incorrect records. Our approach color-codes these transformed records and then presents them for users to examine. The approach allows users to either enter an example for a record transformed incorrectly or confirm the correctness of a record. Our approach can learn from the users' labels to refine the meta-classifier to accurately identify the incorrect records. Simulation results and a user study show that our method can identify the incorrectly transformed records and reduce the user efforts in examining the results.

Categories and Subject Descriptors

H.5.2. [**Information Interfaces and Presentation**]: User Interfaces; D.2.4. [**Software Engineering**]: Program Verification; K.8.1. [**Personal Computing**]: Database processing

Keywords

Programming by Example, Data Transformation, Program Synthesis

1. INTRODUCTION

500 million users are using spreadsheets to manage their data. These users come from various backgrounds and lack the necessary programming skills to automate their tasks. Programming-by-example (PBE) approaches [12] enable these users to generate programs without coding. Recently, these approaches have been successfully applied to data transformation problems [6] to save users from writing many task-dependent transformation programs.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

IUI'16, March 07 - 10, 2016, Sonoma, CA, USA

© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ISBN 978-1-4503-4137-0/16/03…$15.00
DOI: http://dx.doi.org/10.1145/2856767.2856791

To use a PBE system for data transformation, users are only required to provide input-output pairs (also referred to as example). The PBE system generates the programs that are consistent with given examples. For example, Figure 1(b) shows the dimensions for a set of artworks. To extract the first degree (height) from the dimension, the user enters "10" as the target output for the first entry. The PBE approach generates programs that can transform the inputs into corresponding outputs specified in the examples. It applies the program to the rest of records to transform them. If the user finds any incorrect output, she can provide a new example, the approach refines the program to make it consistent with all given examples. The user often interacts with the system for several iterations and stops when she determines that all records are transformed correctly.

Despite the success of generating programs using PBE approaches, the correctness of the results is still an issue. Real-world data transformation often involves thousands of records with various formats. Each record consists of *raw data (input)* and *transformed data (output)*. The users are often not aware of all the variations of the data that they should transform. They know whether the records are transformed correctly when they see them. They lack the insight of the unseen formats of the records buried in the middle of datasets.

To help users verify whether the records are transformed correctly, existing PBE approaches [6, 13, 19] provide recommendations or highlight certain records for the users. The user checks these records and provides new examples for those incorrect records. Here, we consider a record as transformed correctly (referred to as *a correct record*), when it is transformed into the expected format. Otherwise, the record is considered incorrect (referred to as *an incorrect record*).

To generate such recommendations, there are several challenges. First, the dataset is usually huge. The users need to see the results to provide additional examples if necessary on the fly. Fully transforming the entire dataset and analyzing all the transformed records to generate recommendations takes too long to be practical.

Second, the users' intentions are highly task-dependent and there is not a universal rule for determining whether the results are correct. Meanwhile, the approach should be able to hypothesize users' intentions accurately to provide useful recommendations. For example, the scenario in Figure 1(a) is to encode different texts into numbers. The users want to transform the three records into "3", "2" and "1" respectively. After giving the two examples in the dashed rectangle, the learned program does not transform the 'Fewer than 100' into the expected value '1', as it has not seen an example of that type before. The problem is 'Fewer than 100' has different words from

Raw (Input)	Transformed (Output)
300 or more	3
Between 100 and 299	2
Fewer than 100	3
...	...

Examples

Incorrect record

(a) incorrect record has a different input format

Raw (Input)	Transformed (Output)
10" x 8	10
26" H x 24" W x 12.5"	26
3 x 6"	3 x 6
...	...

Examples

Incorrect record

(b) incorrect record has a different output format

Figure 1: Different rules for recognizing incorrect records

the inputs of the two examples when they are represented using bag-of-words model [4]. To capture this incorrect record, we can use a rule to identify the records that have different words in the *input* from the examples. However, in Figure 1(b), the counts of numbers, blank spaces, quotes and "x" in the inputs are the same for the third and the first record. We need to use a different rule that identifies the records with different *output* formats to locate the incorrect records since the output of the third record contains an "x" and blank spaces while the first two records do not have.

Third, recommendation should place the incorrect records at the beginning of the recommended list so that users can easily notice these records. Otherwise, users have to examine many correct records before identifying the incorrect one, which would be a burden.

Fourth, users are often too confident with their results to examine the recommended records, which they regard as an extra burden [14, 10]. Even when there are incorrect records in the recommendation, the users may ignore them and stop the transformation.

To address the challenges above, our approach samples records to allow users to focus on a small portion of the entire dataset. It statistically guarantees that a user-specified percentage of the records of the entire dataset are transformed correctly with certain confidence when all records in the small sample are transformed correctly. Our approach also maintains a library of different classifiers representing different rules for checking whether a record is transformed correctly. It uses an ensemble method to combine these different classifiers to automatically identify incorrect records in various scenarios. To save users' time in checking the recommendations, we provide users two ways to label the recommended records: (1) users can confirm a recommended record is correct or (2) they can provide the expected outputs for the incorrect records. Our approach then learns from these labels and provides users with refined recommendations. Finally, besides providing the recommendation that exposes incorrect records to users, we have also developed a method that identifies a minimal set of records that the user should examine before finishing the transformation.

Our contributions are summarized below:

- minimizing the user effort in obtaining a user-specified correctness
- allowing users to focus on a small sample of a large dataset
- combining multiple classifiers based on different perspectives for verifying the correctness of records
- refining recommendations when users confirm certain recommendations are correct or incorrect
- controlling the user overconfidence by requiring users to examine certain records before finishing the transformation.

