LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

UCRL-CONF-209719

Automatic Generation of Data Types for Classification of DeepWeb Sources

A. H. H. Ngu, D. J. Buttler, T. J. Critchlow

February 15, 2005

2nd International Workshop on Data Integration in the Life Sciences
San Diego, CA, United States
July 20, 2005 through July 22, 2005

Automatic Generation of Data Types for Classification of Deep Web Sources *

Anne H.H. Ngu **[1], David Buttler[2], and Terence Critchlow[2]

[1] Department of Computer Science, Texas State University, San Marcos, TX 78666

[2] Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, CA 94551

Abstract. A Service Class Description (SCD) is an effective meta-data based approach for discovering Deep Web sources whose data exhibit some regular patterns. However, it is tedious and error prone to create an SCD description manually. Moreover, a manually created SCD is not adaptive to the frequent changes of Web sources. It requires its creator to identify all the possible input and output types of a service *a priori*. In many domains, it is impossible to exhaustively list all the possible input and output data types of a source in advance. In this paper, we describe machine learning approaches for automatic generation of the data types of an SCD. We propose two different approaches for learning data types of a class of Web sources. The Brute-Force Learner is able to generate data types that can achieve high recall, but with low precision. The Clustering-based Learner generates data types that have a high precision rate, but with a lower recall rate. We demonstrate the feasibility of these two learning-based solutions for automatic generation of data types for citation Web sources and presented a quantitative evaluation of these two solutions.

1 Introduction

One of the main impediments to large-scale integration of Deep Web sources is the inability to reconcile the semantic heterogeneity of the sources in an automatic and consistent manner. The problem can be decomposed into homogenization of the input, output, and interaction semantics of a Deep Web source. While there is a large body of research work [4, 9, 5] on homogenizing semantics of the input schema of Web sources, not much work is reported on homogenizing the output and the interaction patterns of Web sources. We proposed a practical, heuristic approach for reconciling the semantic of a class of life science Web sources using a Service Class Description (SCD) driven by users application needs. This SCD describes the generic functionalities of services in a particular domain, using example queries, the expected output, and a graph representation (workflow) of how service class members are expected to operate. We are able to classify two-thirds of BLAST sources with 100% accuracy using the manually created SCD in our initial experiments [7].

However, the manual creation of SCD is tedious and error-prone. It requires expert knowledge of both XML and regular expressions. Moreover, any change that affects the input, output, or navigation pattern

* This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-ENG-48. UCRL-JC.

** This work was performed while the author was a summer faculty scholar at LLNL

of the sites, which is not already embedded in the SCD, will affect the accuracy of the classification. It is impossible to be able to anticipate all the possible input and output data types in a class of dynamic web sources. Instead, an adaptive approach where an SCD can be incrementally created and updated is extremely valuable. The automatic creation of SCD enables a scalable and adaptive approach to semantic reconciliation of large numbers of sources in the presence of frequent re-organization, variation in navigation styles and input and output format of underlying sources. Based upon a few known Web sources of interest in a particular domain, a user can interact with those sites by going to a query page, posting a query and retrieving the results. The actions and responses from these chosen sites are used to construct a target SCD that can be fine-tuned with each successive example site.

We propose two different approaches for learning a set of rules that can be used to discover the data types of a class of Web sources. The Brute-Force approach is able to generate regular expressions ranging from the most specific to the most generic patterns for a given tagged example. The generated rules have a high recall rate but low precision. The Clustering-based approach aims to generate regular expressions that best fit the given set of training examples. The generated rules have a high precision rate, but with a lower recall rate than the Brute-Force approach. Both of these approaches allow new rules to be added or revised incrementally with new training examples. Eventually our learner will create a representative set of rules (data types) for a class of Web sources that is tailored to that domain's user information seeking behavior. The main contribution of this paper is the demonstration of the feasibility of these two learning-based solutions for automatic data types generation and a quantitative evaluation of these two solutions.

