Interactive Example-Based Finding of Text Items

Eric Medvet[a,*], Alberto Bartoli[a], Andrea De Lorenzo[a], Fabiano Tarlao[a]

[a]*DIA, University of Trieste, Italy*

Abstract

We consider the problem of identifying within a given document all text items which follow a certain pattern to be specified by a user. In particular, we focus on scenarios in which the task is to be completed very quickly and the user is not able to specify the exact pattern of interest. The key use case corresponds to the interactive exploration of documents in search of snippets that do not fit Boolean, word-based search expressions. We propose an *interactive* framework in which the user provides *examples* of the items he is interested in, the system identifies items similar to those provided by the user and progressively refines the similarity criterion by submitting selected queries to the user, in an *active learning* fashion. The fact that the search is to be executed very quickly places severe requirements on the algorithms that can be used by the system, both for identifying the items and for constructing the queries. We propose and assess experimentally in detail a number of different design options for the components of the learning machinery. The results demonstrate the ability of our approach to achieve effectiveness close to state-of-the-art approaches based on regular expressions, while requiring an execution time which is orders of magnitude shorter.

Keywords: Active learning, Text processing, Pattern

1. Introduction

A common practical problem consists in the identification within a given document of text items which follow a certain pattern to be specified by a user. For example, text editors, pdf viewers, web browsers and alike, usually allow specifying word-based search expressions that may include Boolean operators and a few operand modifiers (e.g., case-insensitive, ignore certain special characters); the user may then navigate in the document by jumping between the items that match the search expression. The search results may be also presented as a list of snippets that are centered around each matching item and that can all be analyzed at a glance, for easier exploration. Although this framework is simple and effective, it is also unable to accommodate several practical needs. It is often the case that the user does not really know what the pattern of interest is, as such a pattern could emerge only after observing the results of several, successively refined candidate patterns. Furthermore, word-based Boolean search expressions are not a good fit for several kinds of searches, for example: names in unfamiliar languages or that admit several spelling variants; model numbers of electronic devices, which often contain optional or country-dependent portions; items that may or may not be abbreviated; noisy documents obtained through optical character recognition (OCR); items that could be broken by an end of line or contain typing errors; items that share a common structure with varying portions. Some of these needs could be addressed by specifying the desired pattern with a formalism more powerful than word-based Boolean search expressions, for example, many text editors (e.g., Google Docs) allow searching for items matching a user-specified regular expression. On the other hand, this approach requires specialized skills, is often difficult to apply in practice and may be effective only for certain kinds of patterns.

In this work we propose an *interactive* framework for finding text items in an unstructured document modeled after the use cases outlined above, i.e., in which the user is not able to specify the exact pattern of interest yet all the relevant items are to be identified very quickly. In our proposal the user provides only *examples* of the items he is interested in; the system identifies items similar to those provided by the user and progressively refines the similarity criterion by submitting selected queries to the user, in an active learning fashion. We focus on the underlying algorithms required for supporting such a framework without insisting on any specific embodiment—i.e., we do not pose any assumptions on whether the user will use the identified items for easier document navigation, for extracting a list of items or surrounding snippets, for correcting mistakes, and alike.

We strive for interactive execution, thus the algorithms for: (i) inferring a general pattern from the available examples; (ii) selecting from the input document the next query to be submitted to the user; (iii) allowing the user to answer the query; and, (iv) actually finding all occurrences

*Corresponding author

Email addresses: `emedvet@units.it` (Eric Medvet), `bartoli.alberto@units.it` (Alberto Bartoli), `andrea.delorenzo@units.it` (Andrea De Lorenzo), `fabiano.tarlao@phd.units.it` (Fabiano Tarlao)

of the current pattern within the document, must all be executed very quickly. We used one minute as time budget for the entire execution, including the time taken by the user to answer the query, which is clearly a very tight requirement placing severe restrictions on the algorithms that can be realistically used. We selected this time budget arbitrarily, assuming that it is indeed representative of the interactive and one-off use cases we are interested in. We do not make any assumption on the nature of the input data, in particular, we do not assume any segmentation in terms of words, lines, sentences or alike that may be exploited by the system—i.e., the input data is merely a flat text string in which items of interest may occur at any point. While this choice widens the generality of our approach, it also magnifies the difficulty of the problem as the number of candidate text items to be analyzed by the system grows quadratically with the length of the document.

