Regex-based Entity Extraction with Active Learning and Genetic Programming

Alberto Bartoli
DIA - University of Trieste
Italy
bartoli.alberto@units.it

Andrea De Lorenzo
DIA - University of Trieste
Italy
andrea.delorenzo@units.it

Eric Medvet
DIA - University of Trieste
Italy
emedvet@units.it

Fabiano Tarlao
DIA - University of Trieste
Italy
fabiano.tarlao@phd.units.it

ABSTRACT

We consider the long-standing problem of the automatic generation of regular expressions for text extraction, based solely on examples of the desired behavior. We investigate several active learning approaches in which the user annotates only one desired extraction and then merely answers extraction queries generated by the system.

The resulting framework is attractive because it is the system, not the user, which digs out the data in search of the samples most suitable to the specific learning task. We tailor our proposals to a state-of-the-art learner based on Genetic Programming and we assess them experimentally on a number of challenging tasks of realistic complexity. The results indicate that active learning is indeed a viable framework in this application domain and may thus significantly decrease the amount of costly annotation effort required.

CCS Concepts

•**Information systems** → **Users and interactive retrieval;** •**Mathematics of computing** → *Evolutionary algorithms;* •**Computing methodologies** → *Learning paradigms;*

Keywords

Information Extraction; Entity Extraction; Programming by Examples; Machine Learning

1. INTRODUCTION

A large class of entity *extraction* tasks from *unstructured data* may be addressed by *regular expressions*, because in many practical cases the relevant entities follow an underlying syntactical pattern and this pattern may be described by a regular expression. A long-standing problem in this area consists in the *automatic* generation of a regular expression suitable for a specific task based solely on *examples* of the desired behavior.

A wealth of research efforts in this area considered *classification* problems either in *formal* languages [12, 18, 39, 17, 20, 23] or in the realm of *deterministic finite automata* (DFA) [25, 15, 10, 29]. Those results considered scenarios that do not fit practical text processing applications, which have to cope with much longer sequences of symbols drawn from a much larger alphabet. Text *extraction* problems of non trivial size and complexity were first considered in a procedure that automatically optimized an initial regular expression to be provided by the user based on examples of the desired functioning [27]. Later proposals still required an initial regular expression but were more robust toward initial expressions of modest accuracy and noisy datasets [3, 32]. The need of an initial solution was later removed in several proposal [14, 11, 4]. A more recent proposal based on Genetic Programming advanced significantly over earlier approaches and is capable of addressing text extraction tasks of practical complexity effectively, with a few tens of examples of the desired behavior [5, 6].

In this work, we investigate the feasibility of an *active learning* approach for relieving the user from the need of examining the full input text (i.e., the dataset) in search of all the desired extractions to be annotated for learning [1, 36, 38, 13, 28]. We develop and evaluate experimentally a framework in which the user initially marks *only one* snippet of the input text as desired extraction. A learner based on Genetic Programming then constructs a solution, digs into the (possibly very long) input text, selects the most appropriate snippet to be used for improving the current model and presents it to the user as an *extraction query*. The user merely answers the query by specifying whether the selected snippet has to be extracted or not extracted and the process continues iteratively, improving the solution at each query.

The resulting framework is highly attractive and may greatly broaden the potential scope of automatic regex generation from examples. On the other hand, actually implementing this framework is challenging because the scenario presents significant differences from successful applications of active learning.

Active learning approaches usually consider datasets where each item is an input instance and thus a candidate query. This property is shared also by approaches based on Ge-

t	`I␣was␣born␣in␣1979␣and␣he␣was␣born␣in␣1974.`
s_q	`I␣was␣born␣in␣1979␣and␣he␣was␣born␣in␣1974.`
M, U	`I␣was␣born␣in␣1979␣and␣he␣was␣born␣in␣1974.`

Figure 1: Oracle annotation example: desired (undesired) extractions are in dark (light) gray; the query is boxed.

netic Programming [16, 33, 22]. Our case is different because the dataset is a single, possibly long, input text without any native segmentation in smaller units. Depending on the application, it may consist of one very long line or several lines with possibly variable length; furthermore, more than one desired extractions may occur within a single line (e.g., IP addresses in network logs) or a single desired extraction may span across several lines (e.g., HTML elements including their content). Assuming that the text is natively segmented in lines or in sentences (as in, e.g., [19]) would severely restrict the scope of possible applications of the system. Moreover, the size of the query to be presented to the user should be chosen carefully. Presenting a large snippet (e.g., one or more entire lines) to the user for annotation may nullify the objective of minimizing user annotation effort. On the other extreme, repeatedly asking the user to annotate very short snippets may not be effective.

In other words, not only we have the problem of *choosing* the next query among candidate queries, we also have the problem of *constructing* candidate queries out of the available input text. In this respect, it is useful to remark that the number of possible queries in (i.e., the number of snippets of) an input text grows quadratically with the text size and becomes huge very quickly—e.g., even if we assume that the learner cannot generate queries ex novo and can only query a snippet of the input text, if the latter size is just 10^5characters then there are $\approx 10^{10}$ candidate queries. Furthermore, active learning usually targets scenarios with hundreds of queries (e.g., [38, 13, 28]) whereas we must be able to improve over a random query chooser and provide solutions of good quality even with a few tens of examples, similarly to [22].

Our contribution consists in:(a) a model for the external oracle that may participate in the construction of queries, which improves the quality of annotation information while at the same time maintaining a behavior very intuitive to unskilled users; (b) a technique for constructing queries suitable to regex-based entity extraction from unstructured text, which does not assume any internal segmentation of input text; (c) an implementation of several active learning approaches taking into account the need of constructing candidate queries; (d) a novel variant for the learner in which the number of generations executed between consecutive queries may vary dynamically depending on the quality of the current solution; and, (e) an experimental analysis on a number of challenging datasets of several active learning approaches, which target different accuracy/annotation effort regions of the design space.