2. PROGRAMMING BY EXAMPLE

Our approach is mainly built on the IPBE approach [21, 20, 22], which extents the previous PBE approach [6]. The approach is based on a domain specific transformation language (DSL), which supports a restricted, but expressive form of regular expressions, which also includes conditionals and loops. The approach synthesizes transformation programs from this language using examples.

```
Transform(value)
  switch(classify(value)):                        <- Conditional statement
    case format_1 :
      pos_1 = value.indexOf(START, NUM, 1)        <- Branch transformation program
      pos_2 = value.indexOf(NUM, '"', 1)
      output= substr(pos_1, pos_2)
    case format_2 :
      pos_3 = value.indexOf(START, NUM, 1)        <- Branch transformation program
      pos_4 = value.indexOf(NUM, BNK, 1)
      output=substr(pos_3, pos_4)
  return output
```

Figure 2: An example transformation program

$$branch = segment_1 + segment_2 + \ldots$$
$$segment = const \mid substr(p_s, p_e, offset = 0) \mid loop(w, branch)$$
$$p = indexOf(leftcxt, rightcxt, c + offset)$$
$$loop(w, branch) = branch_1 + branch_2 + \ldots + branch_w$$

Figure 3: The DSL for branch transformation programs

For example, after the users provide examples for the first three records shown in Figure 1(b), the transformation program is shown in Figure 2. The program has a function ($classify(value)$) to recognize the format of input (value). Based on the input format, it uses the conditional statement $switch$ to invoke the corresponding branch transformation program to transform the input. Here, $format_1$ refers to the format of the first and second records where a double quotation separates the first degree information from the rest of the string. The $format_2$ corresponds to the format of the third record where a blank space separates the first degree from the rest.

The syntax of the branch transformation program is a concatenation of several segment programs as shown in Figure 3, where it returns a string concatenating the outputs of all its segment programs ($segment_i$). This language allows the users to perform simple string deletion, insertion and reordering. For simplicity, the branch program in Figure 2 only contains one segment program. A segment program can be specified in three ways: (1) a constant string

($const$), (2) extracting substring from the input between two positions ($substr_i$) or (3) a loop statement ($loop(w, branch)$). The $substr$ program also has an offset with a default value 0, which is omitted by default. This parameter is described later with the loop statement. The two position programs (p_s and p_e) in the $substr$ program specify the start and end positions in the input to extract the substring. A position program identifies a location in the input. The position programs can be specified using (1) an absolute position, or (2) restricted regular expressions that identify the context of the given position, which can be represented using a triple as (leftcxt, rightcxt, occ). The "leftcxt" describes the left context of the position and "rightcxt" describes the right context. The "occ" refers the occ-th appearance of the position with the specified context. The contexts are all specified using token sequences.

Our approach tokenizes texts into token sequences. The different types of token are defined below. START represents the beginning of the raw value. END is for the end of the raw value. UWRD represents an uppercase letter, LWRD means a sequence of lower letters, the BNK means a blank space, NUM refers to a sequence of digits and WORD refers to a sequence of alphabetical letters, etc. Each punctuation is a token, such as Dquot refers to a double quotation. Therefore, (NUM,"“", 1) means the first occurrence of a position whose left is a number and whose right is a double quotation mark. Our language also supports loop statements, which returns a concatenated results of w branch programs. The $i-th$ branch program's offset value is i. By having the offset values from 1 to w, the loop statement can repeatedly extract substrings of the same pattern.

In order to generate the transformation program, IPBE [21, 20, 22] first creates traces [9] for the given examples. A trace essentially specifies the input-output pairs for all transformation steps of each particular example when transforming the raw data into the target data. For example, one trace for the first example in Figure 1(b) is [$substring$ = "10"($pos_1 = 1, pos_2 = 3$)]. The trace specifies a way to transform the input into the target output. It dictates that the program should contain one segment program and its output should be a string "10". It also specifies that the substring should be extracted between position 1 and 3. Thus, the first position program (pos_1) should output 1 and the second (pos_2) should return 3. With the traces of examples, the IPBE generalizes over these specific traces to derive programs.

When the user provides a new example, IPBE tries the refine the previous program to generate a new program that is consistent with all given examples. If such program does not exist, the approach partitions the examples into multiple clusters using constrainted agglomerative clustering [20]. Each cluster contains the examples of the same format. IPBE learns a multi-class SVM classifier [2] to recognize these different formats. To learn the classifier, the approach first converts the records into features vectors. The features can be categorized into two types: (1) counts of tokens and (2) the average indexes of tokens in the sequence. As shown in Figure 4, the string "6 x 8"" is convert into a sequence of tokens. Each token has its type and content. The feature vector contains the counts for different tokens, such as the 2 under NUM means there are two NUM tokens. It also contains the average positions of these tokens. For example, first NUM has an index of 1 in the token sequence and the second NUM's index is 5. Thus, the average position (NUM_pos) is 3 here.

After learning the SVM classifier, IPBE uses this classifier as the $classify$ function in the $switch$ statement shown in Figure 2 to choose the branch program that should be invoked to transform an input. It learns the branch transformation program for each cluster of examples. Finally, it combines the conditional statement with the branch programs to create final transformation program.

Text	6 x 8”				
Token sequence	NUM(6) BNK LWRD(x) BNK NUM(8) Dquot(“)				
Feature vector	NUM	BNK	NUM_pos	BNK_pos	...
	2	2	3	3	...