We approached the problem as a binary classification problem over substrings (*tokens*) that we obtain by segmenting the document with a *tokenization* heuristics. The heuristics splits the document in tokens based on the characters delimiting each example, hence the segmentation is refined as more and more examples become available. We explored a number of design variants, both alone and in combination, regarding nature of classifier, tokenization heuristics and construction of queries (i.e., active learning scheme). We considered classifiers based on such general techniques as, e.g., Random Forests and Artificial Neural Networks, applied on varying sets of features extracted from each token to be classified; and, classifiers more tailored to the specific application domain based on several string similarity indexes. We considered several variants of the tokenization heuristics for taking into account the surrounding context in which an item of interest is and, concerning the construction of queries, we tailored several active learning schemes to our specific problem, e.g., uncertainty-based, query-by-committee.

We assessed the numerous variants by means of an extensive experimental evaluation on 10 real world text search tasks which have already been used in previous works. In order to make the assessment more relevant for our intended use case, we involved several human subjects and measured the time required for actually answering queries. These data enabled us to fit the number of examples that may be elicited as a function of the time budget available. The results demonstrate the ability of several design variants for finding text items with effectiveness close to state-of-the-art approaches based on regular expressions, while requiring an overall time which is orders of magnitude shorter.

2. Related work

In this section we place our proposal in perspective with existing literature.

A very powerful and widespread tool for formalizing text patterns is *regular expressions*: a regular expression describes the pattern of interest in a compact form and a specialized engine extracts from an input string only the substrings that adhere to the specified pattern. The authoring of a regular expression specifying exactly the pattern of interest is not an easy task, however. It is not surprising, thus, that practitioners and researchers devoted significant efforts in developing methods and systems able to automatically construct a regular expression starting solely from *examples* of the desired extraction behavior. Li et al. (2008) were among the first to show a technique able to cope with data of practical interest (such as URLs and phone numbers): the proposed technique allowed to improve an existing, user-provided regular expression, based on thousands of examples. Later, a team from SAP AG removed the necessity of providing an initial regular expression (Brauer et al., 2011). A different approach for constructing regular expressions has been proposed by Cochran et al. (2015), where a way of exploiting the collaboration of skilled and unskilled users (by means of crowdsourcing) is described in order to obtain both more useful annotated data and better regular expressions. More recently, Bartoli et al. made significant advancements in this area, by increasing the extraction effectiveness and decreasing the amount of examples needed by the system (Bartoli et al., 2014a, 2016e). Indeed, the system developed in these works turned out to be competitive even with skilled human operators (Bartoli et al., 2016b). The same group explored the usage of active learning techniques in order to decrease the amount of effort required by the user for providing examples of the desired extraction behavior (Bartoli et al., 2016a, 2017).

As in all the cited works, we focus on the identification of text items that adhere to a general pattern within an unstructured stream, extraction being just one of the the simplest processing steps that may follow the identification of the item. However, we strive for a radically different trade-off between user effort and extraction quality: unlike all those works, we address the region of the design space where user effort and system execution time are so small to enable interactive, real-time extraction while, at the same time, ensuring good extraction quality. Indeed, our proposal compares favorably to earlier approaches in this respect (we use the same data as in (Bartoli et al., 2016a, 2017)) and none of those approaches is amenable to interactive execution: either the user has to provide all the examples and then wait for several minutes before obtaining the corresponding regular expression (Li et al., 2008; Brauer et al., 2011; Bartoli et al., 2014a, 2016e,b), or the user has to engage an interaction in which the system asks for feedback at intervals that last in the order of the tens of minutes or even more (Cochran et al., 2015; Bartoli et al., 2016a, 2017).

Interesting proposals for automating specific text processing tasks which typically occur in an one-off fashion (Gualtieri, 2011; Bartoli et al., 2015b) include, e.g., search-

and-replace (De Lorenzo et al., 2013), editing (Yessenov et al., 2013), and tabular data extraction (Le & Gulwani, 2014). The latter two tasks have been approached with a programming-by-examples (PBE) methodology, where the input is an incomplete specification of the desired behavior in the form of user-provided examples while the output is a program in some domain-specific language (DSL) (Gulwani, 2016).

We emphasize that our proposal does not include any form of semantic or natural language processing, thus it does not fit application domains where such capabilities are essential. Examples of such domains include extraction from unstructured text of, e.g., named entities (McCallum & Li, 2003; Pennacchiotti & Pantel, 2009; Finkel et al., 2005), facts (Etzioni et al., 2004), relations (Angeli et al., 2014), structured knowledge base (Carlson et al., 2010).

In these contexts there have been many proposals aimed at minimizing the user effort needed to identify and annotate informative examples, e.g., Zhang (2008); Dalvi et al. (2016); Budlong et al. (2013); Bajaj et al. (2015). Some proposals advocated the execution of an initial, fully unsupervised analysis of the data corpus followed by some explicit instructions from the user on how to proceed with the processing of, e.g., logs (Fisher et al., 2008), collections of facts (Etzioni et al., 2005), or relations (Yates et al., 2007).

