UNSUPERVISED INFERENCE OF DATA FORMATS IN HUMAN-READABLE NOTATION

Christopher Scaffidi
Institute for Software Research, School of Computer Science, Carnegie Mellon University
5000 Forbes Ave, Pittsburgh, PA, 15217, United States

Keywords: Data integration, unsupervised learning, outlier finding, data formats, spreadsheets, databases, web services.

Abstract: One common approach to validating data such as email addresses and phone numbers is to check whether values conform to some desired data format. Unfortunately, users may need to learn a specialized notation such as regular expressions to specify the format, and even after learning the notation, specifying formats may take substantial time. To address these problems, this paper introduces Topei, a system that infers a format from an unlabeled collection of examples (which may contain errors). The generated format is presented as understandable English, so users can review and customize the format. In addition, the format can be used to automatically check data against the format and find outliers that do not match. Topei shows substantially higher precision and recall than an alternate algorithm (Lapis) on test data. Topei's usefulness is demonstrated by integrating it with spreadsheet, database, and web services systems.

1 INTRODUCTION

Information systems typically use semantics-free character strings to store small semi-structured data, including postal codes, person names, and URLs. The usual approach to ensuring strings' validity is to write programs such as database triggers to check if values match a desired format.

From a human-computer interaction standpoint, two major challenges interfere with this approach.

First, users must learn specialized notation to write a format. Regular expressions are probably the most common notation for specifying simple formats such as email addresses, but the notation is hard for users to master (Blackwell, 2001). Even experienced programmers recognize that regular expressions are hard to learn. For example, the top few results in a Google search for "regular expressions" will return comments such as, "Do not worry if the above example or the quick start make little sense" and, "Sometimes you have a programming problem and it seems like the best solution is to use regular expressions; now you have two problems."

Second, even after a user learns a notation, specifying simple constraints can be time-consuming and complex. For example, IPv4 addresses have four integers, separated by periods, and each integer can range from 0 through 255. Just the regular expression for an integer from 0 through 255 is lengthy: `25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?`

The Tope system begins to address these challenges by presenting formats in understandable English, so users do not need to learn a specialized notation (Scaffidi, 2007). To date, the system's main weakness is its limited support for helping users to get started: Although users do not need to learn specialized notation in order to specify a format, there is still the cognitive work of examining data, breaking it into parts, and representing the parts in the format.

This paper addresses this limitation by presenting Topei, an algorithm that examines examples and infers a format. The process involves three steps:

1. The user provides examples to Topei, which infers a format.
2. The user reviews the format in the Tope system's editor tool. Although the format is internally represented in an XML notation, the user never sees this, as the editor presents the format to the user as a set of statements in English. After any customization by the user, the editor saves the format in a file.
3. The user validates strings by passing them through the Tope system's parser tool, which attempts to parse each string using an augmented

Scaffidi C. (2007).
UNSUPERVISED INFERENCE OF DATA FORMATS IN HUMAN-READABLE NOTATION.
In *Proceedings of the Ninth International Conference on Enterprise Information Systems - HCI*, pages 236-241
DOI: 10.5220/0002347902360241

context-free grammar (CFG) automatically derived from the format.

In short, Topei provides automatic inference of formats, saving users time and cognitive load, while retaining the advantage of editability. This is an important advantage, since every inference algorithm can make errors, so users should be able to understand the inferred format and make corrections if needed. In particular, the sample data may not exemplify some aspects of the format, so users must manually add these aspects.

An important aspect of data validation is outlier finding, which involves identifying invalid data values that are mixed among a set of valid values. This paper evaluates Topei as an outlier finder by comparing it to the Lapis system (Miller & Myers, 2001), revealing that Topei has higher precision and recall.

Section 2 reviews related work on inference and data formats. Section 3 describes the integration of Topei with spreadsheets, databases, and web services. Section 4 discusses the inference algorithm and evaluates its accuracy and limitations.

2 RELATED WORK

This paper's contribution, Topei, has these traits:

- It infers a data format.
- It is unsupervised (uses unlabeled examples).
- Its formats permit soft constraints.
- Its formats are presentable in English.
- Its formats are useful for outlier finding.