The remainder of the paper is organized as follows. We give an overview of Service Class Description in Section 2. We introduce our Data Type Learner in Section 3 and give an overview of learner system architecture as well as the annotated examples, the domain template type, and the training documents required by the learner. We show in Section 4 the details of our two different approaches toward learning the data type rules. We then evaluate the effectiveness of the two learning approaches in Section 5. We conclude with future research in Section 6.

2 Service Class Description

The service class description provides a mechanism for encapsulating the components that are common to all members of the class and is the means for hiding insignificant differences between individual sources in a particular domain. However, it must also provide enough information to differentiate members of the class from a set of arbitrary Web sources. Service classes are specified by a *service class description*, which uses an XML format and regular expression to define the relevant functionality of a category of Web sources, from an application's perspective. The service class description format supports three categories of information used to define a Deep Web source: *data types*, *example queries*, and *control flow*.

Data Types are used to describe the input and output of a service class and any data elements that may be required during the course of interacting with a service. The service class data type system is modeled after the XML Schema [3] type system and includes constructs for building atomic and complex types. The `DNASequence` type in Figure 1 is an example of an user-defined type in the nucleotide BLAST service class

```
<type name="DNASequence" type="string" pattern="[GCATgcat-]+" />
<type name="AlignmentSequence" >
    <element name="AlignmentName" type="string" pattern=".{1,100}:" />
    <element type="whitespace" />
    <element name="m" type="integer" />
    <element type="whitespace" />
    <element name="Sequence" type="DNASequence" />
    <element type="whitespace" />
    <element name="n" type="integer" />
</type>
```

Fig. 1. *Sample BLAST service class data type definitions.*

description. Figure 1 also shows examples of user defined types called `AlignmentSequence` which makes use of `DNASequence` type.

Control flow graphs are used for enumerating the expected navigational paths used by all members of the service class. A control flow graph consists of a set of states connected by edges. *Examples* contain queries that can be executed against an instance of the service class. Specifically, examples can be used to determine if a site accepts input (data) as required by the service class. In the context of this paper, we only examine the automatic generation of the data types, which are used by both the example queries and the control flows. The automatic generation of example queries and control flow for a specific web site required in a service class description is beyond the scope of this paper.

3 Data Type Learner

The automatic generation of data types for a Service Class Description (SCD) alleviates the tedious and error-prone approach of manual SCD creation. It involves 1) locating the regions in the document that the system is interested in generating the data types for, 2) partitioning the regions into tokens of suitable granularity for data types generation, 3)generating a regular expression (regex) for each data type that balances specificity and generality from the set of annotated examples. We assume that existing techniques such as PageDiff [7], QA-pagelets [2] and Omini region identification [1] can be used to locate regions of interest. We use a simple tokenization mechanism based on whitespace and punctuation marks in our current Data Type Learner for token generation. The core of our Data Type Learner is the automatic generation of regexes (the fundamental data types that makes up SCD definition) based on a set of annotated examples. Our approach to regex generation is similar to WHISK [8] which learns rules in the form of regular expressions from human annotated examples.

The input to the Data Type Learner is a training document, a set of user-annotated examples and a domain specific template type. The output is a set of rules (regex) that can be used to identify the data types specified in the template in an unseen document from the same class. There are two types of atomic rules that can be generated by the Data Type Learner: Matrix rules and Token Set rules. These two types

of atomic rules are at the opposite extremes in the spectrum of rule generation. Matrix rules match a fixed number of strictly defined tokens, while Token Set rules match a statistical group of tokens from a set. Tokens in a Matrix rule are strictly ordered, while tokens in a Token Set are unordered. We can create Composite rules from a collection of other rules (such as Matrix rules, Token set rules, and other Composite rules).