Our proposal is based on a form of active learning, i.e., the system progressively refines a learned model based on the answer provided by the user to queries selected by the system itself. We compare different active learning schemes and assess them not only on how they affect the extraction effectiveness, but also on their impact on user annotation effort. User effort is usually assessed by assuming that all queries require the same answering effort, but it has been shown that such an assumption may frequently lead to misleading conclusions (Settles et al., 2008). For this reason, we take into account the precise amount of time the user takes to annotate the text by responding to system queries, rather than simply counting the number of queries. From a general point of view, there have been other works concerning active learning in which the annotation (or labeling) costs are not equal among queries: an experimental study involving 4 practical cases (including one related to information extraction) can be find in (Settles et al., 2008), whereas a broader discussion has been done by Settles (2009). In some cases, approaches have been proposed for trying to predict the benefit of a user answer on the learning and relate it to the (possibly estimated) cost of that answer (Donmez & Carbonell, 2008; Haertel et al., 2008; Settles et al., 2008). However, Haertel et al. (2015) and Settles & Craven (2008) conclude that there is no significant improvement when the answer cost is taken into account while constructing the query: Haertel et al. (2015) motivate the finding both theoretically and experimentally and concerns a Return-on-Investment Active Learning (ROI AL) framework, whereas Settles & Craven (2008) support it by experiments on using the ratio between informativeness and cost to drive the choice of the query.

In our proposal the system interactively learns a model for describing strings that are structurally similar to the examples provided by the user. Some of the design variants that we explored are based on quantitative indexes of string similarity proposed earlier in the literature (Cheatham & Hitzler, 2013a; Cohen et al., 2003; Bilenko et al., 2003; Bartoli et al., 2016c). Notions of this kind have been applied in several application domains (Hu et al., 2015; Yu et al., 2016), in particular, they have proven to be a powerful tool for operations of data integration and cleansing (Jiang et al., 2014), ontology alignment (Cheatham & Hitzler, 2013b). Such problems consist, essentially, in finding similar string pairs within given collections of strings. The key difference with our problem is that we strive for interactive execution, which places severe limits on the algorithms that can be used, and that our search is asymmetric in the sense that we need to identify a set of strings within a text sequence (input document) that is potentially large. Furthermore, we dot not assume any structure in the input document hence we cannot exploit any informative signal related to either the content of specific document fields (as in applications of record deduplication (Jin et al., 2003; Culotta & McCallum, 2005; Chandrasekar et al., 2013) and author disambiguation (Chin et al., 2014)) or to the geometric layout of the rendered document (Bartoli et al., 2014b).

3. Problem statement and system architecture

We are concerned with the interactive finding of text items that are similar to those provided by the user. For ease of presentation, we shall refer to the *extraction* of text items from an input document. We remark, though, that we do not assume any specific embodiment of our framework and do not make any assumption on the processing steps that might possibly involve the items found.

A *problem instance* is defined by a string s, a set $X^\star$ of substrings of s, a substring $x_0^m \in X^\star$, and a substring $x_0^u \notin X^\star$—a substring is specified by its content and its position in s. Set $X^\star$ represents the ideal set of extractions which the user expects to obtain from s. Substring x_0^m represents a desired extraction while substring x_0^u represents an undesired extraction. The input of the problem instance consists of (s, x_0^m, x_0^u) while the expected output is $X^\star$: in an actual use case, $X^\star$ is initially unknown to both the user and the system.

We propose a system in which a problem instance is processed according to the following iterative procedure (Figure 1). Let X^m and X^u be respectively the sets of labeled *desired* and *undesired extractions* provided by the user up to a given iteration. At the beginning, $X^m = \{x_0^m\}$ and $X^u = \{x_0^u\}$, then:

1. An extractor E is learned on (s, X^m, X^u) and applied to s. (*Extraction*)

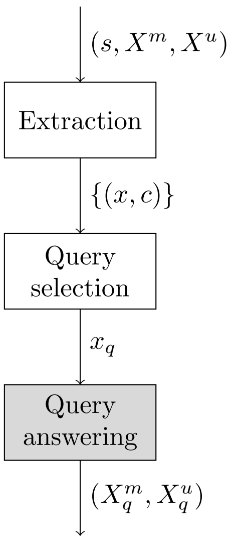


Figure 1: Overview of an iteration cycle of our proposed system, as described in the text. The step of the box with gray background is performed by the user whereas the others are performed by the system.