2. OUR APPROACH

The problem consists in generating a regular expression automatically based on examples of the desired extraction behavior on a text t. Such examples are *annotations*: snippets of t that are to be extracted (*matches*) or snippets of t that are not to be extracted (*unmatches*).

We propose an approach based on active learning, as follows. Initially an external *oracle*, i.e., the user, annotates an extremely small portion of t—we experimented with only one match. The learner consists of three components: the *solver*, which generates a regular expression suited to the matches and unmatches annotated by the oracle so far; the *query trigger*, which determines *when* a query has to be proposed to the oracle; and the *query builder*, which *constructs* candidate queries and determines *which query* should be proposed to the oracle.

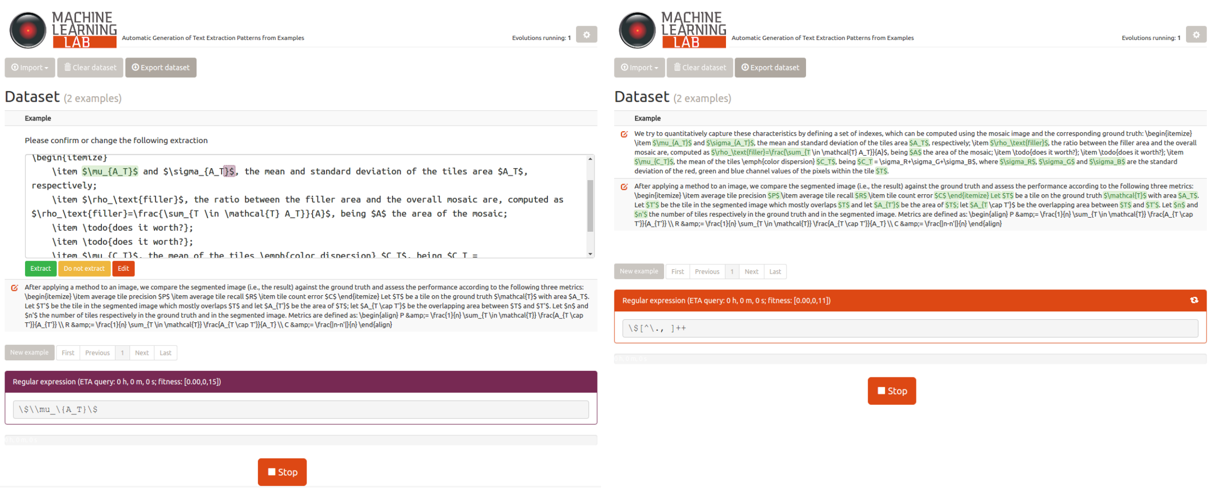
Each query consists of a snippet of t, denoted s_q, to be annotated by the oracle. We propose the following behavior for the oracle: the oracle's answer is a pair M, U, where M is the (possibly empty) set of all matches which overlap s_q and U is the (possibly empty) set of maximal subsnippets of s_q which are unmatches—Figure 1 shows an example of annotation.

In other words, we propose an oracle that may modify the received query slightly and then answer the modified query. With most active learning approaches the user is required to provide the class of queried data and is not allowed to modify those data. The proposed behavior for the oracle is very practical and is easily implemented with a GUI, though. When the queried snippet consists exactly of a desired extraction or does not contain any desired extraction, one single click suffices to answer the query. Otherwise, when the query partly overlaps a match, the user is expected to expand the query on either or both sides—an action which is more intuitive to unskilled users, nevertheless results in answers which are more informative to the learner.

We developed a web-based prototype with a GUI that efficiently implements the proposed interaction model. Figure 2 shows how the user interface appears when a query is made (left) and while the learning algorithm is running (right). In the first case, a query is shown as a highlighted portion of the text (in purple) t and the user is presented with 3 buttons: 'Extract", "Do not extract" and "Edit". When the query corresponds exactly to a desired extraction or does not contain any desired extraction, then one single click suffices to answer the query (button "Extract" or "Do not extract", respectively). Otherwise, when the user has to describe a more complex answer, by clicking the "Edit" button the user may extend the selection boundaries of the query and delimit desired extractions precisely. The GUI also highlights (in green) the extractions of the current best solution, in order to help the user in understanding the behaviour of the current solution. The state of the current solution is reported also while the search is in progress, as illustrated in the left part of Figure 2. The aim of this design is to help the user in deciding when to stop the regex search—i.e., when the user is satisfied by the current solution.

The solver is based on the proposal in [6, 7], whose code is publicly available[1]. The proposal is based on Genetic Programming [24]: a population of regular expressions, rep-

[1] `https://github.com/MaLeLabTs/RegexGenerator`



Please confirm or change the following extraction

```
\begin{itemize}
    \item $\mu_{A_T}$ and $\sigma_{A_T}$, the mean and standard deviation of the tiles area $A_T$,
respectively;
    \item $\rho_\text{filler}$, the ratio between the filler area and the overall mosaic are, computed as
$\rho_\text{filler}=\frac{\sum_{T \in \mathcal{T} A_T}}{A}$, being $A$ the area of the mosaic;
    \item \todo{does it worth?};
    \item \todo{does it worth?};
    \item $\mu_{C_T}$, the mean of the tiles \emph{color dispersion} $C_T$, being $C_T =
```

```
After applying a method to an image, we compare the segmented image (i.e., the result) against the ground truth and assess the performance according to the following three metrics: \begin{itemize} \item average tile precision $P$ \item average tile recall $R$ \item tile count error $C$ \end{itemize} Let $T$ be a tile on the ground truth $\mathcal{T}$ with area $A_T$. Let $T'$ be the tile in the segmented image which mostly overlaps $T$ and let $A_{T'}$ be the area of $T$; let $A_{T \cap T'}$ be the overlapping area between $T$ and $T'$. Let $n$ and $n'$ the number of tiles respectively in the ground truth and in the segmented image. Metrics are defined as: \begin{align} P &amp;= \frac{1}{n} \sum_{T \in \mathcal{T}} \frac{A_{T \cap T'}}{A_{T'}} \\ R &amp;= \frac{1}{n} \sum_{T \in \mathcal{T}} \frac{A_{T \cap T'}}{A_T} \\ C &amp;= \frac{|n-n'|}{n} \end{align}
```