Figure 4: The token sequence and feature vector for a string

Examples you entered:	
10" H x 8" W	10
"14.75" H x 14.75" W x 1.5" D	14.75
H: 58 x W: 25"	58
	30 x 46
Recommended Examples:	
30 x 46"	30 x 46
11" H x 6" diameter	11
Sampled Records:	
12" H x 9" W	12
10" H x 8" W	10

Figure 5: User interface

3. VERIFYING THE TRANSFORMED DATA

To verify the correctness of the transformed data, our approach samples records from the entire dataset. The approach then automatically identifies the potentially incorrect records and recommends these identified records for the users to examine. When the recommendations are shown to users, they can provide the expected values for the incorrect records or they can confirm a record is transformed correctly. Our approach uses the records that the users have edited or confirmed as new examples to refine the recommendations.

The user interface of our system is shown in Figure 5. The interface consists of three areas:

- examples you entered: this area shows all the examples provided by the user. There are also buttons with cross icons used for deleting previous examples
- recommended examples: this area shows all the potentially incorrect records for users to examine. If the user finds an incorrect record, she can click the record and enter the target output in the popup window as in Figure 5. She can also click the button with a check icon to confirm that the record is correct
- sampled records: this area shows all the records in the sample.

The records in the GUI are color-coded. The transformed result is a concatenation of the substrings either extracted from the inputs or inserted as constant strings. The substrings that are extracted from inputs are colored in the same color in both the input and the transformed result. As shown in the Figure 5, the "10" is colored blue in both the input and output of the last record to indicate the correspondences between the input and output. By providing the color-coding, the users can have an insight of the learned programs.

They can also notice the potentially irregular color pattern to identify the incorrect records.

4. SAMPLING RECORDS

Our approach uses hypothesis testing to decide whether the number of incorrect records in the entire dataset is below a certain percentage. The hypothesis is *the percentage of incorrect records is smaller than* p_{lower}. The alternative hypothesis is *the percentage of incorrect records is not less than* p_{upper}. The p_{lower} represents that the percentage of incorrect records that we want to achieve in the dataset. The p_{upper} refers to a percentage of incorrect records that we want to avoid ($p_{upper} \geqslant p_{lower}$).

We use the binomial distribution $B(n, p)$ to model the distribution of incorrect records, as each record is either transformed correctly or incorrectly. The parameter n is the number of sampled records. The parameter p is the probability that a record is incorrect, which can also be interpreted as a p fraction of records are transformed incorrectly. To find the sample size for testing the hypothesis, we use the binomial cumulative distribution function as shown in Formula 1 [3]. $Pr(x < Z_a; n, p)$ represents the probability of the number of incorrect records (x) is less than Z_a in the binomial distribution $B(n, p)$. We use $1-\alpha$ and $1-\beta$ to adjust our confidence level and power for the hypothesis test: (1) α controls the probability of rejecting our hypothesis when it is true and (2) $1-\beta$ controls the probability of rejecting the alternative hypothesis when it is false. Typically, the confidence level $1-\alpha$ is set to 0.95 and the power $1-\beta$ is set to 0.80 in practice [3]. The z_α is the allowable number of incorrect records. If the incorrect number of records x is smaller than z_α, our hypothesis passes the test with confidence α and power $1-\beta$ over the alternative hypothesis. Given α, β, n, p_{upper} and p_{lower}, we can calculate z_α based on Formula 1.

Since the user ideally stops when there is no incorrect record in the sample, the x is zero and it is strictly smaller than z_α.

$$\begin{aligned} &Pr(x < z_\alpha; n, p_{upper}) > 1-\beta, \\ &where\ Pr(x < z_\alpha; n, p_{lower}) < \alpha \end{aligned} \quad (1)$$

Demanding a lower error percentage, a higher confidence and power of the hypothesis often requires a larger sample. For example, when $p_{lower} = 0.01$, $\alpha < 0.05$, the alternative hypothesis is $p_{upper} \geqslant 0.02$ and $1-\beta > 0.8$, we need a sample of 910 records. Meanwhile, if we change the $p_{lower} = 0.001$ and $p_{upper} \geqslant 0.002$, the minimal sample size is 9635. However, we can configure the parameters to achieve the balance between sample size and level of confidence to meet different user requirements.

5. RECOMMENDING RECORDS

Our approach automatically examines the records in the sample to identify potentially incorrect records (line 2 to line 9 in Algorithm 1). It then sorts these records and recommends one for the users to examine.

Our approach has two phases. First, it identifies the *records* ($R_{runtime}$) with runtime errors. Runtime errors here are the errors that occur during the execution of transformation programs and cause the programs to exit abnormally. Second, our approach identifies the *questionable records* ($R_{questionable}$) with potential incorrect results when there is no record with runtime errors.

5.1 Finding the records with runtime errors

Our approach finds the records with runtime errors. These records cause the learned program to exit abnormally in execution. There are mainly two types of runtime errors: (1) the position program cannot locate a position and (2) the segment program has a start position larger than the end position.

Algorithm 1: Algorithm for recommending records

```
   Input: Set of all the records R, transformation program P and
          MetaClassifier F
   Output: Recommended set of Records R*
   R_runtime = [], R_questionable = [], R* = []
1  R_s = sample(R)
   for record r in R_s do
2  |   r_t = applyTransformation(r, P)
3  |   if r_t contains runtime error then
4  |   |   r_t.score = number of failed position programs
5  |   |   R_runtime.add(r_t)
   |   else
6  |   |   score = F.getScore(r_t)
7  |   |   if score < 0 then
8  |   |   |   r_t.score = score
9  |   |   |   R_questionable.add(r_t)
   |   |   end
   |   end
   end
   if R_runtime.isEmpty() then
10 |   sort R_questionable acscendingly based on record score
11 |   R* = R_questionable
   else
12 |   sort R_runtime acscendingly based on record score
13 |   R* = R_runtime
   end
   return R*
```

For example, the approach applies the learned program shown in Figure 2 to a new input "H: 24 x W: 7 " to extract the first degree information. The program uses the first branch program to transform this record. The start position program cannot locate a position in the input and output "-1", as "24" does not appear at the beginning of the input. The end position program cannot locate a position either, as there is no """ after "24". The corresponding segment program also has a runtime error too, as both its position and end position are smaller than 0 (-1). The other case of runtime error is that the segment program has a start position bigger than the end position. In this case, the two position programs of the segment program both successfully locate two indexes in the input. However, the end position is before the start position, which only causes a runtime error of the segment program.