Other algorithms share some traits but lack others:

Many machine learning algorithms train a recognizer to notice certain features, enabling the recognizer to identify outliers lacking those features (Mitchell, 1997). However, such algorithms generally do not infer a human-editable data format.

Other algorithms generate formats in specialized notation. These include (Blackwell, 2001), (Lerman & Minton, 2000), (Lieberman et al., 2001), and (Nardi et al., 1998), all of which use regular expressions or CFGs. One last system, Lapis, has a specialized notation; it is like Topei in that it is intended to be highly intuitive (Miller & Myers, 2001). Yet using its notation still takes practice, since expressing simple constructs can be cumbersome. For instance, the Lapis library defines a day (in a date) as:

```
@DayOfMonth is Number equal to
          /[12][0-9]|3[01]|0?[1-9]/
ignoring nothing
```

In contrast, Topei infers formats that are presented to users as understandable English, and it supports "soft" (non-mandatory) constraints on data.

Several tools recognize or manipulate some of the same kinds of data as Topei (Hong & Wong, 2006) (Pandit & Kalbag, 1997) (Stylos et al., 2004). However, these tools' formats are hard-coded and cannot be extended or customized by end users.

3 INTERFACES

Topei supports a graphical user interface (GUI), a command-line interface (CLI), and an application programming interface (API).

3.1 Spreadsheet Assertions

Spreadsheets are an important information repository in organizations, but most spreadsheets contain errors (Panko, 1998). Topei helps users validate strings, which comprise nearly 40% of all spreadsheet cells (Fisher, 2004). Users can create a format from examples in one or more cells and attach the format to those cells; if a cell's contents do not match the format, then the cell is highlighted as a possible error that the user can either fix or ignore.

For instance, consider Figure 1, part of a sample spreadsheet from Microsoft. The screenshot includes the Topei "Patterns" toolbar (an Excel plug-in).

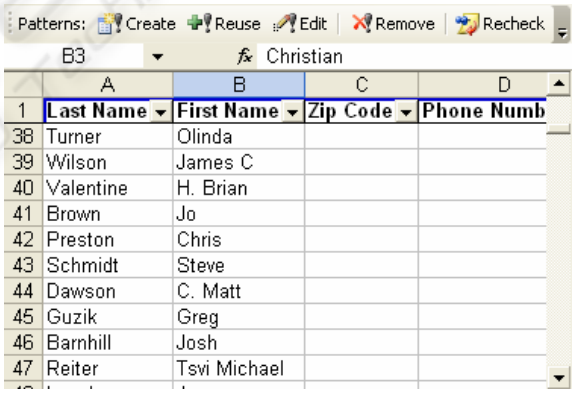

Figure 1: Sample Excel spreadsheet and Topei plug-in.

Though the first 38 rows each contain a person first name in column B, some rows also have a middle initial or name. Such outliers are semantically inconsistent with most cells, and the surface format reflects this inconsistency. In particular, most outliers have an extra uppercase letter, a space, and a period.

To find such errors, the user highlights column B and clicks the toolbar's "Create" button. Topei infers a format, which the editor loads and displays (Figure 2). The user can add, edit, and remove format parts and/or constraints, and constraints can be marked as

"soft constraints" (i.e.: not "always" true). For more details on the editor, refer to (Scaffidi, 2007).

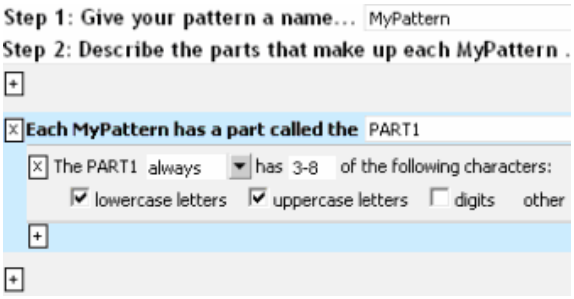

Figure 2: Editor for reviewing / customizing formats.

In this example, the user might rename "MyPattern" to "First Name" and "PART1" to "name," then broaden the length range from 3-8 characters to 2-10.