Regular expression (ETA query: 0 h, 0 m, 0 s; fitness: [0.00,0,15])

```
\$\\mu_\{A_T}\$
```

```
We try to quantitatively capture these characteristics by defining a set of indexes, which can be computed using the mosaic image and the corresponding ground truth: \begin{itemize} \item $\mu_{A_T}$ and $\sigma_{A_T}$, the mean and standard deviation of the tiles area $A_T$, respectively; \item $\rho_\text{filler}$, the ratio between the filler area and the overall mosaic are, computed as $\rho_\text{filler}=\frac{\sum_{T \in \mathcal{T} A_T}}{A}$, being $A$ the area of the mosaic; \item \todo{does it worth?}; \item \todo{does it worth?}; \item $\mu_{C_T}$, the mean of the tiles \emph{color dispersion} $C_T$, being $C_T = \sigma_R+\sigma_G+\sigma_B$, where $\sigma_R$, $\sigma_G$ and $\sigma_B$ are the standard deviation of the red, green and blue channel values of the pixels within the tile $T$.
```

```
After applying a method to an image, we compare the segmented image (i.e., the result) against the ground truth and assess the performance according to the following three metrics: \begin{itemize} \item average tile precision $P$ \item average tile recall $R$ \item tile count error $C$ \end{itemize} Let $T$ be a tile on the ground truth $\mathcal{T}$ with area $A_T$. Let $T'$ be the tile in the segmented image which mostly overlaps $T$ and let $A_{T'}$ be the area of $T$; let $A_{T \cap T'}$ be the overlapping area between $T$ and $T'$. Let $n$ and $n'$ the number of tiles respectively in the ground truth and in the segmented image. Metrics are defined as: \begin{align} P &amp;= \frac{1}{n} \sum_{T \in \mathcal{T}} \frac{A_{T \cap T'}}{A_{T'}} \\ R &amp;= \frac{1}{n} \sum_{T \in \mathcal{T}} \frac{A_{T \cap T'}}{A_T} \\ C &amp;= \frac{|n-n'|}{n} \end{align}
```

Regular expression (ETA query: 0 h, 0 m, 0 s; fitness: [0.00,0,11])

```
\$[^\., ]++
```

Figure 2: Screenshots of the web-based prototype developed for our framework: query submitted to the user (left), learning based on the currently available annotations (right).

resented by abstract syntax trees, is iteratively evolved by applying the genetic operators across many iterations (*generations*). A multiobjective optimization algorithm drives evolution of regular expressions according to their length (to be minimized) and their extraction performance computed on the matches and unmatches (to be maximized). We refer the reader to the cited paper for full details.

We considered two variants for the query trigger. The *Const* variant has been used in other active learning proposals for Genetic Programming [16, 33, 22] and generates a new query whenever a predefined number of generations of the solver has been executed. W experimented with 30 and with 200 generations. The *Solved* variant is an optimization that we explore in this work. This variant triggers the query builder when the best regular expression in the population, as assessed on the current set of matches and unmatches, has remained unchanged for a predefined number of generations of the solver—i.e., a new query is triggered when no further progress seems to be achievable with the available annotations. We experimented with 200 generations, i.e., one of the values selected for the *Const* variant, in order to assess the accuracy/speed trade-off of the two variants.

The query builder constructs candidate queries based on the notion of *disagreement*: given a set C of regular expressions (the *committee*), we define as disagreement of C on a character c of the input text t the quantity $d_C(c) = 1 - 2\text{abs}\left(\frac{1}{2} - \frac{|C_c|}{|C|}\right)$, where $C_c \subseteq C$ is the subset of regular expressions which extract c—$d_C(c) = 1$ if half of the committee extracts c (maximum disagreement), $d_C(c) = 0$ if the all committee agrees on the processing of c (minimum disagreement). Note that we quantify disagreement based on the class chosen by each candidate solution in C (extracted vs. not extracted) [30] without any reference to forms of confidence value, margin or probability [26, 31]. As we pointed out already in the introduction, such notions are not made available by the solver that we have chosen to use.

The procedure for constructing candidate queries takes a set of regular expressions C as parameter and determines the character $c^* \in t$ with maximal disagreement $d_C(c^*)$ in the full input set t. Next, the procedure determines the set S of candidate queries as the set composed of all snippets of t which meet the following conditions: they (a) are extracted by at least a regular expression in C, (b) overlap c^*, and (c) do not overlap any available annotation.

We implemented two variants of a query builder. The *Query by committee (QbC)* variant works as follows: (a) construct the set S of candidate queries using the full population as committee C, (b) compute, for each snippet in S, the average disagreement among the characters of the snippet, and (c) choose the snippet with minimal average disagreement as query. The *Query by restricted committee (rQbC)* variant is similar to QbC except that the committee C contains only the best 25% of the current population (ranking being based on the current set of matches and unmatches).