To identify the records with runtime errors, our approach simply applies the learned program to the sampled records, collects all the records with runtime errors and puts them into the set $R_{runtime}$.

5.2 Building a meta-classifier for detecting questionable records

The set of binary classifiers used for building the meta-classifier can be categorized into 3 types: (1) classifiers based on the distance (f_{dist}) (2) classifiers based on the agreement of different programs ($f_{program}$) and (3) classifiers based on the format ambiguity ($f_{ambiguity}$).

5.2.1 Classifiers based on distance

This type of classifier calculates the distances from records to a set of records. Based on the distribution of the distances, it identifies the records with distances that are larger than certain standard deviations

from the chosen references. To calculate the distance between two records, the approach first converts the records to feature vectors and then calculates the Euclidean distance between the two vectors. The features used here are the same as the ones introduced in the previous section.

This type of classifier ($f_{dist}(e_i|r,t,c)$) is shown in Function 2. It classifies each record e_i as either a correct record (1) or an incorrect record (-1) based on its distance ($d_{e_i,r}$) from the reference (r). N is the number of records. Each classifier of this type can be characterized using a triple (r,t,c). r represents the *reference*. It has two values: "all records" or "examples". The string "all records" specifies that the approach calculates the distance between the feature vector of the record (e_i) with the mean vector of all records except e_i. The string "examples" means that the approach computes the distance from the record (e_i) to the mean vector of the examples. The t has three values: "input", "output" or "combined". "input" and "output" here mean that only inputs or outputs of the records are used to create feature vectors. "combined" means the input and output are concatenated into one string to create the feature vector. The σ is the distance standard deviation. The $c\sigma$ indicates the number (c) of standard deviations. For example, a classifier $f_{dist}(e_i|\text{"}examples'', \text{"}input'', 1.8)$ identifies the records whose inputs are more than 1.8 standard deviations (σ) away from the mean vector of the inputs of examples .

$$f_{dist}(e_i|r,t,c) = \begin{cases} -1 & d_{e_i,r} > c\sigma \\ 1 & d_{e_i,r} \leqslant c\sigma \end{cases} \quad (2)$$
$$where\, \sigma = \sqrt{\frac{1}{N}\sum_i (d_{e_i,r}-\mu)^2},\ \mu = \frac{1}{N}\sum_i d_{e_i,r}$$

Since parameter triple (r,t,c) has many configurations, our approach generates all configurations. Here, the value for c is selected a set of predefined decimal numbers. The approach uses each configuration to create a binary classifier and adds the classifier into the library of classifiers.

5.2.2 Classifiers based on the agreement of programs

This type of classifier identifies the records that consistent programs disagree about the transformation results. Typically, our approach can generate multiple programs that are consistent with given examples. Each program can be considered as an interpretation of the examples. The classifiers of this type maintain a set of programs that are consistent with the examples. The binary classifier identifies the records (output -1) that the programs produce different results for the same input. Providing examples for these records can help to clarify a user's intention and guide the system to converge to the correct programs. However, to build such classifiers, directly generating all the consistent programs and evaluating all these programs on records is expensive, as there are usually a large number of consistent programs given a few examples.

To reduce the computational cost in building this type of classifier, we exploit the fact the approach can independently generate the position programs. Our approach generates all the consistent position programs instead of the whole branch programs and evaluates these position programs on the records. This modification greatly reduces the number of programs that our approach is required to generate and evaluate, as the set of complete programs can be considered as a Cartesian product of sets of position programs. For example, in Figure 6, the start and end position of the substring "10" can both be represented using a set of programs. Every start position program can combine with any end position program to form a segment program. There would be 16 segment programs if there are 4 start position programs and 4 end position programs. But the total number of start and end position programs is only 8. Our approach

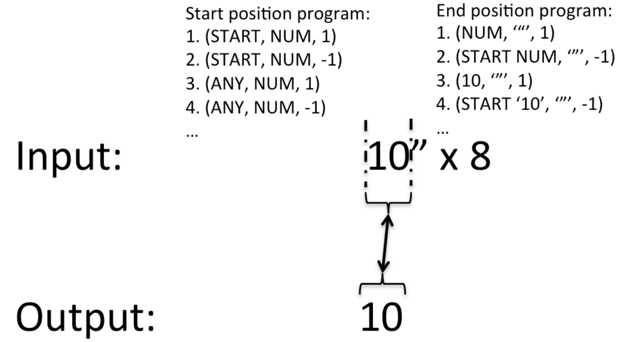

Figure 6: Candidate position programs for one segment program

only needs to generate and evaluate 8 position programs instead of 16 segment programs.

5.2.3 Classifiers based on the format ambiguity

This type of classifier aims to capture those records that are potentially labeled with the wrong format. The transformation program contains a conditional statement, which is used to recognize the format for a record before applying any transformation. Currently, we use a SVM multi-class classifier as the conditional statement as described in the previous work section. The SVM classifier not only classifies the records to their formats but also outputs the probability of the record belonging to that format. To identify the records that are potentially labeled incorrectly by the SVM classifier, our approach selects the records with the probability below a threshold θ. To select the right θ, our approach first creates a classifier using each θ in a predefined set and adds it into the classifier library. Later, our approach selects the classifier with the θ having the best performance described in the next section.