For more complex structured data, the editor would be initialized with more parts, and perhaps with constraints. For example, if the data were dates in `DD/MM/YYYY` form, then the editor would show three parts (the day, the month, and the year, separated by /), and each part could have constraints (e.g., restricting the day to 1-31 and the month to 1-12).

After the user saves the format, the GUI uses the parser to test each selected cell. Any cells that do not match the format, including all constraints, are marked with a red triangle; mousing over the cell brings up an error message. The plug-in integrates with Excel's Reviewing functionality to display this error feedback. Thus, the user can page through errors using the existing "comments" browser within Excel and can ignore messages if desired.

3.2 Database Integration

Topei supports a CLI, demonstrated here with the sample `Person` table provided by Microsoft. First, the user exports example data from the database. The SQL Server command is:

```
osql -d AdventureWorks -E -h-1 -o
"c:\tmp.txt" -Q "SET NOCOUNT ON
select PostalCode from Person.Address"
```

Next, the user loads the examples into Topei:

```
topei.exe c:\tmp.txt
```

Topei infers a format that the editor displays for review and customization. After the user saves the format, it can be applied to data. A SQL Server plug-in provides a new function, `checkPattern`, which loads the format, checks the string against the format, and returns a score between 0 and 1. Here, 1 indicates that the string satisfies the format, and 0 indicates that it does not; if the string violates soft constraints, then `checkPattern` returns a score between 0 and 1. (By passing in a non-negative threshold, the user can make the function throw an exception if the score is less than the threshold.)

For example, based on examples from the `Person` table, Topei infers that the `varchar PostalCode` has one five-digit part. If the user saves this format to a file called "postal.xml," then the column's data can be checked using the following SQL:

```
select PostalCode from Person.Address
where dbo.checkPattern('postal.xml',
PostalCode, -1) < 1
```

Executing this query on the sample Microsoft data returns 7276 rows (out of 19614 rows in the table). Most erroneous values contain only four digits, rather than the five that a US zip code must have. Others are British postal codes; the user could allow for these by creating a second format and doing a query for rows that match neither format.

The SQL shown above can be executed in an interactive query window or in a command-line batch script. A trigger could use a similar query to prevent invalid values from entering the table.

Tope data formats are portable. For example, the user could create a format with the GUI, and then use it to check data with the CLI or API.

3.3 Web Service Validation

Finally, Topei supports a C# API that can be called from programs that manipulate text, XML, or other data. This is useful when a programmer needs to validate data coming from a web service.

For example, in the Google Base web application, users can create records with arbitrary attribute-value pairs. For instance, a `Job` record might have a `contact` field containing a phone number. Unfortunately, Google Base does not validate these values, so they are sometimes malformed. Consequently, when a program retrieves jobs from the Google Base XML feed, some attributes may have invalid values.

One line of code suffices to infer a format (`xt`) from an array of example values (`ex`). A second line creates a parser and validates a string (`str`):

```
string xt=XtopeInference.MakeXtope(ex);
float  sc=Parser.init(xt).check(str);
```

Like the `checkPattern` stored procedure, the `check` function returns a score from 0 to 1 to indicate how well the string matches the format.

4 TOPEI: FORMAT INFERENCE

This section describes the format model, the inference algorithm, and the evaluation of Topei.

4.1 Overview of Formats

The following discussion is a summary. For details on formats and parsing, refer to (Scaffidi, 2007).

Each format contains one or more parts, each of which has zero or more constraints.

The *Pattern* constraint specifies a part's length (or a range of valid lengths) and the character class of its constituents. The character class can be `0` (digits), `a` (lowercase letters) or `A` (uppercase letters), or it can be a composite character class. For example, a mixture of uppercase letters and digits would be expressed using the composite class `A0`.

Other constraints express additional semantics:

- *Wrapped*: neighboring separator
- *Numeric*: numeric inequality or equality
- *Substring*: starts or ends with a string
- *Literal*: in a certain set of valid strings

For example, a date in `DD/MM/YY` format would have three parts, each of which would have a *Pattern* constraint with length 2 and character class `0`. Each part would be restricted to a numeric range with *Numeric* constraints, and each slash would be associated with the preceding part using a *Wrapped* constraint.