QbC and rQbC are based on a principle widely used in active learning [34, 36], i.e., on the assumption that the query for which an ensemble of competing hypotheses exhibits maximal disagreement is the most informative for the learning task [37]. Such a principle has been used also in active learning for Genetic Programming [16, 33, 22]—in those scenarios there is the problem of choosing a candidate query but not the one of constructing queries, though. Indeed, the proposal in [22] augments this principle by also taking into account a measure of diversity between each candidate query and queries already answered. Our preliminary exploration of this additional principle, that we do not illustrate for space reasons, has not delivered satisfactory results. We believe the reason consists in the difficulty of finding a diversity measure for text snippets correlated with diversity between regular expressions—e.g., two text snippets could be very different while at the same time they could be captured by the same regular expression or by regular expressions that are very similar.

Concerning query builders we also observe that a wealth of active learning approaches choose queries based on *uncertainty* of the current solution, especially when the learner is not based on an ensemble of competing hypotheses [26, 35, 34, 38]. On the other hand, such approaches do not fit the state-of-the-art regex learner that we use in our system, because such a learner does not provide any confidence level about the handling of a given snippet (i.e., extracted vs. not extracted) by the current solution.

We also implemented a third query builder that randomly chooses an unannotated snippet. We place an upper bound to the maximum length of the query that may be generated: we set the actual bound value in our experimental evaluation to the maximum size of a desired extraction across all our datasets (few hundreds characters). The upper bound causes this query builder to filter out candidate queries which are too long, which hence advantages this builder w.r.t. one which selects a truly random snippet of t. For this reason, we call this builder *SmartRand*.

3. EXPERIMENTS

We focused on the *extraction performance* of the regular expression generated for a given amount of *user annotation effort*. We quantify extraction performance with F-measure (Fm), which is the harmonic mean of precision (ratio between the number of correctly extracted snippets and the number of all the extracted snippets) and recall (ratio between the number of correctly extracted snippets and the number of all the snippets which should have been extracted). We chose to quantify user annotation effort by the number of annotated characters (AC).

We evaluated all the 9 combinations between the proposed design variants and we considered 11 challenging extraction tasks used in [6]. For each extraction task, we randomly selected a subset of the original corpus containing approximately 100 desired extractions. The name of each extraction task can be seen—along with the size of the input text expressed in number of characters—in Table 1: it is composed of the name of the corpus followed by the name of the entity type to be extracted.

We assessed our system variants as follows. For each task and variant, we chose a random desired extraction as the only starting annotated snippet and executed the variant with a simulated oracle. We repeated the above procedure 15 times, with 5 different starting matches and 3 different random seeds. We terminated each execution upon the query for which either at least 25% of the available characters was annotated or the F-measure on the full input text (i.e., not only on the annotated portion) was 1. Although a real deployment cannot quantify F-measure on a yet unannotated input text, we chose to include the latter condition in the termination criterion in order to provide a fair assessment of variants which are able to generate perfect solutions before reaching the predefined annotation budget. We chose 25% of the available characters as annotation budget because we have found that, with these datasets, it corresponds to a few minutes of actual annotation.

Table 1 shows the main results (statistical significance is analyzed in more detail later). For each task, Fm is computed on the full input text and averaged across the 15 repetitions of each experiment. Values in the bottom rows of the table are averaged across all tasks. We define the *computational effort* (CE) as the number of characters analyzed for fitness evaluations across an execution. This quantity is a hardware-independent performance index. Execution times are in the order of minutes, similarly to [6], we do not list them in detail for space reasons: the time taken by the query trigger and the query builder is negligible w.r.t. the time taken by the solver.

It can be seen that for nearly all tasks several of our active learning variants are able to generate regular expressions of very good quality. This result is significant because it strongly suggests that active learning is indeed a viable framework for the task of automatic generation of regular expressions.

Another important outcome is that the rQbC query builder tends to deliver better F-measure than the SmartRand query builder while requiring less annotations—ΔFm between 0.05 and 0.1 on the average. In many applications of active learning, a random query chooser is often quite effective and often turns out to be a challenging baseline for more sophisticated query choice strategies [2, 21, 38]. Although we may observe this phenomenon also in our scenario (in which the random selection is enhanced by a lenght-based filtering, see Section 2), we also observe a clear superiority of approaches based on rQbC. The QbC query builder, on the other hand, is not effective as it tends to exhibit worse results from the three points of view summarized in the table: F-measure, annotation effort, computational effort.

We speculate that the superiority of rQbC over SmartRand may become even more apparent with datasets in which the density of desired extractions is smaller than ours—in our datasets, the likelihood of randomly choosing a snippet that partly overlaps a desired extraction is not very small. We need to investigate this conjecture further, however.

Concerning the behavior of query triggers with rQbC, it can be seen that each of the three options analyzed belongs to a different region of the design space. The Const30 query trigger is much faster (CE) at the expense of obtaining a relatively good but smaller F-measure, while at the same time requiring more annotations (AC). Const200 and Solved represent more useful trade-offs because they deliver the best average F-measure: they require the same amount of annotations, trading a small difference in F-measure for a substantial difference in computational effort.

In order to illustrate the significance of these results further, we executed the state-of-the-art learner proposed in [6] on the same tasks. This learner requires a training set fully annotated before starting execution. For each task we randomly generated 5 training sets, each one with 25% of the available characters and with a random generation procedure carefully tailored so as to ensure that each training set contains approximately 25% of the desired extractions. It may be useful to emphasize that the size of the training set corresponds to the size of the training set of active learning upon the *last* query: in this case the training set is instead available to the solver for the *full* execution; furthermore, in active learning the user need not take any effort to dig out an adequate amount of desired extractions from the (potentially large) available data. We executed each task 5 times, each execution using one of the 5 different training

Table 1: The F-measure obtained with each variant on each task. The average F-measure, CE and AC are also shown.

