Active Learning of Predefined Models for Information Extraction: Selecting Regular Expressions from Examples

Alberto BARTOLI, Andrea DE LORENZO, Eric MEDVET and Fabiano TARLAO,

a Department of Engineering and Architecture, University of Trieste, Italy

Abstract. We consider the problem of constructing a regular expression for information extraction automatically, based only on examples of the desired extraction behavior. We describe an active learning framework that is not aimed at synthesizing a solution from scratch, but rather is aimed at selecting a solution from a set of more than 3000 solutions that have already proven useful in a broad range of practical applications. The user provides only one example of desired extraction and then interactively annotates text snippets selected by the system. The system constructs such queries based on uncertainty sampling, i.e., by selecting the snippet on which it is most uncertain at each learning step. The resulting framework allows solving many practical extraction problems quickly and simply.

Keywords. Active learning, information extraction, regex, regular expressions

Introduction

A common practical problem in many data mining applications consists in the identification of data items which follow a certain syntactic pattern to be specified by a user. In many application domains such patterns take the form of regular expressions, due to their expressive power and the wide availability of software and tools that implement regular expressions efficiently. Unfortunately, writing a regular expression for solving a given information extraction task is difficult and requires specialized technical skills that are typically orthogonal to those of the specific data mining application. For this reason, a long-standing problem in the scientific community is the automatic construction of a regular expression based on examples of the desired extraction behavior (please refer to [1] for a detailed survey). Recent proposals based on genetic programming have proven to be practically feasible and competitive with human experts, even on extraction tasks of realistic complexity [1,2,3].

The problem of constructing a regular expression automatically is usually cast by assuming that each instance of the problem is unique, that is, by constructing a regular expression from scratch as if the extraction task encountered by the user had never
be seen before. Actually, it is fair to claim that many extraction tasks are relatively frequent, such as extracting email addresses with or without the mailbox name preceding the @ character, extracting only certain HTML tags, extracting dates in different formats, and so on. Based on this observation, that we corroborated by analyzing the typical usage of our webapp for automatic construction of regular expressions (http://regex.inginf.units.it/), we consider the problem of selecting a regular expression from a large precompiled set rather than constructing such an expression from scratch. Specifically, we collected a set of more than 3000 regular expressions that solve common extraction tasks and developed an interactive tool that selects the regular expression most suitable to the examples provided by the user.

Our tool is based on active learning, a principled framework for automatically choosing, from a collection of unlabelled data, data to be labelled by a human operator [4,5]. Active learning is aimed at minimizing the annotation effort spent by the user and has proven useful in a broad range of different application domains, e.g., [6,7,8,9,3]. In our tool the user initially annotates only one desired extraction in the input text and then interactively answers extraction queries constructed by the system, by merely annotating on a graphical interface which portions of a query are, or are not, to be extracted. The system selects candidate regular expressions from the precompiled set, by exploiting the annotation information progressively provided by the user, and selects queries based on uncertainty sampling, an active learning criterion based on minimizing the uncertainty between multiple candidate solutions.

Our contribution is twofold. First, we describe an active learning framework that is not aimed at learning a model of the training data from scratch, but rather is aimed at selecting an element from an existing set of models. The set contains models that have already proven useful in a broad range of practical applications. We believe that frameworks of this kind could become more and more useful as applications of machine learning become more and more commonplace. Second, we describe a tool able to solve a practical and important problem: constructing a regular expression for information extraction from an unstructured text stream. While the tool is not able to solve all possible instances of the problem, it is able to solve many such instances very quickly and simply.

1. Active Learning Framework

The problem consists in automatically generating a regular expression based on examples of the desired behavior. These examples take the form of annotations on a text stream \( t \), i.e., substrings of \( t \) that are to be extracted (desired extractions) or substrings of \( t \) that are not to be extracted (undesired extractions). Initially the user provides only one desired extraction in \( t \). Then, the following steps are executed iteratively:

1. The system exploits the annotation information provided by the user so far for:
   (i) constructing a list of candidate solutions; and, (ii) determining an annotation query (briefly, a query) aimed at improving the quality of that list. A query is a carefully chosen substring of the text stream \( t \) and shown to the user.
2. The user responds to the query by specifying whether the substring is to be extracted or not (more details on answers to queries are given below).
As in any active learning framework, the iteration may proceed indefinitely. In practice, execution terminates when the user is satisfied by one of the candidate solutions, or when the annotation budget has exhausted, or when the time spent in interacting with the system has exhausted.