Users or domain experts must first describe the generic structure of data in a specific domain that they are interested in before they can use the learner to generate the data types for this domain. This high-level type information is described in a domain specific template type. Figure 2 shows a template for a type meant to describe a citation. The `citation` template states that a citation data type is a composition of `author`,

```
<type name="citation" type="CompositeRule" >
        <type name="author" type="MatrixRule"/>
        <type name="title" type="MatrixRule"/>
        <type name="venue" type="MatrixRule"/>
        <type name="date" type="MatrixRule"/>
</type>
```

Fig. 2. *Template Type for Citation.*

`title`, `venue` and `date`. For each data type, users can specify the type of rules that can be generated for it. For simplicity in presentation, the example in Figure 2 specifies that Matrix rules need to be generated for all the data types associated with a citation.

A Matrix rule is defined to be a list of rule tokens. Each rule token in a Matrix rule can be a literal string, a semantic class, an user-defined regex type or a regex pattern. The semantic class and user-defined regex type rule tokens are techniques used to inject specific domain knowledge into the Data Type Learner. This will improve the quality of the generated rules. For example, in generating the Matrix rules for a date, knowledge about the valid years, months and days can be incorporated via a semantic class or a regex type. A semantic class is effectively an enumeration of all instances of a type in a specific domain, while regex type is a predefined regular expression for a specific type common to a domain. An example of a predefined regex type for the BLAST domain is DNASequence type shown in Figure 1. The other inputs to a Data Type Learner are the training document and a set of annotated examples from that document. A training document in our case is an html page showing a list of citations. The annotated examples are data that users want to extract.

The Composite rule is defined as an ordered composition of atomic data types or another Composite type. There is no limitation on the number of times a particular atomic data type can appear in the citation. Thus, many matrix rules can be generated for the `author` atomic type. Each matrix rule represents a composition of regex patterns that can be used to identify one form of the atomic type. Figure 3 shows an example of a generated matrix rule for a date instance `2004 Jan`. A generated matrix rule for the date instance states that the given date can be identified by the regex pattern `\d{4}` followed by a blank space and then by a valid month in the semantic class "month.cls". Being a matrix rule, the ordering is important here.

```
<Rule>
  <matrixRule>
    <TokenList><Text>\d{4}</Text></Tokenlist>
    <TokenList><Text>\p{Blank}</Text></TokenList>
    <TokenList><Text>month.cls</Text></TokenList>
  </matrixRule>
</Rule>
```

Fig. 3. *Generated Date Matrix rule.*

4 Approaches for Learning Data Type Rules

The data type rules generation phase can be divided into three main steps. The first step is the tokenization, the second step is the rule generation, and the third step is the rule measurement. A specific data type example, such as a date, is read in to the rule generator. The example is then broken up into a list of text tokens. For example, the date `May 23,2004` becomes [“May”, “ ”, “23”, “,”, “2004”]. Then, for each token, a list of rule tokens in the form of regular expression are generated using either the Brute-Force approach or the Clustering-based approach. From the generated set of rule tokens, a filtering mechanism is employed to eliminate rules that resulted in very low precision and add rules with high precision to the final rule set.

4.1 Brute-Force Approach to Rules Generation

The Brute-Force rules generation generates all the candidate regular expressions that can match a given piece of text token. In this approach, regular expression rule tokens are generated from three sources. First, a simple regular expression (regex) that exactly matches the text token is generated. This is analogous to creating a literal string for every text token. Second, a list of user-supplied named regex type is searched for generating other regexs that can match the given text token. Finally, a hierarchy of domain independent generic regex types is searched for generating all other potential regexes for the text token. These generic regex types match words in simple ways, such as ”Capitalized”, ”punctuation”, or ”lower case”. Thus, in Brute-Force approach, our date example can be transformed into the ragged matrix as shown in Table 1:

May		**23**	**,**	**2004**
May	`\p{Blank}`	23	,	2004
First Name		number	comma	year
Month.cls				
Capitalized				

Table 1. Rule Tokens for ”May 23,2004” date instance

Given a ragged matrix, the Brute-Force algorithm creates a list of rules. Potentially, the list of rules equal to $\Pi_{i=1}^{n}|t_i|$, or the total number of combinations of each possible match for each text token in the example. The

list of generated rules is then passed to the rule measurement phase. The rule measurement phase removes equivalent rules, consolidates rules when a general rule can be generated to replace two or more specialized rules, and provides a ranking for the generated rules. From the ranked list of rules, obviously bad rules are discarded. Bad rules are those rules that match too many non-examples in the training document.