Each constraint can have an attribute in the range 0 through 100 to express a confidence that the constraint should be true. Constraints with confidence 100 (the default) must always be true; those with confidence 0 represent statements that should never be true. For example, a *Numeric* constraint with confidence 0 can be used to specify, "the day *never* is > 31." Constraints with confidence between 0 and 100 are soft. If a string violates soft constraints, then the parser returns a score between 0 and 1.

Each constraint has a straightforward English equivalent. This equivalence is the key to presenting formats in human-readable form, as Figure 2 depicts. A harder challenge is to examine example data and infer a format in the notation. This is achieved with Topei, as described below.

4.2 Inference Algorithm

Format inference has two phases. First, Topei identifies the format's parts and each part's composite character class. Second, Topei identifies constraints.

The discussion below uses six example email addresses to demonstrate the algorithm:

```
apple@gmail.com
banana1@hotmail.com
carrot@company.com
DATE@UNIVERSITY.edu
eggplant@firm-name.com
fig.plant@mail.univ.edu
```

For each example, Topei replaces each letter and digit with its character class, and then collapses runs of identical classes. The collapsed string is called a "signature." The email addresses yield five unique signatures:

```
a@a.a
a0@a.a
A@A.a
a@a-a.a
a.a@a.a.a
```

To finish this phase, Topei uses separators to identify and align parts, and it abstracts each part's character class to the least general composite character class that covers the examples of that part. In the email address examples, the first three signatures have three parts and identical separators (one @ followed by a period), so they are aligned and abstracted:

This concludes the first phase of inference, which has generated a primary format (`aA0@aA.a`) with three parts. Two email addresses have signatures that are incompatible with this format, so Topei does not include them in the primary format. Instead, Topei uses them for secondary formats that can be retrieved through the API. The user can also customize the primary format in the editor so that omitted examples are covered; whether the user decides to do this depends on the data's semantics.

In the next phase, Topei generates constraints for each format's parts. In the example signature above, the two punctuation marks (`@` and `.`) are separators, each of which corresponds to a *Wrapped* constraint.

Next, for each part, Topei histograms the lengths of examples. It selects the lengths that together cover at least 95% of the examples, then expresses this set of lengths as a range over a composite character class (e.g.: part 1 of the email addresses has 4-7 `aA0`, part 2 has 5-10 `aA`, and part 3 has `3 a`). This yields a *Pattern* constraint.

Then, Topei histograms the examples' text and searches for a set of 3 or fewer strings that together cover at least 95% of the examples (e.g.: `edu` and `com` cover the email addresses' part 3). If Topei finds such a set, it creates a *Literal* constraint, requiring that the part's text must be in the set.

If no *Literal* constraint is created, then Topei tries constraining the part to a numeric range. It converts the part examples to numbers and histograms them. If Topei finds a range of numeric values that covers at

least 95% of all examples (including non-numeric strings), then it constrains the part to that range (using *Numeric* constraints).

If no *Literal* or *Numeric* constraints are created, then Topei tries "begins with" and/or "ends with" constraints. Again using histograms, it tries to find the longest substring that prefixes at least 95% of examples; if such a prefix exists, and then Topei creates a corresponding *Substring* constraint. Likewise, it creates a suffix *Substring* constraint if some substring suffixes at least 95% of examples.

Topei sets the confidence to 100 for *Pattern* constraints but sets other constraints' confidence to 60 (to help prevent over-fitting). When reviewing the format, the user can tighten constraints to a confidence of 100.

In a few details, this algorithm resembles existing algorithms. For example, Topei uses character classes similar to those of (Lerman & Minton, 2000). However, in order to decide whether to generalize to a composite character class, Lerman & Minton assume that string values are independently sampled from a probability distribution. In practice, if a value appears in one spreadsheet cell or database row, then that value also tends to appear in nearby cells or rows. Thus, real data violate Lerman & Minton's independence assumption. Therefore, rather than presupposing the existence of a particular probability distribution, Topei generalizes to the least general composite character class and then uses the editor to display the format so the user can review and customize the format. (In addition, their algorithm does not infer numeric constraints, and their formats are not portable, as they are specialized to matching strings mixed with HTML.)

