Synthesizing Extraction Rules from User Examples with SEER

Maeda F. Hanafi, Azza Abouzied
New York University - Abu Dhabi
{maeda.hanafi, azza}@nyu.edu

Laura Chiticariu, Yunyao Li
IBM Research - Almaden
{chiti, yunyaoli}@us.ibm.com

ABSTRACT

Our demonstration showcases SEER's end-to-end Information Extraction (IE) workflow where users highlight texts they wish to extract. Given a small set of user-specified example extractions, SEER synthesizes easy-to-understand IE rules and suggests them to the user. In addition to rule suggestions, users can quickly pick the desired rule by filtering the rule suggestion by accepting or rejecting proposed extractions. SEER's workflow allows users to jump start the IE rule development cycle; it is a less time-consuming alternative to machine learning methods that require large labeled datasets or rule-based approaches that are labor-intensive. SEER's design principles and learning algorithm are motivated by how rule developers naturally construct data extraction rules.

Keywords

Data Extraction; Example-driven learning; Information Extraction

1. INTRODUCTION

The need to quickly and automatically retrieve important information from semi-structured and unstructured documents, such as financial reports or news articles, has propelled interest in information extraction (IE) [2]. Many IE tools used extensively in many applications can be time-consuming, labor-intensive, or expensive. Most techniques fall under two major approaches: machine learning (ML) methods or programming language (PL) methods [2]. ML methods (surveyed in [3]) often require large training datasets and use complex, opaque models. PL methods [12, 6] often provide clear, easy-to-understand domain-specific languages. Yet, scripting even in a relatively easy language is labor-intensive and time-consuming. Similar to learning-by-example data extraction tools such as Wrangler [5], FlashExtract [8] and Wrapper Inductors [11, 7], SEER [4] combines the best of both worlds and learns easy-to-understand data extraction rules *uniquely suitable for free-form text* from a few user-provided examples of data extractions.

Consider a value-investor, Wendy Buffet, trying to assess the financial stability of several companies. She has access to five years of financial reports and statements produced by each company. Daunted by the manual task of extracting quarterly revenues for each company, Wendy decides to learn how to code in Python to automate the task. As a novice developer, she first spends hours writing and debugging a simple script to extract numbers following a dollar sign. Through several time-consuming coding attempts, she finally writes a script that can extract several variations of monetary values (e.g. '$50,000', '1.5 million USD', '600,000 dollars') only to realize that she still needs to filter her results to only monetary values that refer to quarterly revenues. She continues this long and laborious process until she achieves the desired quality of results. Wendy, then, realizes that she also wishes to examine losses. Intimidated by her initial experience, she stumbles upon machine learning. Confused by the terminology, the plethora of learning models, and the need for a labeled training dataset, she quickly gives up on this approach.

We designed SEER to address the challenges faced by users like Wendy with data extraction. SEER (1) reduces the barrier of learning a new programming language for novice developers, (2) minimizes the time-consuming, manual effort required in constructing rules, and (3) does not require large labeled datasets, which are hard to obtain. SEER suggests easy-to-understand extraction rules in Visual Annotation Query Language (VAQL) [9], a commercially available declarative graphical language for IE rule development over IBM's SystemT extraction engine [6]. With SEER, Wendy simply highlights examples of revenue or loss statements such as 'Fourth-quarter revenue was 6 million dollars' and SEER automatically suggests extraction rules, which she can easily modify to better suit her needs. Alongside rule suggestions, SEER also displays a collection of extractions, known as *refinements*, that help differentiate the suggested rules. Based on user feedback (accept or reject) on these refinements, SEER further refines the set of suggested rules. Filtering the suggestions helps the user quickly select the desired rule without having to analyze each suggested rule.