4.2 Clustering-based Approach to Rules Generation

The main problem with the Brute-Force approach to rule generation is that an exponential number of rules are being generated as the number of text tokens in an example instance increase. This means the search space is very large when using the generated rules for classification. The Clustering approach is based on the observation that when similar examples are clustered together, the regularity across all the examples can be captured by computing the intersection of the set of regexs that match all the instances in that class. We call the resulting list of regexs from the intersection computation the *maximal* regex for that specific class of examples.

The algorithm for Clustering-based approach to rule generation is shown in Figure 4. It consists of three main steps. The first step clusters the example instances of a specific type with the same number of tokens together. The second step generates the regex patterns for each text token of the example instance of a specific class based on observed characteristics of the example instance as contrasted to using a generic text hierarchy patterns, and the third step computes the *maximal* regex pattern for all instances of a specific class. For each tagged example of a specific type (such as date), we cluster it based on the number of text tokens in that instance. For example, the various instances of dates are clustered into Class3 (which has three text tokens), Class5 (which has five text tokens) and Class7 (which has seven text tokens) in Figure 4. For each cluster of example instances of a specific type, we generate regex patterns for each text token of each instance based on a set of rules as described in [6].

5 Experimental Evaluation

In this section, we evaluate the quality of rules generated and the computational trade-off between the two approaches. We use the same training document, annotated examples and template type for both data type learners. The data types that we are interested in generating and measuring are the `author` and `date` in the citation example.

The experiment is conducted in two phases for both types of learners. The first phase is the learning phase where rules for recognizing a citation are generated. The second phase is the classification phase where the generated rules are applied to an unseen citation document. The number of rules generated for each data type by each type of learners is recorded. The time taken to classify the unseen citation document using the generated rules is recorded for each data type learner. These details are shown in Table 2. The larger the number of rules being generated, the longer is the processing time. We use the standard precision and recall to evaluate the effectiveness or the quality of the generated rules. True positive in our results tables are instances that are correctly identified. False negative instances are those that are relevant, but which the generated rules failed to identify them (e.g. those author names which we missed). False positive are

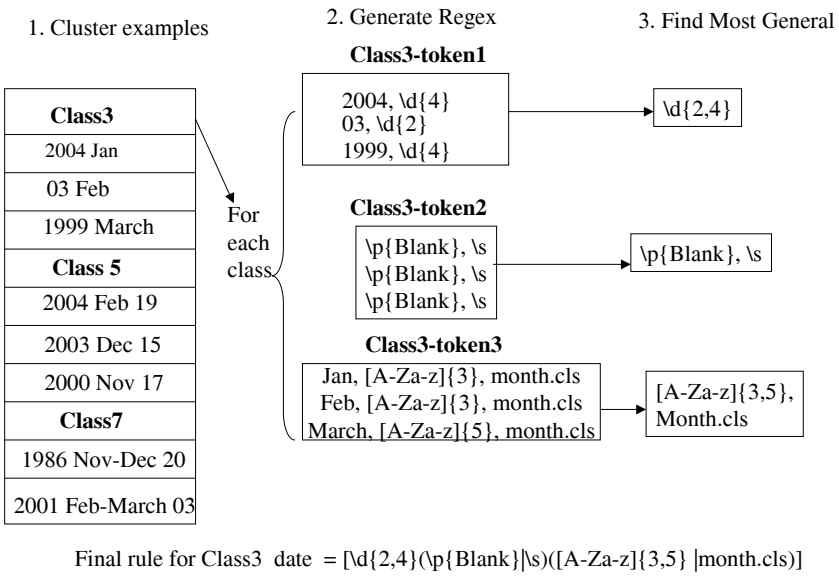

Fig. 4. A Clustering-based algorithm for rules generation.

instances that are wrongly identified as being relevant. Table 3 shows the result of Cluster Learner while Table 4 shows the result of Brute-Force Learner. A discussion on those results can be found in [6].