		QbC	QbC	QbC	SmartRand	SmartRand	SmartRand	rQbC	rQbC	rQbC
Task	Size	Const30	Const200	Solved	Const30	Const200	Solved	Const30	Const200	Solved
ReLIE-HTML/All-URL	16 655	0.61	0.82	0.76	0.75	0.86	0.82	0.69	0.84	0.85
ReLIE-Email/Phone-Num.	18 123	0.97	0.99	0.99	0.57	0.95	0.70	0.97	0.98	0.99
Cetinkaya-HTML/href	14 922	0.77	0.87	0.88	0.95	1.00	1.00	0.81	1.00	1.00
Cetinkaya-Text/All-URL	7573	0.98	0.99	0.99	0.99	0.99	0.99	0.98	0.99	0.99
Twitter/Hashtag+Citation	5308	0.91	0.93	0.98	0.98	0.99	0.99	0.92	0.99	0.99
Twitter/All-URL	9537	0.92	0.92	1.00	1.00	0.92	1.00	0.92	1.00	1.00
Log/IP	5766	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Log/MAC	10 387	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Email-Headers/IP	36 925	0.89	0.80	0.94	0.39	0.85	0.53	0.69	0.71	0.77
NoProfit-HTML/Email	4651	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Web-HTML/Heading	37 678	0.51	0.54	0.54	0.81	0.75	0.82	0.52	0.83	0.60
Average Fm		0.87	0.89	0.91	0.86	0.94	0.89	0.86	0.94	0.92
Average AC		3311	3202	2734	2997	2864	2646	3238	2506	2525
Average CE ($\times 10^9$)		6.5	45.4	44.2	4.3	33.6	27.3	7.2	40.0	27.2

sets. We obtained, on average, Fm = 0.97, CE = 29.8×10^9 and AC = 3748, i.e., 49% more annotated characters than rQbC-Const200 and 48% more than rQbC-Solved.

We performed an analysis of the statistical significance of the results based on the Wilcoxon signed-rank test: we chose this test since it is non-parametric and does not require the population to be normally distributed. The results are in Table 2 (F-measure, above, and annotated characters, AC, below)—we omit results about CE for space reasons. In each table, cell (i, j) contains the difference in the average value of the corresponding performance index between variant in row i and variant in row j. Statistical significance of performance index comparison is indicated for varying p-values of the test and highlighted with asterisks.

These results confirm the analysis of Table 1, but they also indicate that the rQbC/Const200 and rQbC/Solved actually does not guarantee any statistically significant improvement in Fm over SmartRand/Const200. On the other hand, there is indeed some statistically significant evidence of an improvement in terms of smaller annotation effort—12.5% for rQbC/Const200 and 11.8% for rQbC/Solved. Concerning CE (not shown for space reasons), rQbC/Const200 requires 19% more character evaluations but this result is not statistically significant; rQbC/Solved instead requires 19% less character evaluations with the strongest statistical significance.

Figure 3 illustrates the trade-off AC vs. F-measure (left) and AC vs. CE (right). The figure contains one point for each task; the different query triggers are represented as points of different colors while the different query builders are represented with different shapes. For each point, F-measure, AC and CE are averaged among 15 experiment repetitions—5 folds and 3 different random seeds.

In the left figure it can be seen that points representing the Const30 query trigger—light grey points—tend to be distributed in the rightmost and lower part of the figure—i.e., this query trigger requires high AC but obtains low F-measure. Points representing the Const200 and Solved query trigger—dark gray and black points—tend instead to be distributed in the leftmost and higher part of the figure, i.e., for each AC value we may obtain high F-measure values. Concerning query builders, the graphical distribution of points does not provide any significant insights; in this respect, the other analyses discussed previously are more effective. In the right figure shows for each point the average CE vs the average AC for one task it can be seen that points representing the Const200 and Solved query triggers tend to be distributed in the highest part of the figure, as expected Const200 and Solved query triggers require CE values higher than the Const30 ones. We may note that the points representing the SmartRand query builder tend to occupy the highest part of the figure, in other words SmartRand query builders require CE values higher than the QbC and rQbC ones.

Finally, in Table 3 we report the detailed execution trace of two significant experiments based on the rQbC/Solved configuration: one for the Twitter/Hashtag+Citation task and another for the Email-Headers/IP task. The table contains one row for each query constructed by the system. Each row contains: the sequential number of the query; the number of annotated matches $|\bigcup M|$ and unmatches $|\bigcup U|$, available to the learning algorithm; the content of the query s_q; the response provided by the user in terms of desired matches M and desired unmatches U. Each row also contains the currently best solution, along with the F-measure associated with such solution and the total amount of AC.

4. CONCLUDING REMARKS

We have proposed several active learning approaches tailored to the automatic generation of regular expressions for entity extraction from unstructured text. We have assessed these approaches experimentally on a number of challenging extraction tasks that have been previously used in the literature. The results indicate that active learning, starting with only one annotated match, is indeed a viable framework for this application domain and may thus significantly decrease the amount of costly user annotation effort. We have also identified design options and explored the design space, in terms of computational effort and annotation effort, while delivering very good F-measure. We believe that our results are significant and highly promising.

As future work we intend to broaden the experimental anal-

Table 2: Average differences of Fm and AC of pairs of the proposed variants. For each pair, the statistical significance is shown: *: $p < 0.1$, **: $p < 0.05$, ***: $p < 0.01$ (the last condition corresponds to the strongest statistical significance; absence of any asterisk indicates that the comparison is not statistically significant, i.e., $p \geq 0.1$).