SEER is unique as its design principles are motivated by how actual rule developers construct rules [4]. First, rule developers often prefer to use *pre-built* semantic extractors that extract concepts such as phone numbers or dates over more syntactic extractors such as regular expressions that perform an identical extraction. SEER augments its learning algorithm with heuristics that encode such preferences. Thus, with the help of heuristics, SEER leverages domain knowledge that is unavailable to ML methods without the presence of large labeled datasets. Second, rule developers often create *dictionaries* that capture a set of interchangeable string literals such as 'increased', 'rose'. SEER's learning algorithm automatically merges literals to create dictionaries on-the-fly. Third, rule developers often construct multiple rules to extract text variations. SEER automatically generates natural disjunctions of rules when extractions cannot be captured by a single rule. Finally, SEER suggests structurally *diverse* rules as rule developers often prefer to examine rule variations that can lead to different extraction outcomes rather than slight variations of one rule.

SIGMOD'17, May 14-19, 2017, Chicago, IL, USA
© 2017 ACM. ISBN 978-1-4503-4197-4/17/05. . . $15.00
DOI: http://dx.doi.org/10.1145/3035918.3056443

Our evaluations for SEER [4] reveal that users with scripting experience were able to finish IE tasks in less time and with more accuracy than when constructing rules themselves in VINERy (Visual Integrated Development Environment for Information Extraction) [9], a commercially available drag-and-drop tool for manually building VAQL rules. Moreover, SEER also encouraged users to be more thorough: they proactively searched instances of target extractions in the documents and verified the extractions more often than with VINERy. Both activities are essential for the construction of precise and accurate rules. Without SEER's support, users often overlooked these activities and were confident of the accuracy of their self-constructed rules after only skimming their first few extractions [4].

SEER is different from existing synthesis works like FlashExtract [8] and Wrapper Induction methods [11, 7], because such works depend on features and consistent patterns of the surrounding target extraction text, or *context*, e.g. surrounding punctuation such as commas, semicolon, or HTML or XML tags, etc. SEER does not learn from the context and works just as well for both structured documents, e.g. CSVs, logs, etc., and unstructured documents, e.g. reviews, press statements, etc. Moreover, while ML techniques often require large labeled datasets and use complex and opaque models that are hard to interpret [2], SEER can learn rules in VAQL [9] from a handful of examples thus providing transparency and ease of debugging. SEER jump starts the process for developing IE rules from a blank canvas. Works related to SEER address different scenarios, such as refining existing rules [10] or extracting relations from the web [1].

2. DEMONSTRATION OVERVIEW

At the demonstration, conference attendees will use SEER to construct extraction rules and extract data from multiple preloaded data sets or any easily downloadable text dataset of their choosing. For example, users can extract financial information such as quarterly revenues from a dataset of press releases from different companies; or faculty contact information from online university directories; or crime data from a dataset of FBI press releases. The demonstration will walk each attendee not only through the user interface of SEER (Figure 1) but also through SEER's language, its rule learning and refinement algorithms, its system architecture and a summary of its user evaluation. To handle overcrowding, a short video demo of SEER will also be available for independent viewing on tablets.

In what follows, we will describe how SEER works, in a fashion similar to how we would describe it to attendees with the help of the following scenario.

2.1 Scenario

Jane is writing an op-ed on changes in crime patterns across the US. At the demo booth, she uses SEER to extract relevant information from a dataset of FBI press releases. In particular, she wishes to extract percentage changes in offenses. Jane will load the dataset into SEER (documents panel in Figure 1). She then searches for the word 'offenses' and highlights an example of a percentage change, e.g. 'offenses dropped 9.8 percent'. SEER colors the text with a yellow highlight. She then clicks on the button labeled "Positive Example?" to indicate that she wishes to extract similar data; the text is now highlighted in green. Alternatively, if she clicks the "Negative Example?" button, she indicates that she does not wish to extract the highlighted text and SEER colors it red.

Jane highlights more positive examples: 'offenses dropped 9.8 percent', 'offenses decreased 3.2 percent', 'offenses declined 8.0 percent', etc. She then clicks "Suggest me some rules!". As SEER learns and suggest rules in response to Jane's example highlights, we will describe SEER's rules (section 2.2), rule generation from positive (section 2.3) and negative examples (section 2.4) and how rule refinement works (section 2.5).

2.2 SEER rules

Given Jane's three positive example highlights, SEER suggests the rules in Figure 2a. SEER's rules are specified using a subset of Visual Annotation Query Language (VAQL) [9], and are executed by the SystemT engine [6].