4.3 Validation: Comparison to Lapis

Topei is evaluated by using it and Lapis 1.2 (Miller & Myers, 2001) to locate outliers in test data, then manually examining the data to find true outliers. Topei shows higher precision and recall than Lapis.

Test data come from the EUSES spreadsheet corpus (Fisher & Rothermel, 2004), which contains 4498 spreadsheets culled from the web. Two kinds of test data are used—country names and American phone numbers—in order to test Topei on unstructured and structured data. A column of spreadsheet data is included in the country test data if it has at least 20 cells, and at least 1 cell equals "Portugal," and the first cell contains the word "country." A column is included in the phone number test data if it has at least 20 cells with exactly 10 digits, and at least 2/3 of cells have exactly 10 digits, and the first cell contains the word "phone." Visually examining the data reveals that one phone column is a mixture of many formats; discarding this column leaves 1124 countries in 7 columns and 6288 phone numbers in 37 columns.

To locate outliers for each spreadsheet column, Topei is used to infer a format, and the parser is used to check each of the column's cells against the format. If the parser returns a score less than 1 for a cell, then it is flagged as an outlier.

To locate outliers for each column, Lapis maps cells into a feature space, and then computes the distance of each cell from the centroid. Lapis finds the maximum of these distances, `MAXDIST`, and any cell value that is at least `MAXDIST/2` from the centroid is flagged as an outlier. There is one extra heuristic: if more than half of the cells in the column would be flagged as outliers, or if 10 or more would be flagged, then Lapis flags *none* of them as outliers. (Lapis comes pre-equipped with hand-coded country and phone number formats, which are disabled for this test, though the formats for numbers, words, delimiters, and other layout remain enabled.)

To locate true outliers, each column is manually examined. Countries are labeled as outliers if they contain abbreviations, misspellings, or a different name than the one usually used by English-speakers (e.g.: Côte d'Ivoire is an outlier in one case; in fact, it is that column's only value spelled in French), though "Brasil," "US," "USA," "UK," and various pre-reunification names for Germany are not labeled as outliers, as they are commonly used. Phone numbers are labeled as outliers if their format differs from the column's main format (e.g., insertion of extra spaces, or using periods as separators when most cells use hyphens), or if their area codes or exchanges are not in the list of valid values from www.nanpa.org. These criteria identify 92 country names (8%) and 1669 phone numbers (26%) as true outliers.

In outlier finding, precision is the number of cells correctly flagged as outliers by an algorithm, divided by the total number of cells flagged as outliers. Recall is the number of cells correctly flagged as outliers, divided by the number of true outliers. As shown in Table 1, Topei demonstrates higher precision and recall than Lapis.

Table 1: Precision and recall of Topei and Lapis.

Task	Algorithm	Precision (%)	Recall (%)
Country	Topei	56.5	94.6
Country	Lapis	46.7	7.6
Phone	Topei	97.7	99.8
Phone	Lapis	44.0	2.4

The Lapis heuristics are intentionally biased toward low recall, as the algorithm's designers believed that "highlighting a large number of outliers is un-

helpful to the user, since the user must examine each one" (Miller & Myers, 2001). Recall can be raised by lowering the outlier threshold from its default value (½ of the maximal distance to the centroid) and eliminating the heuristic that caps the number of outliers per column. As Figure 3 shows, this raises Lapis's recall as high as 70% and 9% for the tasks, respectively, with little loss of precision. Yet these scores remain much lower than those of Topei.

In these tests, Topei generates few soft constraints, so the parser usually returns scores of 0 or 1. Thus, reducing Topei's threshold from 1 only alters its precision and recall by 2%.