We emphasize that step 1 is almost instantaneous (as perceived by the user). This feature is in sharp contrast with our previous work based on a similar active learning framework [3], in which step 1 took tens of seconds. The reason for such a difference is because in the cited work we constructed candidate solutions from scratch, by executing at each step a carefully optimized evolutionary search based on genetic programming, while in this work we select candidate solutions from a precompiled set.

The answer to a query $q$ consists of a pair $(Q_D, Q_U)$, where: $Q_D$ is the set of all desired extractions which overlap $q$; $Q_U$ is the set of all maximal substrings of $q$ which are not desired extractions (either of two sets may be empty, but not both). The reason for this interaction model is as follows. Intuitively, the answer should be either “yes”, meaning that $q$ is to be extracted, or “no” meaning that $q$ is not to be extracted. We designed a more general interaction between system and user because: a query could overlap with a desired extraction only in part; and, a query could span across several desired extractions that are adjacent yet should be kept separate. In the first case the answer to the query should be a string obtained by extending the boundaries of $q$ on one or both sides. In the second case the answer should be an indication of the separations between the desired extractions. The answer form described above accommodates all these cases. We emphasize that the proposed interaction model is different from most active learning approaches, because in those approaches the user is required to provide only the class of queried data and is not allowed to modify those data. In our model, in contrast, the user is allowed to (slightly) modify the data queried by the system.

2. Graphical User Interface

We developed a prototype based on web technologies (see Figure 1) that implements the interaction model described in the previous section. The user loads a text stream (e.g., a file), selects a snippet as a desired extraction and starts the system. The system then executes the iteration described in the previous section.

Each query is highlighted in the text stream (with yellow lines in our prototype, see Figure 1). By clicking on the query, the user is presented with 3 options in a drop-down menu (see the inner box in Figure 1). One single click suffices to answer a query that coincides with a desired extraction (“Yes, I want this”) or that does not overlap with a desired extraction (“No, I don’t want this”). When the answer has to be more complex, the “Edit this query” option enters an annotation mode (not shown for brevity) in which the user may extend the snippet on one or both sides, as well as indicate separations between adjacent desired extractions. In case of Yes/No answers, the system execution resumes immediately, without any further user action. Thus, in these cases, one single click suffices to answer the query, start the refining of the candidate solution list, and obtain the next query.

When the system is waiting for an answer to the current query (step 2 of the above iteration), the user may take as much time as necessary for assessing the current list of proposed solutions, that is always shown to the user (see left panel in Figure 1). The
user may click on one of these solutions and the system highlights on the text stream the substrings extracted by that regular expression. The system also highlights all the annotation information provided by the user so far, that is, all the desired extractions and undesired extractions. Figure 1 shows a scenario in which the first candidate solution is selected; the selected regular expression extracts two substrings in the portion of the text stream that appears in the window (14–09–2011 and 23/03/2009, highlighted in green); the second of these substrings is also underlined, meaning that it was a query submitted in a previous iteration that the user answered as a desired extraction. The substring A BILL is also highlighted and underlined but in red color, meaning that this substring was a query that the user answered as an undesired extraction.

The visible portion of the text stream is usually not enough for assessing the behavior of a candidate solution. In practice, after answering a few queries, the user will start analyzing the behavior of a selected solution on larger portions of the text stream by using the scrolling bar. After a few experiments we quickly realized the need for a “Find query” button (left panel), which scrolls the text stream so that the query becomes visible and located at the top of the window automatically.

3. Implementation

We collected a set of 3721 of regular expressions from web forums devoted to programming, most of them from the public web service of http://regexlib.com. These regular expressions should solve such common tasks as extracting email addresses, dates, HTML tags, and so on, with varying degrees of accuracy and different choices for managing corner cases or specific requirements. After removing duplicate expressions, we checked their syntactical validity with a regular expression engine and discarded those that caused a syntax error. We then applied each of the remaining expressions on a long text stream (in the order of several MBs) for measuring the time required for processing the full stream. We chose to include in our prototype only the regular expressions that required less than 500 ms for such processing on a modern laptop. The resulting set is composed of 3058 regular expressions. Candidate solutions are selected from this set.
We need to describe how the system selects candidate solutions from the set of available regular expressions and how it selects queries from the text stream.