Rules extract text from documents pre-tokenized by VAQL. VAQL delimits text on white space characters, such as spaces, newlines, tabs, etc. VAQL considers symbols like dashes, commas, etc., as tokens. Hence, '1998-Jan14' would have the following tokens: '1998', '-', and 'Jan14'.

SEER learns a disjunctive union of VAQL sequence rules, where each rule consists of a sequence of one or more extraction *primitives*, of one of the types listed below. The results of executing a rule or primitive over the input text are called *extractions*. We say that a rule *captures* a sequence of tokens if the sequence is among the extractions of the rule on the input text.

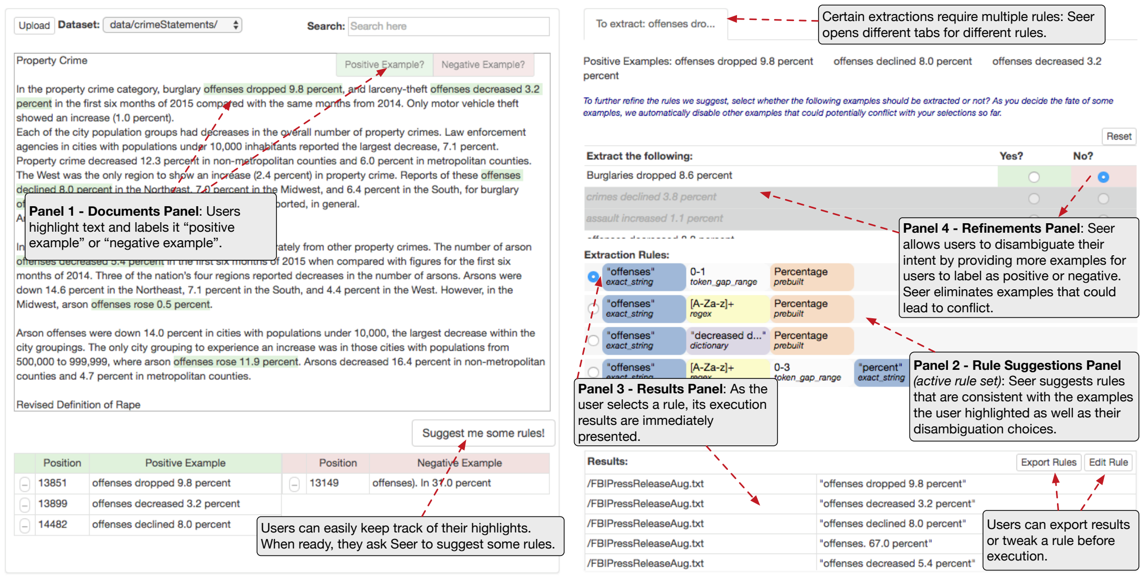
- **Pre-built:** Pre-builts capture one or more tokens of a particular concept such as organization, person, phone number, etc. For example, the pre-built *P*: Percentage captures percentages like '11.9 percent' and '9.8%'.
- **Literal:** Literals capture one or more tokens matching an exact string, e.g. *L*: 'percent' captures all tokens 'percent' that appear in the text.
- **Dictionary** Dictionaries contain a set of literals and capture tokens that match one of those literals, e.g. *D*: {up, down} captures any of the two words that appear in the text.
- **Token Gap**: Token gaps skip over a number of tokens. Token gaps cannot be placed in the beginning or the end of a rule. Within the rule, *L*: '11' *T*: 0-3 *L*: 'percent', the token gap *T*: 0-3 skips over zero to three tokens to capture the middle tokens of the following texts: '11.9 percent' or '11 percent'.
- **Regular Expression**: Regular expressions describe search patterns, e.g. *R*: [A-Za-z]+ captures a token consisting of letters. Other expressions are listed in [4].

SEER comes pre-loaded with a catalog of pre-builts. Users may also supply their own pre-builts in addition to the default pre-builts in VAQL. Such pre-builts are user-defined rules. For instance, a user can provide a pre-built for ordinal numbers, *P*: Ordinal Numbers, which is a rule consisting of a single dictionary primitive: *D*: {first, second, third, ...}.