F-measure (Fm)

Variant	QbC Const30	QbC Const200	QbC Solved	SmartRand Const30	SmartRand Const200	SmartRand Solved	rQbC Const30	rQbC Const200	rQbC Solved
QbC/Const30		−0.03***	−0.05***	0.01	−0.07***	−0.03**	0.01*	−0.07***	−0.05***
QbC/Const200	0.03***		−0.02*	0.04**	−0.04**	0.00	0.03***	−0.04***	−0.03***
QbC/Solved	0.05***	0.02*		0.06***	−0.02	0.02	0.05***	−0.02**	−0.01***
SmartRand/Const30	−0.01	−0.04**	−0.06***		−0.08***	−0.04***	−0.01	−0.08***	−0.07***
SmartRand/Const200	0.07***	0.04**	0.02	0.08***		0.04***	0.07***	0.00	0.01
SmartRand/Solved	0.03**	0.00	−0.02	0.04***	−0.04***		0.03**	−0.04***	−0.03***
rQbC/Const30	−0.01*	−0.03***	−0.05***	0.01	−0.07***	−0.03**		−0.07***	−0.06***
rQbC/Const200	0.07***	0.04***	0.02**	0.08***	0.00	0.04***	0.07***		0.01
rQbC/Solved	0.05***	0.03***	0.01***	0.07***	−0.01	0.03***	0.06***	−0.01	

Annotated characters (AC)

Variant	QbC Const30	QbC Const200	QbC Solved	SmartRand Const30	SmartRand Const200	SmartRand Solved	rQbC Const30	rQbC Const200	rQbC Solved
QbC/Const30		109***	577***	314***	447***	665***	73**	805***	786***
QbC/Const200	−109***		468***	205**	338***	556***	−35	696***	677***
QbC/Solved	−577***	−468***		−263**	−129	88	−503***	228**	209**
SmartRand/Const30	−314***	−205**	263**		133***	351***	−240***	491***	472***
SmartRand/Const200	−447***	−338***	129	−133***		218***	−374***	358*	338*
SmartRand/Solved	−665***	−556***	−88	−351***	−218***		−592***	140	120
rQbC/Const30	−73**	35	503***	240***	374***	592***		732***	712***
rQbC/Const200	−805***	−696***	−228**	−491***	−358*	−140	−732***		−19***
rQbC/Solved	−786***	−677***	−209**	−472***	−338*	−120	−712***	19***	

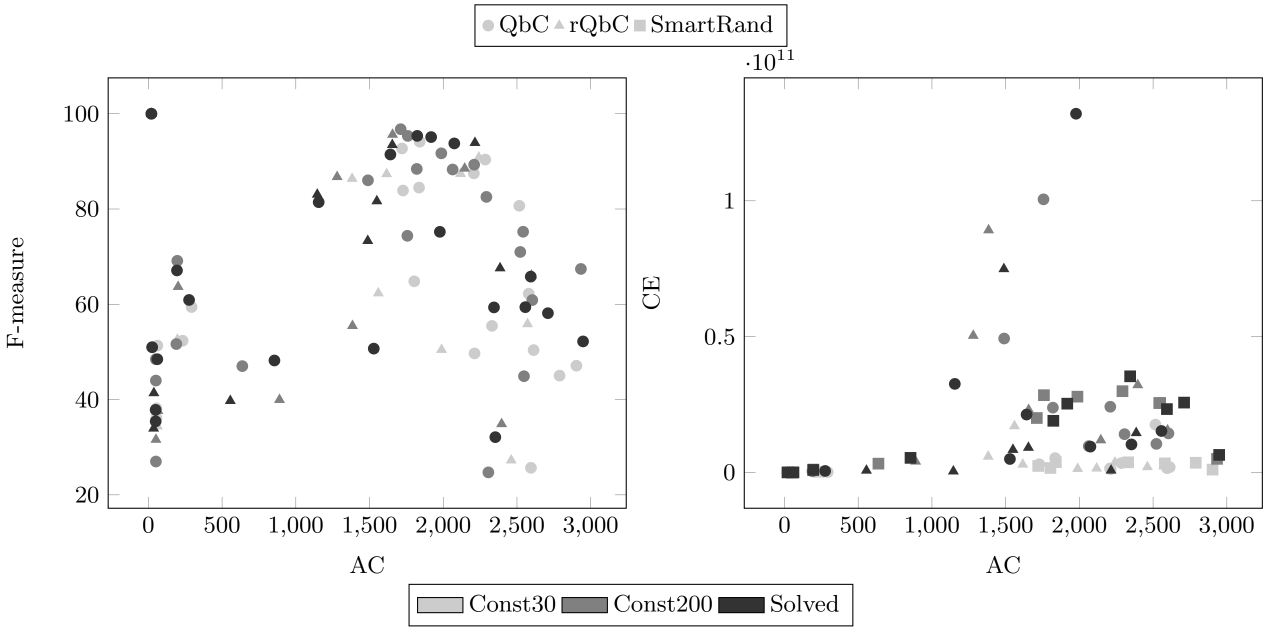


Figure 3: AC vs. F-measure (left) or vs. CE (right): one point for each task (corresponding to the average index across the repetitions).

Table 3: Sequences of queries generated for two different experiments. For each query are reported the total number of matches and unmatches annotated, the query s_q, the user answer in terms of M and U, the current best solution, the corresponding F-measure and the current AC.