Let $X^D$ and $X^U$ denote, respectively, the set of desired extractions and undesired extractions (i.e., substrings that should and should not be extracted). These sets contain substrings provided by the user when answering queries and constitute the examples of the desired behavior. Initially, $X^D$ contains only one element and $X^U$ is empty. We assume the absence of conflicting annotations, i.e., there is never any overlap between an element of $X^D$ and an element of $X^U$.

Let $X_s$ denote the set of extractions of candidate solution $s$ and let $x_s$ denote an element of such set. We say that $x_s$ is a true positive if $x_s$ coincides with an element of $X^D$; a false positive if either $x_s$ overlaps in part with an element of $X^D$, or it coincides with an element of $X^U$, or it overlaps with an element of $X^U$. We say that all elements in $X^D \setminus X_s$ are false negatives, i.e., substrings that should be extracted but that $s$ fails to extract. Let $TP_s$, $FP_s$, $FN_s$ be the number of true positives, false positives, false negatives resulting from all $x_s \in X_s$. We define precision and recall of $s$ as, respectively, $\text{Prec}_s = TP_s/(TP_s + FP_s)$ and $\text{Rec}_s = TP_s/(TP_s + FN_s)$. Finally, we define the F-measure of $s$ as $\text{Fm}_s = 2\text{Prec}_s\text{Rec}_s/(\text{Prec}_s + \text{Rec}_s)$.

At each iteration the system computes the F-measure of all regular expressions available, ranks these expressions in decreasing order of F-measure and presents to the user the top $N$ regular expressions. We chose to assign $N = 10$ in our prototype.

The query is selected in two phases. In a first tokenization phase the text stream is subdivided in substrings, i.e., tokens. Each token is a candidate query. In the second phase the specific query to be submitted to the user is selected based on uncertainty sampling [10,11,12,13]. This criterion, described in more detail below, essentially consists in selecting the token for which there is most uncertainty between the candidate solutions that are currently top ranked.

The tokenization procedure constructs a set of characters $S_0$ with each character that immediately precedes or follows each element in the current set of desired extractions $X^D$. Using characters in $S_0$ as token separators might lead to an excessive number of queries that cannot be answered with a quick Yes/No answer, because in several extraction tasks there are characters that act as separator for certain desired extractions but are also part of other desired extractions (e.g., spaces). For this reason, we construct a list with all characters in $S_0$, ordered by decreasing number of occurrences. Next, we iterate on the first $k$ elements of the list, with $k$ ranging from 1 to the length of the list; we consider as token separators the first $k$ characters in the list and count the resulting number of tokens. Finally, we select as token separators the set of characters that result in the largest number of tokens (in case of ties we select the smallest set of token separators).

Once we have the token separators, we tokenize the text stream and discard all the tokens which overlap to any element in $X^D$ or $X^U$. We then assign an uncertainty score to each remaining token as follows. Initially, we assign score 0 to each token. We apply each solution $s$ of the $N$ top ranked candidate solutions to the text and modify the score of each token with these rules: if the token is extracted by $s$, then we increment the score of the token; otherwise, we decrement the score of the token. Having processed all the top ranked solutions, we divide all scores by $N$. The score of each token is thus in the range $[-1, +1]$, where $-1$ corresponds to maximal agreement that the token is not to be extracted, $+1$ maximal agreement that the token is to be extracted and 0 corresponds to maximal uncertainty between the candidate solutions regarding whether the token should
be extracted. We select as query the token whose score is closest to 0; in case of tie, we select the topmost (with respect to the text stream) token.