2.3 Rule Generation from Positive Examples

The final set of rules in Figure 2a are *diverse* and *consistent*. Rule diversity ensures that the suggested rules capture different sets of extractions. Specifically, a diverse set of rules should not contain rules composed of mainly one type of primitive; for instance, a non-diverse set of rules would only contain pre-built primitives. The final set of rules suggested are also *consistent* with the given examples, i.e. they capture all the positive examples and capture none of the negative examples. SEER also creates dictionary primitives from unequal literals (see the fifth rule in Figure 2a). The dictionary contains the middle tokens of the examples, e.g. 'declined', 'decreased'.

SEER tests whether different primitives can capture tokens in a brute-force fashion and then stores sequences of these primitives in a rule *tree* for each example. Details of tree generation can found in the full paper [4] and we will explain these details to the demo


Upload **Dataset:** data/crimeStatements/ **Search:** Search here

Property Crime — Positive Example? | Negative Example?

In the property crime category, burglary offenses dropped 9.8 percent, and larceny-theft offenses decreased 3.2 percent in the first six months of 2015 compared with the same months from 2014. Only motor vehicle theft showed an increase (1.0 percent).
Each of the city population groups had decreases in the overall number of property crimes. Law enforcement agencies in cities with populations under 10,000 inhabitants reported the largest decrease, 7.1 percent.
Property crime decreased 12.3 percent in non-metropolitan counties and 6.0 percent in metropolitan counties.
The West was the only region to show an increase (2.4 percent) in property crime. Reports of these offenses declined 8.0 percent in the Northeast, 7.0 percent in the Midwest, and 6.4 percent in the South, for burglary of … ported, in general.

In … ately from other property crimes. The number of arson offenses decreased 5.4 percent in the first six months of 2015 when compared with figures for the first six months of 2014. Three of the nation's four regions reported decreases in the number of arsons. Arsons were down 14.6 percent in the Northeast, 7.1 percent in the South, and 4.4 percent in the West. However, in the Midwest, arson offenses rose 0.5 percent.

Arson offenses were down 14.0 percent in cities with populations under 10,000, the largest decrease within the city groupings. The only city grouping to experience an increase was in those cities with populations from 500,000 to 999,999, where arson offenses rose 11.9 percent. Arsons decreased 16.4 percent in non-metropolitan counties and 4.7 percent in metropolitan counties.

Revised Definition of Rape

Suggest me some rules!

	Position	Positive Example		Position	Negative Example
–	13851	offenses dropped 9.8 percent	–	13149	offenses). In 31.0 percent
–	13899	offenses decreased 3.2 percent			
–	14482	offenses declined 8.0 percent			

Panel 1 - Documents Panel: Users highlight text and labels it "positive example" or "negative example".

Users can easily keep track of their highlights. When ready, they ask Seer to suggest some rules.

To extract: offenses dro...

Certain extractions require multiple rules: Seer opens different tabs for different rules.

Positive Examples: offenses dropped 9.8 percent offenses declined 8.0 percent offenses decreased 3.2 percent

To further refine the rules we suggest, select whether the following examples should be extracted or not? As you decide the fate of some examples, we automatically disable other examples that could potentially conflict with your selections so far.

Reset

Extract the following:	Yes?	No?
Burglaries dropped 8.6 percent	○	●
crimes declined 3.8 percent	○	○
assault increased 1.1 percent		

Panel 4 - Refinements Panel: Seer allows users to disambiguate their intent by providing more examples for users to label as positive or negative. Seer eliminates examples that could lead to conflict.

Extraction Rules:

●	"offenses" *exact_string*	0-1 *token_gap_range*	Percentage *prebuilt*	
○	"offenses" *exact_string*	[A-Za-z]+ *regex*	Percentage *prebuilt*	
○	"offenses" *exact_string*	"decreased d..." *dictionary*	Percentage *prebuilt*	
○	"offenses" *exact_string*	[A-Za-z]+ *regex*	0-3 *token_gap_range*	"percent" *exact_string*

Panel 2 - Rule Suggestions Panel *(active rule set)*: Seer suggests rules that are consistent with the examples the user highlighted as well as their disambiguation choices.