Combining classifiers using ADABOOST: we use ADABOOST [5] to combine classifiers above to create a meta-classifier ($F(e)$) for classifying whether a record (e) is transformed correctly as shown in Function 3. The meta-classifier outputs -1 for incorrect records and 1 for correct records. The output is 1, if the weighted sum of the output of a set of binary classifiers (f_i) is no less than 0; -1 if the sum is negative. During training ADABOOST iteratively selects the binary classifier (f_i) from a pool of classifiers described above to minimize the error on the misclassified training instances. It also assigns weights (w_i) to these classifiers indicating their importance in the final meta-classifier.

$$F(e) = sign(\sum_i w_i f_i(e)) \quad (3)$$

Our approach only uses ADABOOST to select the binary classifiers and learn their weights **once** to create the meta-classifier. Our approach uses this meta-classifier for all future transformations. Notably, the meta-classifier only defines the binary classifiers to be used and the weights for them. The approach still learns the binary classifiers constituting the meta-classifier for each specific transformation. Our approach learns the parameters for the binary classifiers (f_i) from the examples, records and consistent programs that are unique to each iteration. It combines these learned binary classifiers with the assigned weights to create the meta-classifier. The approach can use the meta-classifier with the learned binary classifiers to identify incorrect records.

5.2.4 Sorting the recommended records

As the approach recommends multiple records, it places the records that are more likely to be incorrect and contain more valuable information on the top of the recommendation area shown in Figure 5. This saves users' time in examining the recommended records and the system can also obtain more informative examples from the users. Our approach calculates a score for each record to measure how likely a record is incorrect and how much information it can provide in synthesizing the program.

The records with runtime errors are all incorrect. The score for these records is the number of failed subprograms including segment and position programs. A higher score means the approach can learn more information from this record, if the user provides an example for this record. The approach sorts these records in a descending order.

As to the records without runtime errors, we assume that the records that are more likely to be incorrect can provide more information for the system. The approach uses $-\sum_i w_i * f_i(x)$ in Function 3 as the score for each record. A higher score indicates more classifiers or the classifiers with heavier weights consider the record as an incorrect record. The approach sorts these records in descending order.

5.3 Minimal test set

We want to ensure that a user labels a minimum number of records, users are recommended to validate at least one record in a minimal set of records by either confirming the correctness of the record or entering a new example for that record. As mentioned before, there are multiple consistent programs given a set of examples. These programs conflict with each other as they generate different results on certain records. The minimal test set contains the records that these consistent programs disagree on the outputs. Ideally, we should ask users to verify the outputs of all the programs to identify the correct programs. However, fully generating all the programs and executing them on records is infeasible in practice based on two reasons: (1) the users are waiting for the responses on the fly and (2) the infinite number of conditional statements as there can be an infinite number of decision hyperplanes in the feature vector space. Our approach only generates all the consistent position programs and evaluates them on the records to approximate all the programs that should be tested. To identify the minimal test set, our approach simply uses the same set of records that are labeled as incorrect by the classifier based on the agreement of programs. Our approach highlights these records with blue borders in the GUI as seen in Figure 5. When the minimal test set is empty, there are not conflicting position programs.

6. EVALUATION

To evaluate the performance of our approach, we performed simulated experiments and a user study to compare our system with alternative approaches. Transforming a dataset usually requires several iterations in the evaluation. An *iteration* starts when the user provides an new example and ends when the system has learned the transformation program and applied the program to the rest of the data records.

6.1 Simulated experiment

There are two goals of this experiment: (1) test whether our recommendation can capture the incorrect records, and (2) test whether our approach can place at least one incorrect record on top so that users can easily notice these incorrect records.

6.1.1 Dataset

We used the 30 scenarios published in our previous work [21]. Each scenario contains about 350 records. The data was gathered from student mashup projects in a graduate-level course, which required the students to integrate data from multiple sources to create various applications. They were required to perform a variety of transformations to convert the data into the target formats. Each scenarios contains two columns of data. The first column shows the raw data and the second column shows the transformed data.

6.1.2 Experiment setup

To collect the training data for learning the meta-classifier, we should have both the transformation results and the labels to indicate whether these records are correct or not. Our approach first chose a record, provided the expected output and used it as an example. It learned the transformation program and applied the program to the rest of records. It compared the transformed data with the expected output and labeled each record as correct or incorrect. It also calculated the confidence of the conditional statement on each record. After collecting the data for the iteration, it started a new iteration by identifying the first incorrect record and providing an example for that record. The process ended when all the records were transformed correctly. Our approach collected training data from all iterations of the 30 scenarios. We divided all the scenarios into 5 groups and ran 5-fold cross-validation. Our approach trained the meta-classifiers using 4 groups of training data and tested the meta-classifier on the remaining group.

We used two metrics below to evaluate our approach and alternative approaches in each scenario:

- iteration accuracy: the percentage of iterations that our recommendations contain at least one incorrect record out of all the iterations having incorrect records.
- mean reciprocal rank (MRR): the average of the reciprocal rank of the first identified incorrect record. Q is the total number of iterations and $Rank_i$ is the index of the first incorrect record in the recommended list in the i-th iteration. If the recommendation fails to include the incorrect record and there exists one, the $\frac{1}{Rank_i}$ is set to 0.

$$MRR = \frac{1}{Q}\sum_{i=1}^{Q}\frac{1}{Rank_i}$$

We compared our current approach [1] with the state-of-the-art approach (Approach-β) [20] and a baseline approach. The **Approach-β** also provides recommendations for users to examine. There are two main differences between our approach and approach-β. First, our approach learns a meta-classifier from a pool of classifiers, while Approach-β only uses one classifier that is just one of the classifiers in our pool. Second, our approach recommends multiple records for users to review, while Approach-β only recommends one record. The **baseline** approach does not provide recommendation, which is also used by many existing applications. The baseline only randomly shuffles the transformed records for users to examine and users directly examine these shuffled records.