#	$\|\bigcup M\|$	$\|\bigcup U\|$	s_q	M	U	Best regex	F-m	AC
1	1	0	#20topsongsever	#20topsongsever		#\w++	0.39	24
2	2	0	#hacking	#hacking		#\w++	0.39	32
3	3	0	#ti	#tips		#\w++	0.39	37
4	4	0	#O	#OpPiggyBank		#\w++	0.39	49
5	5	0	#plurfamily	#plurfamily		#\w++	0.39	60
6	6	1	#FF␣mee␣!!!"	#FF	␣mee␣!!!"	#\w++	0.39	72
7	7	2	#bast@Rd	#bast	@Rd	#\w++	0.39	80
8	8	2	@Callum	@Callum_Rose		[@#]\w++	1.00	92
1	1	0	199.87	199.87.247.43		199\.87\.247\.43	0.08	26
2	2	0	209.85.216.170	209.85.216.170		\w++\.\w++\.\w++\.\w++	0.77	40
3	3	0	10.2	10.231.24.9		\w++\.\w++\.\w++\.\w++	0.77	51
4	4	2	:␣by␣10.231.102.195␣with␣SMTP	10.231.102.195	:␣by␣ ␣with␣SMTP	(?:\d++\.)++\d++	0.68	80
5	5	2	10.236.195.3	10.236.195.34		(?:\w++\.)++\d++	0.66	93
6	5	3	go2mr11586177wib.22		go2mr11586177wib.22	\w++\.\w++\.\w++\.\w++	0.77	112
7	5	4	etPan.528e775f.6ce90669.a		etPan.528e775f.6ce90669.a	\d++\.\d++\.\d++\.\d++	0.92	137
8	6	4	199.7.202.190	199.7.202.190		\d++\.\d++\.\d++\.\d++	0.92	150
9	7	4	199.7.202.190	199.7.202.190		\d++\.\w++\.\w++\.\w++	0.84	163
10	7	5	Exim␣4.80.1		Exim␣4.80.1	\d++\.\w++\.\w++\.\w++	0.84	174
11	7	6	h6mr48792qew.9		h6mr48792qew.9	\w++\.\w++\.\w++\.\d++	0.87	188
12	7	7	5.1gphBQfkbkwG8rjXKOhM		5.1gphBQfkbkwG8rjXKOhM	\w++\.\w++\.\w++\.\d++	0.87	210
13	7	8	x8mr5889809oek.49.1		x8mr5889809oek.49.1	\w++\.\w++\.\w++\.\d++	0.87	229
14	7	9	jx4mr3506406vec.35.1		jx4mr3506406vec.35.1	\w++\.\w++\.\w++\.\d++	0.87	249
15	7	10	6.0.3790.4		6.0.3790.4	\w\w++\.\w++\.\d++\.\d++	0.91	259
16	7	11	1.2013.11.11		1.2013.11.11	\w\d++\.[^1]\w*+\.(?!3)\d++\.\d++	0.80	271
18	8	13	217.12.10.166;	217.12.10.166	;	\w\d++\.\w++\.\d*+\.\d++	0.95	360

ysis by taking into account more facets of the user effort, including a measure of the user annotation time as a function of the number, length and complexity of individual queries. We also intend to devise a suitable metric for taking into account the user cost broadly involved in the elapsed time between consecutive queries. Finally, we may explore the feasibility of an online estimate of the difficulty of obtaining a suitable regular expression given the current set of annotations, basing on the work in [8, 9].

5. REFERENCES

[1] D. Angluin. Queries and concept learning. *Machine learning*, 2(4):319–342, 1988.

[2] J. Atserias, M. Simi, and H. Zaragoza. H.: Active learning for building a corpus of questions for parsing. In *In: Proceedings of LREC 2010*, 2010.

[3] R. Babbar and N. Singh. Clustering based approach to learning regular expressions over large alphabet for noisy unstructured text. In *Proceedings of the Fourth Workshop on Analytics for Noisy Unstructured Text Data*, AND '10, pages 43–50, New York, NY, USA, 2010. ACM.

[4] D. F. Barrero, M. D. R-Moreno, and D. Camacho. Adapting searchy to extract data using evolved wrappers. *Expert Systems with Applications*, 39(3):3061–3070, Feb. 2012.

[5] A. Bartoli, G. Davanzo, A. De Lorenzo, M. Mauri, E. Medvet, and E. Sorio. Automatic generation of regular expressions from examples with genetic programming. In *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '12, pages 1477–1478, New York, NY, USA, 2012. ACM.

[6] A. Bartoli, G. Davanzo, A. De Lorenzo, E. Medvet, and E. Sorio. Automatic synthesis of regular expressions from examples. *Computer*, 47(12):72–80, Dec 2014.

[7] A. Bartoli, A. De Lorenzo, E. Medvet, and F. Tarlao. Inference of regular expressions for text extraction from examples. *Knowledge and Data Engineering, IEEE Transactions on*, 28(5):1217–1230, 2016.

[8] A. Bartoli, A. D. Lorenzo, E. Medvet, and F. Tarlao. Predicting the effectiveness of pattern-based entity extractor inference. *Applied Soft Computing*, 46:398 – 406, 2016.

[9] A. Bartoli, A. D. Lorenzo, E. Medvet, and F. Tarlao. Syntactical similarity learning by means of grammatical evolution. In *14-th Parallel Problem Solving from Nature (PPSN)*, 2016. to appear.

[10] J. Bongard and H. Lipson. Active coevolutionary learning of deterministic finite automata. *The Journal of Machine Learning Research*, 6:1651–1678, 2005.

[11] F. Brauer, R. Rieger, A. Mocan, and W. M. Barczynski. Enabling information extraction by inference of regular expressions from sample entities. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1285–1294. ACM, 2011.

[12] A. Brāzma. Efficient identification of regular expressions from representative examples. In *Proceedings of the Sixth Annual Conference on Computational Learning Theory*, COLT '93, pages 236–242, New York, NY, USA, 1993. ACM.