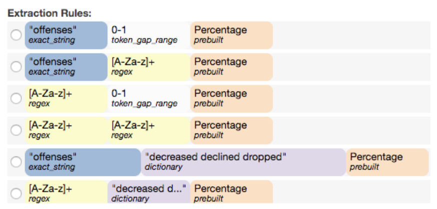
Panel 3 - Results Panel: As the user selects a rule, its execution results are immediately presented.

Results: Export Rules | Edit Rule

/FBIPressReleaseAug.txt	"offenses dropped 9.8 percent"
/FBIPressReleaseAug.txt	"offenses decreased 3.2 percent"
/FBIPressReleaseAug.txt	"offenses declined 8.0 percent"
/FBIPressReleaseAug.txt	"offenses. 67.0 percent"
/FBIPressReleaseAug.txt	"offenses decreased 5.4 percent"

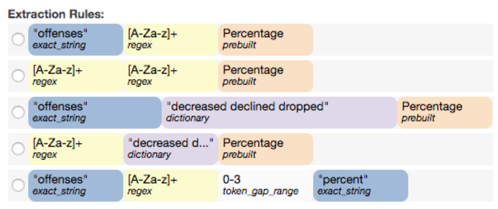
Users can export results or tweak a rule before execution.

Figure 1: SEER's Interface


Extraction Rules:

○	"offenses" *exact_string*	0-1 *token_gap_range*	Percentage *prebuilt*
○	"offenses" *exact_string*	[A-Za-z]+ *regex*	Percentage *prebuilt*
○	[A-Za-z]+ *regex*	0-1 *token_gap_range*	Percentage *prebuilt*
○	[A-Za-z]+ *regex*	[A-Za-z]+ *regex*	Percentage *prebuilt*
○	"offenses" *exact_string*	"decreased declined dropped" *dictionary*	Percentage *prebuilt*
○	[A-Za-z]+ *regex*	"decreased d..." *dictionary*	Percentage *prebuilt*

(a) Rules learned from only the positive examples.


Extraction Rules:

○	"offenses" *exact_string*	[A-Za-z]+ *regex*	Percentage *prebuilt*	
○	[A-Za-z]+ *regex*	[A-Za-z]+ *regex*	Percentage *prebuilt*	
○	"offenses" *exact_string*	"decreased declined dropped" *dictionary*	Percentage *prebuilt*	
○	[A-Za-z]+ *regex*	"decreased d..." *dictionary*	Percentage *prebuilt*	
○	"offenses" *exact_string*	[A-Za-z]+ *regex*	0-3 *token_gap_range*	"percent" *exact_string*

(b) Rules learned from the positive and negative examples.

Figure 2: Suggested rules.

audience. Examples whose trees *intersect* are grouped together [4]. Examples whose trees do not intersect result in a disjunction of different rule suggestion sets.

The examples e_1 = 'offenses dropped 9.8 percent' and e_2 = 'offenses decreased 3.2 percent' are grouped together as rules, such as $\boxed{L\text{: 'offenses'}}$ $\boxed{T\text{: 0-1}}$ $\boxed{P\text{: Percentage}}$, can capture both of the examples. Tree intersection will yield a non-empty tree of rules that captures both examples. The example e_3 = '0.5 percent increases in offenses', results in a new group as its tree cannot be intersected with the previous rule trees. The top right bar in Figure 1 would hold multiple tabs: one for each group of examples that can be captured by one intersected tree of rules.

At then end of this stage, Jane analyzes each rule. Suppose she selects the first rule, $\boxed{L\text{: 'offenses'}}$ $\boxed{T\text{: 0-1}}$ $\boxed{P\text{: Percentage}}$. Looking at the results panel, she notices a wrong extraction, 'offenses. 67.0 percent'. In the next section, we will illustrate how she can provide negative examples to eliminate such extractions.

