Does Anybody Really Know What Time It Is?

Automating the Extraction of Date Scalars

Richard Wesley Vidya Setlur Dan Cory

ABSTRACT

With the advent of modern data visualization tools, data preparation has become a bottleneck for analytic workflows. Users who are already engaged in data analysis prefer to stay in the visualization environment for cleaning and preparation tasks whenever possible, both to preserve analytic flow and to take advantage of the visualization environment to spot inconsistent data. This has led many visualization environments to include simple data preparation functions such as scalar parsing, pattern matching and categorical binning in their analytic toolkits.

One of the most common parsing tasks is extracting date and time data from string representations. Several databases include date parsing "mini-languages" to cover the wide range of possible formats. Tableau has recently provided a DATEPARSE function with a single syntax that is translated into the nearest equivalent in the underlying database. Subsequent analysis of customer usage of this function shows that the parsing language syntax was difficult for users to master.

In this paper, we present two algorithms for automatically deriving date format strings from a column of data in our syntax, one based on minimum entropy and one based on natural language modeling. Both have similar accuracies of over 90% on a large corpus of date columns extracted from Tableau Public and are in substantial agreement with each other. The minimal entropy approach can also produce results below the user's perceptual threshold, making it suitable for interactive work.

Categories and Subject Descriptors

D.3.3 [**Data Processing**]: Data Cleaning, Natural Language Processing

General Terms

Algorithms, Performance

Keywords

Date parsing, automated cleaning, structure extraction

1. INTRODUCTION

In recent years, there has been a growth of interest in data visualization technologies for human-assisted data analysis using systems such as [1,10,11]. While computers can provide high-speed and high-volume data processing, humans have domain knowledge and the ability to process data in parallel, notably by using our visual systems. Most importantly, humans provide the definition of what is valuable in an analysis. Accordingly, human/computer analytic systems are essential to extracting knowledge of value to humans and their societies from the large amounts of data being generated today.

1.1 Interactivity

Visualization systems are most effective when they are interactive, thereby allowing a user to explore data and connect it to their domain knowledge and sense of what is important without breaking cognitive flow. In recent years, a number of such systems have been developed, both by the academic community and by the commercial sector. Exploration of data consists not only in creating visual displays, but also in creating and modifying domain-specific computations. Consequently, most data visualization systems include facilities for defining such calculations as part of the data model being analyzed. The most effective systems allow users to define these calculations as part of the analytic interaction, which permits the user to stay in the flow of analysis [9].

During the analytic process, a user may discover that parts of the data are not yet suitable for analysis. Solutions to this problem are often provided by data preparation tools external to the visual analysis environment, which requires the user to break their analytic flow, launch another tool and reprocess their data before returning to their analysis. If the user does not own this process (e.g. it is the responsibility of another department), then there can be significant delays (including "never.") More subtly, the result of updated external processing may not be compatible with the user's existing work, which can lead to more time lost reconciling the new data model with the existing analysis.

From the user's perspective, the boundary between preparation and analysis is not nearly so clean cut. Bad data is often discovered using visual analysis techniques (e.g. histograms or scatter plots) and it is most natural for the user to "clean what she sees" instead of switching to a second tool. This leads to an "adaptive" process whereby users will prefer leveraging existing tools in the analytics environment (no matter how well suited to the task) over switching to another application. Thus a well-designed interactive visual analysis environment will provide tools that enable users to perform such cleaning tasks as interactively as possible.

1.2 The Tableau Ecosystem

In this paper, we shall be looking at an example of this problem in the context of the Tableau system. Tableau is a commercial visual analysis environment derived from the Polaris [1] system developed at Stanford. In addition to an interactive desktop application for creating data visualizations, the Tableau ecosystem also includes servers for publishing and sharing interactive visualizations. These servers can be privately operated by individual customers or hosted in the cloud.

1.2.1 Tableau Public

Of particular relevance in the present work is a free version of the server environment called Tableau Public [12]. Tableau Public (or "Public") allows authors to publish data visualizations that can be shared with the general public. The published data sets are limited to 100K rows and must be stored using the Tableau Data Engine (or "TDE") [2, 3], but are otherwise free to use the full analytic capabilities of the Tableau system. In particular, the TDE provides native support for all analytic calculations defined by the authoring and publishing components. Moreover, visualizations published to Public are uploaded as complete Tableau workbooks, including the data model developed as part of the analysis. This makes it a rich source of data on the analytic habits and frustrations of Tableau users.

1.2.2 Functions

While the existing standards for query languages are helpful for defining a core set of row-level functions, there are many useful functions beyond the standards that are only explicitly supported by a subset of RDBMSes, often with different names. And even when a database does not provide an implementation of a particular function, it is usually possible to generate it as an inline combination of existing functionality. Tableau's calculation language is designed to be a lingua franca that tames this Babel of dialects by providing a common syntax for as many of these functions as possible.