6.1.3 Results

As shown in Figure 7, our approach accurately captured the incorrect records in the recommendation. The average of iteration accuracy of our approach in all scenarios is 0.98 compared to 0.83 of the Approach-β. The iteration accuracy is left blank for baseline

[1] The documentation and code of our system are available on Github (https://github.com/areshand/Web-Karma)

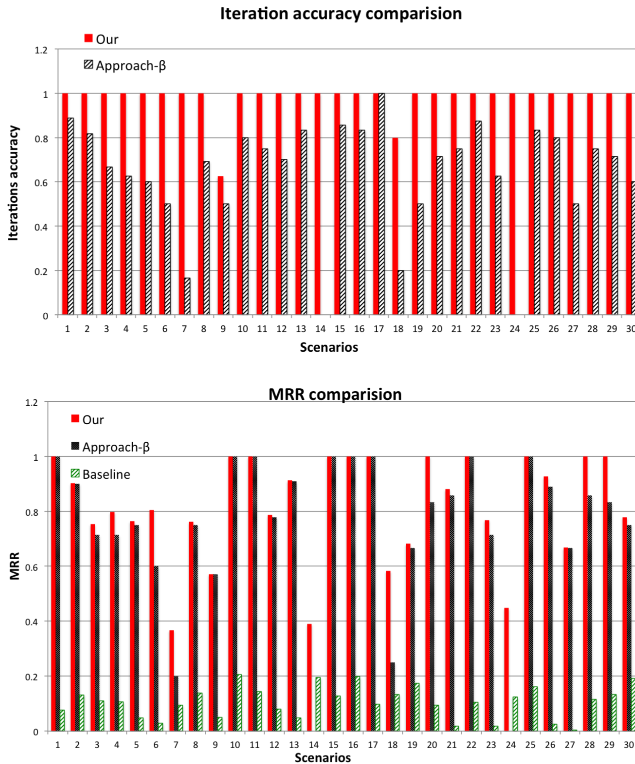

Figure 7: Comparison results

as the baseline approach does not provide any recommendations. The improvement is mainly due to two reasons. First, our approach recommended multiple records compared to that Approach-β only recommended one record in every iteration. Second, our meta-classifier is an ensemble of a library of classifiers. The classifier used in Approach-β is just one in the library. This ensemble of classifiers enables our approach to capture a boarder range of incorrect records. Only scenario 9 and 18 have iterations in which our approach failed to detect the incorrect records. These iterations require examples for unseen input formats that are similar to previous examples, which make the system fail to detect the difference. One example of the scenario 9 is shown in Table 1. The users intended to extract the full prices for student tuitions (1st and 2nd record). Since the third record only has the credit price, it should be transformed to "NULL". However, given only the first and second record as examples, our approach did not know the user required a different transformation for the third record, as it shared a very similar format with two previous records. Thus, our approach did not recommend the 3rd record as a potentially incorrect record for users to examine.

Our approach can place the incorrect records on top of the recommendation, as the average MRR of our approach in all scenarios is 0.75. It saved users' time from fully exploring a long list of records. The average MRR of Approach-β in all scenarios is 0.68. It did not place the incorrect record on top for most of iterations without runtime errors. The MRR of the baseline approach was calculated based on the index of the first incorrect record in the transformed records. We can see both our approach and Approach-β are well above the baseline, as the randomized shuffling can place the incorrect records in the middle of the list.

6.2 User study

We performed a user study to evaluate our approach in real use

Raw	Transformed
$33,926 per year (full-time)	$33,926
$42,296 per program (full-time)	$42,296
$1,286 per credit (full-time)	**$1,286**

Table 1: One typical example of a failed iteration

cases. The goal of this experiment is to test whether the users using our approach can achieve better correctnesses than the users of Approach-β with no more user effort.

6.2.1 Dataset

We collected 5 scenarios with about 4000 records for each scenario on average to evaluate the approaches. The first 2 records and the description of the scenarios are shown in Table 2 to demonstrate the transformation. These transformations involve transforming text into URIs by adding prefixes, replacing blank spaces with underscores or reordering substrings such as s1, s2 and s5. The rest of scenarios focus on extracting substrings from the inputs such as s3 and s4.

6.2.2 Experiment setup

We used three metrics to measure the user performance (1) correctness, which is the percentage of correct records when the users stopped transforming, (2) iteration time, which is the average time (in seconds) used by users in one *iteration* and (3) total time, which is the averaged total time (in seconds) used by users to transform a dataset.

We recruited 10 graduate students and divided them into two groups: $group_A$ and $group_B$. We asked users in $group_A$ to use our system and asked users in $group_B$ to use the Approach-β. We first asked them to work on one sample scenario to learn how to use the two systems. We then described the goal on the other 5 scenarios and asked them to transform the scenarios into the target formats.

We used the training data gathered in the simulated experiment to train our meta-classifier for recommending records. The p_{lower} and p_{upper} were set to 0.01 and 0.04. Our approach sampled 300 records in every iteration.

6.2.3 Results

The results of the user study are shown in Table 3. Our approach achieved a correctness higher than 0.99 in all scenarios. These correctnesses were within the user expected correctness range. Compared with Approach-β, we can see our approach also achieved better correctnesses in all 5 scenarios. Users in $group_A$ not only had higher correctness rates, but also used less time per iteration for 4 out of 5 scenarios and used the same amount of time on the first scenario. We performed paired one-tail t test for the hypothesis that *our approach uses less time per iteration and has higher correctness than Approach-β*. The result shows that the improvements are statistically significant ($p < 0.05$).