2.4 Incorporating Negative Examples

Jane highlights the wrong extraction from the document and labels it as a negative example. She then clicks on "Suggest me some rules!" and SEER begins learning with the new set of positive and negative examples, i.e. 'offenses dropped 9.8 percent', 'offenses decreased 3.2 percent', 'offenses declined 8.0 percent' as positive examples and 'offenses. 67.0 percent' and 'offenses). In 31.0 percent' as negative examples. The new set of examples do not capture the negative examples (see Figure 2b). The rule $\boxed{L\text{: 'offenses'}}$ $\boxed{T\text{: 0-1}}$ $\boxed{P\text{: Percentage}}$ is now removed from the set of rules. As SEER generates the search space of possible rules, SEER prunes rules that capture any one of the negative examples. Thus, it only produces consistent rules. Details of how pruning works are explained in the full paper [4] and we will explain such details to the demo audience.

2.5 Filter Generated Rules

Since it is time-consuming to click through each rule to analyze the differences, Jane filters the suggested rules by accepting and rejecting extractions. Extractions that can be accepted or rejected are called *refinements* (shown in Figure 1 in the refinements panel). For instance, by rejecting the refinement 'Burglaries dropped 8.6 percent', Jane filters out the rules r_1 = $\boxed{R\text{: [A-Za-z]+}}$ $\boxed{R\text{: [A-Za-z]+}}$ $\boxed{P\text{: Percentage}}$ and r_2 = $\boxed{R\text{: [A-Za-z]+}}$ $\boxed{D\text{: \{decreased, declined, dropped\}}}$ $\boxed{P\text{: Percentage}}$.

Specifically, a refinement x is an extraction captured by a subset of the suggested rules, referred to as *covering ruleset*. SEER picks refinements from the union of all the suggested rules' extractions. An extraction is a refinement if no other existing refinements have the exact set of covering rules. For instance, if the extraction e_1 = 'Burglaries declined 3.8 percent' has the exact same covering rules as e_2 = 'Burglaries dropped 8.6 percent' and if e_2 is already a refinement, denoted as x_2, then e_1 will not be added as a refinement. Hence, the final refinements have covering rules that are unique from one another.

As Jane clicks "Yes" or "No" in the refinements panel, the rules are filtered and some of the selections in the refinements panel that have not been answered are disabled and grayed out. By disabling selections, SEER prevents *conflicting selections* to occur, where

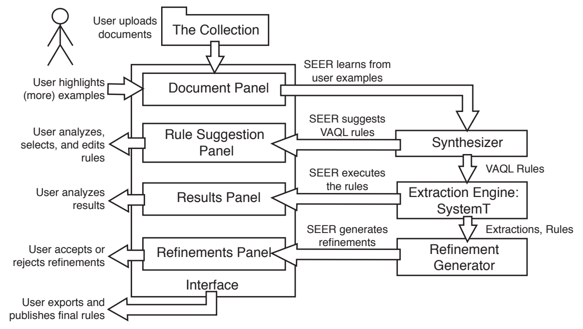


Figure 3: SEER's System Architecture

the user rejects a refinement captured by a rule that was accepted under a different refinement (or vice versa). Since Jane rejected 'Burglaries dropped 8.6 percent', the refinement 'crimes declined 3.8 percent' will be disabled as it is captured by the same rules as the rejected refinement.

2.6 Saving and Publishing Rules

After filtering the rules, Jane wishes to use the rules $r_1 =$ *L*: 'offenses' *R*: [A-Za-z]+ *P*: Percentage and $r_2 =$ *L*: 'offenses' *D*: {decreased, declined, dropped} *P*: Percentage for a larger task, such as extracting the types of crimes associated with each percentage change, e.g. 'offenses decreased 4.5% for property crimes' or 'arson offenses dropped 8.0 percent'. She can save and export a union of the rules, which she can re-import into SEER as pre-builts. She can also edit the rules before exporting them, e.g. she can add entries into the dictionary in r_2 such as 'rose'.

2.7 System Architecture

Figure 3 is a system architecture diagram of SEER. It illustrates SEER's major components such as the rule synthesizer and refinements generator as well as the interactions between different components and the user. SEER is a web application with a Java backend that executes VAQL [9] rules with the SystemT extraction engine [6].

2.8 User Studies

We conclude our demonstration by describing details of a user study we conducted for SEER [4]. In the study, participants were asked to complete IE tasks in SEER and in VINERy, a free and online tool for building IE rules in VAQL. While typically users can edit rules in SEER, the editing features of SEER were disabled in order to observe the differences in rule suggestions and rule building. We wanted to compare how fast participants completed the IE tasks and how accurate and precise the created rules were.