Data Type Learner	No of rules for Author	No of Rules for Date	Classification Time
Brute-Force Learner	928	126	586 secs
Cluster Learner	1	2	90 secs

Table 2. No of rules generated and processing time for each learner.

6 Conclusion

We demonstrated in this paper the feasibility of automatically generating data types for service class description. The minor variation in site specific data patterns coupled with common regularity exhibited by the input and output across a class of Web sources lend itself to supervised machine learning technique. We discussed two different approaches to learning the data type rules. The Clustering based approach has a definite advantage over the Brute-Force approach from our initial set of experiments conducted for recognizing citation data types.

The strategies that we employed for learning is fairly simple and preliminary at the moment. We have not yet exploited the ordering and the specific alphabetic letter or digit that can occur in the text token. We have also not explored the orthographic features such as capitalization and position of the text token within an example instance. Incorporating these heuristics will increase the quality of our generated rules. Our learner uses a very simple clustering mechanism based on the number of tokens in the given text instance. More effective clustering that exploits high-level token semantics needs to be investigated.

Data Set (Cluster-Learner)	True Positive	False Positive	False Negative	Recall	Precision
Training document (author)	72	3	5	93%	96%
Unseen document(author)	192	7	26	88%	97%
Training document (date)	4	0	0	100%	100%
Unseen document(date)	40	12	1	88%	97%

Table 3. Results for Cluster Learner.

Data Set (Brute-Force Learner)	True Positive	False Positive	False Negative	Recall	Precision
Training document(author)	75	19	2	97%	79%
Unseen document (author)	208	101	10	95%	67%
Training document(date)	4	15	0	100%	21%
Unseen document (date)	40	70	1	97%	36%

Table 4. Results for Brute-Force Learner.

References

1. David Buttler, Ling Liu, and Calton Pu. A fully automated object extraction system for the world wide web. *Proceedings of IEEE International Conference on Distributed Computing Systems*, April 2001.
2. J. Caverlee, L. Liu, and D. Buttler. Probe, Cluster, and Discover: Focused Extraction of QA-Pagelets from the Deep Web. In *Proceedings of the 20th IEEE International Conference on Data Engineering (ICDE04)*, Boston, USA, 2004.
3. David C. Fallside. XML Schema Part 0: Primer. Technical report, World Wide Web Consortium, http://www.w3.org/TR/xmlschema-0/, 2001.
4. Bin He and Kevin Chen-Chuan Chang. Statistical schema matching across web query interfaces. In *Proceedings of ACM/SIGMOD Conference on Management of Data*, San Diego, CA, 2003. ACM Press.
5. Jayant Madhavan, Philip A. Berstein, and Erhard Rahm. Generic schema matching with cupid. In *Proceedings of the Twenty-seven International Conference on Very Large Databases*, Roma, Italy, 2001. VLDB Endowment.
6. AHH. Ngu, D. Buttler, and T. Critchlow. Automatic Generation of data Types for Classification of Deep Web Sources . Technical Report UCRL-CONF-209719, Lawrence Livermore National Laboratory, to appear in Data Integration in the Life Sciences Conference-DILS2005, 2005.
7. Anne H. Ngu, Daniel Rocco, Terence Critchlow, and David Buttler. Automatic discovery and inferencing of complex Bioinformatics Web Interfaces. Technical Report UCRL-JRNL-201611 (to appear in WWW journal, Springer), Lawrence Livermore National Laboratory, Livermore, CA, 2004.
8. Stephen Soderland. Learning Information Extraction Rules for Semi-structured and Free Text. *Machine Learning*, 1(44):1–44, 1999.
9. Jiying Wang, Ji-Rong Wen, Fred Lochovsky, and Wei-Ying Ma. Instance-based schema matching for web databases by domain-specific query probing. In *Proceedings of the Thirty International Conference on Very Large Databases*, Toronta, Canada, 2004. VLDB Endowment, Toronta, Canada.