In the user study, we found that users using our approach used less total time in 3 out of 5 scenarios. This was due to users using our approach were willing to perform more iterations. We found that the users in $group_A$ provided more examples than users in $group_B$, as users in $group_A$ can simply click a button to confirm a correct record as a new example. The users confirmed several examples (2 - 5 examples) before stopping transformation. Thus, the number of examples (Example#) provided by the users in $group_A$ is higher than the numbers in $group_B$ as shown in Table 3. The $group_A$ users provided 11.1 examples and users in $group_B$ only provided 8.4 examples on average. Providing more examples gives users more

Scenario	description	Input	Output
s1	change into URI	WidthIN	http://qudt.org/vocab/unit#Inch
		HeightCM	http://qudt.org/vocab/unit#Centimeter
s2	change into URI	Dawson, William	William_Dawson
		Lauren Kalman	Lauren_Kalman
s3	extract issue date	Thor I#172 (January, 1970)	January, 1970
		Machine Man II#2	NULL
s4	extract first degree	7 x 9 in.	7
		6 13/16 x 8 7/8 in.	6 13/16
s5	change into URI	American	thesauri/nationality/American
		South African	thesauri/nationality/South_African

Table 2: Scenarios used in user study

Scenario	Our approach				Approach-β			
	Correctness	Iteration Time	Total Time	Example#	Correctness	Iteration Time	Total Time	Example#
s1	1	16	182.4	11.4	0.828	16	144	9
s2	0.998	17	227.8	13.4	0.994	26	234	9
s3	0.992	16	185.6	11.6	0.873	36	313.2	8.7
s4	0.997	14	196	14	0.983	17	187	11
s5	0.999	12	64.8	5.4	0.872	22	94.6	4.3
Average	0.997	15	171.32	11.1	0.91	23	194.56	8.4

Table 3: User study results

chances to refine the recommendation and examine more records, which in turn leads to a higher correctness rate. Moreover, the users in $group_A$ also used less time. We found when the recommendation contained incorrect records on top, it largely reduced the time users used to examine the results compared to the time spent by users to directly examine the results when the recommendation failed to capture the incorrect records.

In the user evaluation, we observed that our approach recommended a large number of records for users to examine for certain iterations. For most of the iterations, the recommendations had the incorrect records on top of the recommended list and the users identified these incorrect records. For the remaining iterations, users missed certain incorrect records in the recommendation. But the users still obtained correctnesses of final results satisfying the requirement specified in the experiment setup (p_{low}=0.01 and $p_{upper} = 0.04$), as the numbers of unidentified incorrect records were smaller than the allowable number Z_α (7).

We also asked users for their feedbacks on the color-coding of the results. All users said the color-coding helps them to identify incorrect records by noticing some irregular color patterns. Some users also mentioned that it would be more helpful if different formats can also be colored differently

7. RELATED WORK

Recently, programming-by-example (PBE) approaches [11, 8, 6] have proven to be effective in generating transformation programs for simple scenarios without coding. Recently, FlashFill [6], which is an example PBE system, is already integrated into Excel 2013 to help users transform the data. Here, we only review the closely related work focusing on verifying the correctness of user-generated programs [10]. The techniques used to verifying the correctness of programs can generally be categorized into 3 types: (1) testing the users generated programs following formal software testing strategies, (2) detecting potentially incorrect results and asking users to verify, and (3) visualizing the results to reveal irregularities and outliers. Existing approaches usually combine techniques from multiple categories to maximize the chance of detecting errors in the generated programs.

As the users are often overconfident with the correctness of their results, a program testing plan can be designed and users are provided with the feedback of how complete tests have been done. The testing feedback may motivate the users to test the results more thoroughly. The WYSWYT ("What You See is What You Test") [18, 16, 17] describes an approach for users to verify their spreadsheet programs. To test the spreadsheet programs, the approach asks users to provide test cases through validating the correctness of certain values. To ensure it has obtained enough test cases, the approach developed a criterion called definition-use coverage to find the values that users should validate. The criterion is essentially developed based on the data flow adequacy [15] to test the correctness of cell references in spreadsheets. Our approach is inspired by this approach to recommend users to examine a minimal set of records to address the overconfidence problem. However, our approach only focuses on transforming a column of data into another other column. There are not cell dependences we can leverage in deciding which records to examine. Moreover, recommended records can also help users to explore the dataset to allow them to notice the unexpected inputs to refine their programs, which is more related to help users understand the task requirements rather than merely testing. The approach [1] introduced assertions into the spreadsheet program testing. It allows users to specify their expectation and converts them into assertions. The assertions specify allowed cell values in the form of Boolean expressions. Whenever a conflict between an assertion and a cell value happens, the cell value is shown to the users. Our approach is different from this approach as our approach only asks user to provide examples.

Many approaches have been developed to identify the potentially incorrect records. The approach [6] can highlight the entries, which have two or more alternative transformed results. This method generates multiple programs and evaluates these programs on all the records to identify these records with different results. This method is equivalent to one of the classifiers used in our approach to identify incorrect records. Our approach supports more methods for

detecting potentially incorrect records and combines these different methods using a meta-classifier. Moreover, our approach places all identified records together in one area so that users do need to go through all the records to find them. LAPIS [13] highlights the texts that have potentially incorrect matches. Their approach identifies the matches that are different from the majority of matches. Wolfman, et al. [19] extends the approach [11] by reducing the user effort using a mixed initiative approach combining several interaction modes. Wu, et al. [22] recommends only one example for users to examine and the recommendation is only based on the distance from the records to the examples. Compared to approaches above, our approach learns more powerful rules that can identify incorrect records in various scenarios. Our approach recommends and sorts multiple records so that the users have a chance of catching the incorrect records without examining many records.