We measured the time to complete data extraction tasks of varying difficulty and we measured $precision = \frac{t_p}{t_p+f_p}$ (also known as accuracy), $recall = \frac{t_p}{t_p+f_n}$ (also known as coverage), and $F_1 = 2 \cdot \frac{precision \cdot recall}{precision+recall}$, where t_p denotes the number of true positives, f_p denotes the number of false positives and f_n denotes the number of false negatives. Participants using SEER finished the IE tasks faster and with more accurate and precise rules (see Figure 4).

Participants also completed a questionnaire about their experiences with SEER and VINERy. The questionnaire asked participants whether they searched the documents for actual instances of target extractions and whether they verified the extractions of the rules (instead of merely glancing at the first few extractions). The two activities are required during rule creation to have precise and accurate rules. Participants in VINERy admitted that they did not actively

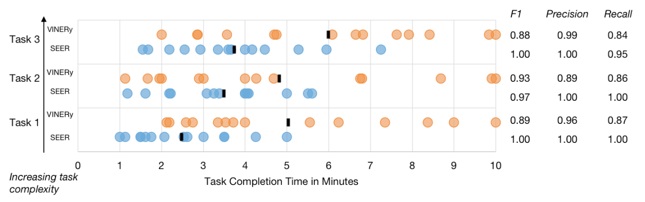


Figure 4: Time for task completion per participant [4]. Black bars indicate mean times. The mean F_1, precision, and recall are also listed.

search for actual instances of target extractions in the dataset and based their constructed rules on example extractions given to them in the experiment instructions. While participants reported feeling that they analyzed and verified more extractions in VINERy than in SEER, the actual number of extractions analyzed was higher in SEER. Users in VINERy were overly confident in the rules that they built and did not properly verify the extractions of their rules. By not searching actual instances of the extractions and not properly verifying their rules, participants built rules with lower recall and lower precision scores in VINERy. The evaluation details are in [4].

3. REFERENCES

[1] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the web. IJCAI'07, pages 2670–2676, 2007.

[2] L. Chiticariu, Y. Li, and F. R. Reiss. Rule-based information extraction is dead! long live rule-based information extraction systems! EMNLP 2013, pages 827–832, 2013.

[3] E. Ferrara, P. D. Meo, G. Fiumara, and R. Baumgartner. Web data extraction, applications and techniques: A survey. *Knowledge-Based Systems*, 70:301 – 323, 2014.

[4] M. F. Hanafi, A. Abouzied, L. Chiticariu, and Y. Li. Seer: Auto-generating information extraction rules from user-specified examples. *CHI*, 2017. *Accepted.*

[5] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: Interactive visual specification of data transformation scripts. CHI '11, pages 3363–3372, 2011.

[6] R. Krishnamurthy, Y. Li, S. Raghavan, F. Reiss, S. Vaithyanathan, and H. Zhu. Systemt: A system for declarative information extraction. *SIGMOD Rec.*, 37(4):7–13, Mar. 2009.

[7] N. Kushmerick. *Wrapper induction for information extraction*. PhD thesis, University of Washington, 1997.

[8] V. Le and S. Gulwani. Flashextract: A framework for data extraction by examples. PLDI '14, pages 542–553, 2014.

[9] Y. Li, E. Kim, M. A. Touchette, R. Venkatachalam, and H. Wang. Vinery: A visual ide for information extraction. *Proc. VLDB Endow.*, 8(12):1948–1951, Aug. 2015.

[10] B. Liu, L. Chiticariu, V. Chu, H. V. Jagadish, and F. R. Reiss. Automatic rule refinement for information extraction. *Proc. VLDB Endow.*, 3(1-2):588–597, Sept. 2010.

[11] I. Muslea, S. Minton, and C. A. Knoblock. Hierarchical wrapper induction for semistructured information sources. *AAMAS*, 4(1-2):93–114, 2001.

[12] W. Shen, A. Doan, J. F. Naughton, and R. Ramakrishnan. Declarative information extraction using datalog with embedded extraction predicates. VLDB '07, pages 1033–1044, 2007.