1.3 DATEPARSE

Among the recent additions to the Tableau function library is a function called DATEPARSE, the usability of which is the focus of the work in this paper.

1.3.1 Scalar Dates

The SQL-99 standard defines three temporal scalar types: DATE, TIMESTAMP and TIME. They are typically implemented as fixed-point types containing an offset from some epoch (*e.g.* Julian Days.) This makes them compact to store and allows some temporal operations to be implemented very efficiently using arithmetic. Thus from the RDBMS perspective, representing dates in scalar form provides numerous benefits for users, both in terms of available operations and query performance.

Tableau models the first two of these types as "Date" and "Date & Time"; the third (pure time) is folded into Date & Time by appending it to a fixed date of 1899-12-30. From the analytic perspective, date types are dimensional (*i.e.* independent variables) and can be used as either categorical (simply ordered) or quantitative (ordered with a distance metric) fields. Categorical dates have a natural hierarchy associated with them generated by calendar binning. The visualization in Figure I1 shows an example of a bar chart employing binned categorical dates in a year/quarter hierarchy.

2. PARAMETERS

3. MINIMAL DESCRIPTIVE LENGTH

4. NATURAL LANGUAGE PROCESSING

4.1 Context-Free Grammar

ICU date-time formats are well defined both structurally and semantically, and can be defined by a context-free grammar (CFG). A CFG is commonly defined as a set of productions or rules of the form A ? ? where A is a variable, ? is a sequence of variables and terminal symbols (the tokens that make up the alphabet of the language) plus null (?), and the production symbol (?) indicates that the variable A can be expanded into ? . A CFG can be formally specified with four components: V, T, P, and S, where V is the set of variables, T the set of terminal symbols, P the set of productions, and S the set of available start symbols (a non-empty subset of V) [cite].

Pattern-recognition problems such as parsing date and time formats initiate from observations generated by some structured stochastic process. In other words, even if the initial higher-level production rule of the grammar is known (i.e. date, time or date-time), there could be several directions that the parser resolve to. For example, in a date string 5/6/2015, the pattern could either be M/d/yyyy or d/M/yyyy.

Probabilistic context-free grammars (PCFGs) have provided a useful method for modeling such uncertainty [cite]. Once we have created a PCFG model of a process, we can apply existing PCFG parsing algorithms to identify a variety of date-time formats. However, the parserÕs success is often limited in the types of the dominant patterns that it can identify. In addition, the standard parsing techniques generally require specification of a complete observation sequence. In many contexts, we may have only a partial sequence available (e.g. an incomplete entry). Finally, we may be interested in computing the probabilities of date-time patterns that the grammar may not explicitly define. To extend the forms of evidence, inferences, and pattern distributions supported, we need a flexible and expressive representation for the distribution of structures generated by the grammar. We adopt Bayesian networks for this purpose, and define an algorithm to generate a probabilistic distribution of possible parse trees corresponding to a set of date-time patterns as opposed to individual ones.

5. ACKNOWLEDGMENTS

This section is optional; it is a location for you to acknowledge grants, funding, editing assistance and what have you. In the present case, for example, the authors would like to thank Gerald Murray of ACM for his help in codifying this *Author's Guide* and the **.cls** and **.tex** files that it describes.

APPENDIX

A. HEADINGS IN APPENDICES

The rules about hierarchical headings discussed above for the body of the article are different in the appendices. In the **appendix** environment, the command **section** is used to indicate the start of each Appendix, with alphabetic order designation (i.e. the first is A, the second B, etc.) and a title (if you include one). So, if you need hierarchical structure *within* an Appendix, start with **subsection** as the highest level. Here is an outline of the body of this document in Appendix-appropriate form:

A.1 Introduction

A.2 The Body of the Paper

A.2.1 Type Changes and Special Characters

A.2.2 Math Equations

Inline (In-text) Equations

Display Equations

A.2.3 Citations

A.2.4 Tables

A.2.5 Figures

A.2.6 Theorem-like Constructs

A Caveat for the TEX Expert

A.3 Conclusions

A.4 Acknowledgments

A.5 Additional Authors

This section is inserted by LATEX; you do not insert it. You just add the names and information in the `\additionalauthors` command at the start of the document.

A.6 References

Generated by bibtex from your .bib file. Run latex, then bibtex, then latex twice (to resolve references) to create the .bbl file. Insert that .bbl file into the .tex source file and comment out the command `\thebibliography`.

B. MORE HELP FOR THE HARDY

The acm_proc_article-sp document class file itself is chock-full of succinct and helpful comments. If you consider yourself a moderately experienced to expert user of LATEX, you may find reading it useful but please remember not to change it.