Different visualization methods have been proposed to help users gain insight into both the data and learned program. OpenRefine [7] supports a large number of ways to visualize the data and it also allows users to customize the way of visualizing the data, such as histogram, facet graphs and etc. Users may notice some irregular pattern when watching these visualizations to uncover potentially incorrect results. Data Wrangler [8] translates transformation programs into natural language so that end-users can read these programs. They can then verify whether the programs are the same as expected. Our approach is orthogonal to the approaches above. Our approach color-codes the substrings in the records so that users can understand how the raw inputs are transformed into the outputs.

8. CONCLUSION AND FUTURE WORK

PBE approaches should enable users to obtain the correct results on large datasets. We present an approach designed to help users of PBE approaches to obtain correct results with minimal efforts. The approach samples a set of records for automatic inspections. It then uses an ensemble of classifiers to identify the potential incorrect records from the sampled records and presents these potentially incorrect records for the users to examine. User can provide labels for these records by either entering examples for incorrect records or confirming a record is correct. The approach learns from these labels to update the recommendation. We performed a simulated experiment on collected scenarios. The results shows that our approach can identify incorrect records in nearly all the iterations and can also place the incorrect records on top so that users can easily notice these incorrect records. We also performed a user study. Its results show that our approach can save users time in obtaining significantly higher correctnesses compared to alternative approaches.

In the future, we plan to introduce some existing orthogonal methods for verifying the correctness of results and programs. For example, we can use histogram to visualize both the raw data and transformed data to allow users to obtain better overview of the data. Translating the transformation programs to readable natural language texts can also help users to gain insight into the generated programs. Thus, they can verify whether the programs work in a way as they expect.

9. REFERENCES

[1] Margaret M. Burnett, Curtis R. Cook, Omkar Pendse, Gregg Rothermel, Jay Summet, and Chris S. Wallace. 2003. End-User Software Engineering with Assertions in the Spreadsheet Paradigm. In *ICSE*.

[2] Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology* (2011). Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[3] M.M. Desu and D. Raghavarao (Eds.). 1990. *Sample size methodology*. Academic press Inc.

[4] Ronen Feldman and James Sanger. 2006. *Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data*. Cambridge University Press, New York, NY, USA.

[5] Yoav Freund, Robert E Schapire, and others. 1996. Experiments with a new boosting algorithm. In *ICML*, Vol. 96. 148–156.

[6] Sumit Gulwani. 2011. Automating String Processing in Spreadsheets Using Input-output Examples. In *POPL*.

[7] David F Huynh and Mazzocchi Stefano. *OpenRefine http://openrefine.org*. `http://openrefine.org`

[8] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: interactive visual specification of data transformation scripts. In *CHI*.

[9] Emanuel Kitzelmann and Ute Schmid. 2006. Inductive Synthesis of Functional Programs: An Explanation Based Generalization Approach. *Journal of Machine Learning Research* (2006).

[10] Andrew J. Ko, Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi, Joseph Lawrance, Henry Lieberman, Brad Myers, Mary Beth Rosson, Gregg Rothermel, Mary Shaw, and Susan Wiedenbeck. 2011. The State of the Art in End-user Software Engineering. *ACM Comput. Surv.* (2011).

[11] Tessa Lau, Steven A. Wolfman, Pedro Domingos, and Daniel S. Weld. 2003. Programming by Demonstration Using Version Space Algebra. *Mach. Learn.* (2003), 46.

[12] Henry Lieberman (Ed.). 2001. *Your Wish is My Command: Programming by Example*. Morgan Kaufmann Publishers Inc.

[13] Robert C. Miller and Brad A. Myers. 2001. Outlier Finding: Focusing User Attention on Possible Errors. In *UIST*.

[14] Raymond R. Panko. 1998. What We Know About Spreadsheet Errors. *J. End User Comput.* (1998).

[15] Sandra Rapps and Elaine J. Weyuker. 1985. Selecting Software Test Data Using Data Flow Information. *IEEE Trans. Software Eng.* (1985).

[16] Gregg Rothermel, Margaret Burnett, Lixin Li, Christopher Dupuis, and Andrei Sheretov. 2001. A Methodology for Testing Spreadsheets. *ACM Trans. Softw. Eng. Methodol.* (2001).

[17] Gregg Rothermel, Lixin Li, Christopher DuPuis, and Margaret Burnett. 1997. *What You See is What You Test: A Methodology for Testing Form-based Visual Programs*. Technical Report.

[18] Karen J. Rothermel, Curtis R. Cook, Margaret M. Burnett, Justin Schonfeld, T. R. G. Green, and Gregg Rothermel. 2000. WYSIWYT Testing in the Spreadsheet Paradigm: An Empirical Evaluation. In *ICSE*.

[19] Steven A. Wolfman, Tessa A. Lau, Pedro Domingos, and Daniel S. Weld. 2001. Mixed initiative interfaces for learning tasks: SMARTedit talks back. In *IUI*.

[20] Bo Wu and Craig A. Knoblock. 2014. Iteratively Learning Conditional Statements in Transforming Data by Example. In *Proceedings of the First Workshop on Data Integration and Application at the 2014 IEEE International Conference on Data Mining*. IEEE.

[21] Bo Wu and Craig A. Knoblock. 2015. An Iterative Approach to Synthesize Data Transformation Programs. In *IJCAI*.

[22] Bo Wu, Pedro Szekely, and Craig A. Knoblock. 2014. Minimizing User Effort in Transforming Data by Example. In *IUI*.