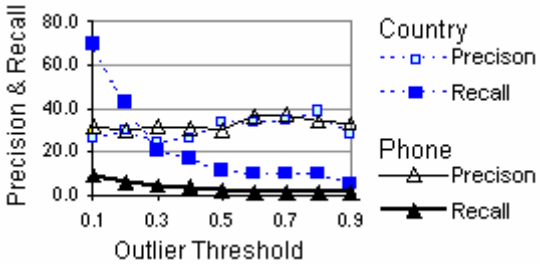

Figure 3: Performance of modified Lapis with outlier threshold expressed as a fraction of `MAXDIST` to centroid.

4.4 Limitations and Future Work

Topei performs better than Lapis at finding outliers in spreadsheet cells. This is probably due to the algorithms' different inductive biases: the notation used to express formats in Lapis is intended for describing regions of text in large documents. Unsurprisingly, Lapis performs better on unstructured data (countries) than structured data (phone numbers). In contrast, Topei is oriented toward single data values (such as spreadsheet cells), particularly those with separator-delimited parts.

Still, Topei makes mistakes. For the phone task, most mistakes occur because Topei fails to notice invalid area codes; in these cases, not enough examples are present to lead Topei to create soft numeric range constraints. In the country task, most mistakes occur because a valid country name contains two words, and Topei currently does not infer word repetition. Adding heuristics that are more sophisticated might help to reduce these mistakes. Comparison with other systems (besides Lapis) may inspire additional ideas for improvement.

Future work could make Topei more flexible by adding support for non-English letters (such as letters with accents). The editor and parser support Unicode but would require interface changes.

Like Lapis, Topei is designed to have O(n) computational complexity, where n is the number of examples. Preliminary tests indicate that the implementation does demonstrate O(n) performance, though additional evaluation would be desirable.

The usability of Topei's user interfaces has not yet been evaluated. Such an evaluation may reveal limitations to how well users understand Topei and how successfully they apply formats to spreadsheets, databases, web services, and other systems.

ACKNOWLEDGEMENTS

This work was funded in part by the National Science Foundation (ITR-0325273) via the EUSES Consortium and by the National Science Foundation under Grant CCF-0438929. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the sponsors.

REFERENCES

Blackwell, B., 2001. SWYN: A Visual Representation for Regular Expressions. *Your Wish is My Command: Programming by Example*, pp. 245-270.

Fisher, M., Rothermel, G., 2004. *The EUSES Spreadsheet Corpus: A Shared Resource for Supporting Experimentation with Spreadsheet Dependability Mechanisms*, Tech. Report 04-12-03, Univ. Nebraska-Lincoln.

Hong, J., Wong, J., 2006. Marmite: End-User Programming for the Web, *Proc. CHI'06 Conf. on Human Factors in Computing Systems*, pp. 1541-1546.

Lerman, K., Minton, S., 2000. Learning the Common Structure of Data, *Proc. AAAI-2000*, pp. 609-614.

Lieberman, H., Nardi, B., Wright, D., 2001. Training Agents to Recognize Text by Example, *Auton. Agents and Multi-Agent Systems*, vol. 4, no. 1, pp. 79-92.

Miller, R., Myers, B., 2001. Outlier Finding: Focusing User Attention on Possible Errors. *Proc. 14th Annual Symp. on User Interface Software and Technology*, pp. 81-90.

Mitchell, T., 1997. *Machine Learning*, McGraw Hill.

Nardi, B., Miller, J., Wright, D., 1998. Collaborative, Programmable Intelligent Agents. *Comm. ACM*, vol. 41, no. 3, pp. 96-104.

Panko, R., 1998, What We Know about Spreadsheet Errors, *J. End User Computing*, vol. 10, no. 2, pp. 15-21.

Pandit, M., Kalbag, S., 1997. The Selection Recognition Agent: Instant Access to Relevant Information and Operations. *Proc. 2nd Intl. Conf. on Intelligent User Interfaces*, pp. 47-52.

Scaffidi, C., Myers, B., Shaw, M., 2007. *The Topes Format Editor and Parser*, Tech. Report CMU-ISRI-07-104/CMU-HCII-07-100, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Stylos, J., Myers, B., Faulring, A., 2004. Citrine: Providing Intelligent Copy-and-Paste. *Proc. 17th Annual Symp. on User Interface Software and Technology*, pp. 185-188.