[13] W. Cai, M. Zhang, and Y. Zhang. Active learning for ranking with sample density. *Information Retrieval Journal*, 18(2):123–144, 2015.

[14] A. Cetinkaya. Regular expression generation through grammatical evolution. In *Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation*, pages 2643–2646. ACM,

2007.

[15] O. Cicchello and S. C. Kremer. Inducing grammars from sparse data sets: a survey of algorithms and results. *The Journal of Machine Learning Research*, 4:603–632, 2003.

[16] J. De Freitas, G. L. Pappa, A. S. Da Silva, M. Gonçalves, E. Moura, A. Veloso, A. H. Laender, M. G. De Carvalho, et al. Active learning genetic programming for record deduplication. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE, 2010.

[17] F. Denis. Learning regular languages from simple positive examples. *Machine Learning*, 44(1-2):37–66, 2001.

[18] B. Dunay, F. Petry, and B. Buckles. Regular language induction with genetic programming. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pages 396–400 vol.1, Jun 1994.

[19] A. Esuli, D. Marcheggiani, and F. Sebastiani. Sentence-based active learning strategies for information extraction. In *IIR*, pages 41–45, 2010.

[20] H. Fernau. Algorithms for learning regular expressions from positive data. *Information and Computation*, 207(4):521 – 541, 2009.

[21] R. L. Figueroa, Q. Zeng-Treitler, L. H. Ngo, S. Goryachev, and E. P. Wiechmann. Active learning for clinical text classification: is it better than random sampling? *Journal of the American Medical Informatics Association*, 19(5):809–816, 2012.

[22] R. Isele and C. Bizer. Active learning of expressive linkage rules using genetic programming. *Web Semantics: Science, Services and Agents on the World Wide Web*, 23:2–15, 2013.

[23] E. Kinber. Learning regular expressions from representative examples and membership queries. *Grammatical Inference: Theoretical Results and Applications*, pages 94–108, 2010.

[24] J. R. Koza. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

[25] K. J. Lang, B. A. Pearlmutter, and R. A. Price. Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In *Grammatical Inference*, page 1–12. Springer, 1998.

[26] D. D. Lewis and W. A. Gale. A sequential algorithm for training text classifiers. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 3–12. Springer-Verlag New York, Inc., 1994.

[27] Y. Li, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Jagadish. Regular expression learning for information extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 21–30. Association for Computational Linguistics, 2008.

[28] Z. Lu, X. Wu, and J. C. Bongard. Active learning through adaptive heterogeneous ensembling. *Knowledge and Data Engineering, IEEE Transactions on*, 27(2):368–381, 2015.

[29] S. M. Lucas and T. J. Reynolds. Learning deterministic finite automata with a smart state labeling evolutionary algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1063–1074, 2005.

[30] N. A. H. Mamitsuka. Query learning strategies using boosting and bagging. In *Machine Learning: Proceedings of the Fifteenth International Conference (ICML'98)*, page 1. Morgan Kaufmann Pub, 1998.

[31] P. Melville and R. J. Mooney. Diverse ensembles for active learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 74. ACM, 2004.

[32] K. Murthy, D. P., and P. M. Deshpande. Improving recall of regular expressions for information extraction. In *Web Information Systems Engineering - WISE 2012*, volume 7651 of *Lecture Notes in Computer Science*, pages 455–467. Springer Berlin Heidelberg, 2012.

[33] A.-C. N. Ngomo and K. Lyko. Eagle: Efficient active learning of link specifications using genetic programming. In *The Semantic Web: Research and Applications*, pages 149–163. Springer, 2012.

[34] F. Olsson. A literature survey of active machine learning in the context of natural language processing. 2009.

[35] T. Scheffer, C. Decomain, and S. Wrobel. Active hidden markov models for information extraction. In *Advances in Intelligent Data Analysis*, pages 309–318. Springer, 2001.

[36] B. Settles. Active learning literature survey. *University of Wisconsin, Madison*, 52(55-66):11, 2010.

[37] H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 287–294, New York, NY, USA, 1992. ACM.

[38] D. Spina, M.-H. Peetz, and M. de Rijke. Active learning for entity filtering in microblog streams. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pages 975–978. ACM, 2015.

[39] B. Svingen. Learning regular languages using genetic programming. *Proc. 3-rd Genetic Programming Conference*, pages 374–376, 1998.

ABOUT THE AUTHORS:

Alberto Bartoli received the degree in Electrical Engineering in 1989 cum laude and the PhD degree in Computer Engineering in 1993, both from the University of Pisa, Italy. Since 1998 he is an Associate Professor at the Department of Engineering and Architecture of University of Trieste, Italy, where he is the Director of the Machine Learning Lab. His research interests include machine learning applications, evolutionary computing, and security.

Andrea De Lorenzo received the diploma degree in Computer Engineering from the University of Trieste, Italy in 2006 and the MS degree in Computer Engineering in 2010. He received the PhD degree in Computer Engineering in 2014, endorsed by the University of Trieste. His research interests include evolutionary computing, computer vision, and machine learning applications.

Eric Medvet received the degree in Electronic Engineering in 2004 and the PhD degree in Computer Engineering in 2008, both from the University of Trieste, Italy. He is currently an Assistant Professor in Computer Engineering at the Department of Engineering and Architecture of University of Trieste, Italy. His research interests include web and mobile security, genetic programming, and machine learning applications.

Fabiano Tarlao received the degree in Electronic Engineering from the University of Trieste, Italy in 2010. He is currently working toward the PhD degree from the Department of Engineering and Architecture at University of Trieste, Italy. His research interests are in the areas of web security, genetic programming, and machine learning applications.