<table>
<thead>
<tr>
<th>Task name</th>
<th>Regular expression with F-measure=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>BibTex-Author</td>
<td>(?&lt;=\author {0,3}=(0,3){[^\n]}0,300)?(A-Z, (d0)){c} {A-Z, (a-z)}{n})</td>
</tr>
</tbody>
</table>
| Bills-Date         | \((?:[12]\{d(1,3)s*\}012\{d(1,3)s*\}012\{d(1,3)s*\})\{?:d(1,2),\}
| Email-Phone        | (?<=W)((?:[0-9](?: |-|\.))?(?:\{0-9\{3(?: |-|\.)?\{0-9\{4}))                           |
| Log-IP             | \b(?:d(1,3))\{3\}\{d(1,3)\}\b                                                  |
| Twitter-Hashtag + Mention | (?<=\W|^)((?:@\{A-Za-z0-9_+\})|(?:#\{[^\d]+\}))                                    |
| Web-URL            | \(?<=^|\b|\W)((?:https?|s?ftps?|smb|mailto):\{a-zA-Z\{[^\s:]+@)?                       |

Table 1. Regular expressions that solve the considered extraction tasks.

4. Discussion and concluding remarks

We performed a preliminary assessment of the proposed tool and the results are highly encouraging. We considered 6 of the extraction tasks analyzed in [1,3,2]. The first part of a task name (Table 1, first column) indicates the nature of the dataset while the second part indicates the nature of the entity to be extracted. For example, BibTex-Author is the task of extracting author names from a corpus of BibTex entries (full details about the corpora and the tasks can be found in the cited works). The second column of Table 1 contains a regular expression that solves the corresponding task with F-measure 1 and is available to our tool: it can be seen that the extraction tasks considered are very complex. Most importantly, these expressions were not scraped from the web: they were written by an expert after many iterations and were carefully tailored to each dataset.

We submitted these tasks to 12 users with widely differing familiarities with regular expressions. We explained to each user the usage of our prototype GUI along with a textual description of each task. We also specified that each user had complete freedom in deciding when a task is to be considered as completed. The average results are summarized in Table 2. It can be seen that, for three tasks, all the users selected a regular expression that solve the task without any mistake (i.e., those in Table 1 or one with the same behavior). An expression that does not exhibit a perfect F-measure was selected by some users in two tasks (Email-Phone and Twitter-Hashtag+mention) and by all users in the Web-URL tasks. Even in these cases, though, the average F-measure is very high.

We plan to perform a detailed study for assessing user behavior in more depth and for determining the robustness of the approach in the presence of noisy answers, as we observed that users tend to make occasional mistakes when answering queries, in particular, at the boundaries of answers constructed with the “Edit” answers. Overall, however, we believe that the approach is very promising as it indeed enables users to select regular expressions for common information extraction tasks quickly and simply.

We also plan to integrate the proposed tool with our other tools for automatic generation of regular expressions, so that “common tasks” can be solved quickly based on
Table 2. Average results for 12 users on 6 extraction tasks (Elapsed time is expressed in minutes:seconds; F-measure is the average among those selected by the users). The last column is the number of selected regular expressions that solve the extraction task without any mistake.

<table>
<thead>
<tr>
<th>Task name</th>
<th>Elapsed time</th>
<th>#Answers</th>
<th>F-measure</th>
<th>#F-measure=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>BibTex-Author</td>
<td>1:23</td>
<td>7.5</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>Bills-Date</td>
<td>2:31</td>
<td>13.5</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>Email-Phone</td>
<td>4:24</td>
<td>17</td>
<td>0.84</td>
<td>9</td>
</tr>
<tr>
<td>Log-IP</td>
<td>0:47</td>
<td>6.5</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>Twitter-Hashtag + Mention</td>
<td>1:19</td>
<td>8</td>
<td>0.98</td>
<td>7</td>
</tr>
<tr>
<td>Web-URL</td>
<td>2:43</td>
<td>7</td>
<td>0.85</td>
<td>0</td>
</tr>
</tbody>
</table>

regular expressions that are already available, while “uncommon tasks” can be solved with a more time-consuming evolutionary search based on the annotation information provided by the user during the early interaction steps with the system.

Acknowledgements

This work has been partially supported by the University of Trieste - Finanziamento di Ateneo per progetti di ricerca scientifica - FRA 2016. The authors are grateful to Pieredoardo Gabutti for his help in implementing the tool.

References