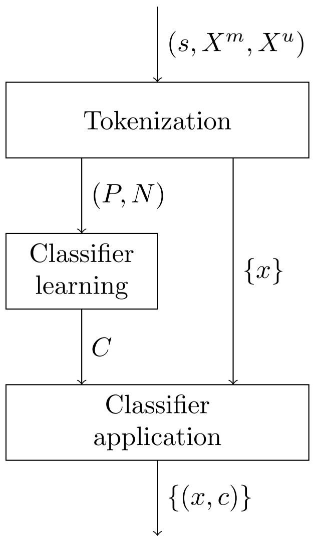


Figure 2: Overview of the extraction step.

2. The outcome of the application of E on s is taken as input by a pool-based active learning algorithm which selects a single substring x_q to be submitted to the user. (*Query selection*)

3. The user inspects the *extraction query* x_q and replies to the system, which updates X^m and X^u accordingly. (*Query answering*)

The three steps are repeated until the user is satisfied by the current extractions or his time budget has been consumed. All the executions of the three steps above concur in consuming the user time budget.

The extraction step and query selection step are described, respectively, in Section 4 and in Section 5. Concerning the query answering step, we assume that each answer belongs to one of these categories: (a) the query x_q exactly corresponds to a desired extraction (i.e., $x_q \in X^\star$); (b) x_q is not a desired extraction and does not overlap any desired extraction; (c) x_q overlaps one or more desired extractions. We call *binary answers* the answers corresponding to cases a and b, whereas we call *edit answers* those of case c. For binary answers, the user may provide the answer by means of a single action (e.g., a click on an "Extract" button or on a "Do not extract" button). For edit answers, more complex user actions are involved since the boundaries of the overlapping desired extractions have to be marked. These behaviors may be embedded in a graphical interface easily.

The answer of the user to an extraction query x_q, thus, consists of a pair (X^m_q, X^u_q) where X^m_q is the (possibly empty) set of all substrings in $X^\star$ which overlaps x_q and X^u_q is the (possibly empty) set of maximal substrings of x_q which do not belong to $X^\star$. Note that for binary answers $|X^m_q| + |X^u_q| = 1$: i.e., exactly one between X^m_q and X^u_q contains x_q while the other is empty. The system consumes the answer by inserting substrings of X^m_q into X^m and substrings of X^u_q into X^u.

4. Extraction

4.1. Extractor architecture

The extraction step of the previous Section consists of three phases (Figure 2):

1. A *tokenization* phase which subdivides s in substrings, i.e., *tokens*; given a problem instance (s, X^m, X^u), this procedure constructs a tuple (P, N), where P is the multiset composed of the tokens in X^m and N is the multiset composed of the tokens in X^u.

2. A *classifier learning* phase in which a binary string classifier C is constructed by using (P, N) as learning data.

3. A *classifier application* phase in which the classifier C constructed at the previous step is applied to the tokens in s; C outputs a numeric *score* c for each input token x: $c < 0$ indicates that x should not be extracted, $c > 0$ indicates that x should be extracted, and $c = 0$ indicates uncertainty about whether to extract x. The larger the absolute value of c, the stronger the confidence in the extraction indication.

The tokenization procedure is necessary because we do not assume any segmentation of the input string s in terms of words, lines, sentences, or alike. In order to be effective in practice, the procedure must be tailored to the specific problem instance. In details, we proceed as follows.

We construct a problem instance-specific set $\mathcal{S}$ of characters acting as token separators. First, we construct the set of characters $\mathcal{S}_0$ including each character immediately preceding or immediately following each substring in X^m.

$$\begin{aligned} s &= \begin{array}{l}\texttt{The␣file␣creation␣date␣is␣7.2.2011,}\\ \texttt{and␣it␣has␣been␣edited␣33␣times␣}\\ \texttt{since␣11-10-2013.␣The␣last␣2␣read␣}\\ \texttt{dates␣are␣24/1/2016␣and␣2/3/2016.}\end{array}\\ X^m &= \{\texttt{7.2.2011}, \texttt{11-10-2013}, \texttt{24/1/2016}\}\\ X^u &= \{\texttt{last␣2}, \texttt{33}\}\end{aligned}$$

(a) Problem instance (s, X^m, X^u).

$$\begin{aligned} &\mathcal{S}_0 = \{␣, \texttt{,}, \texttt{.}\} & \\ &\mathcal{S}_1 = \{␣\} & \Rightarrow |T_1 \cap X^m| = 1\\ \mathcal{S} = \,&\mathcal{S}_2 = \{␣, \texttt{,}\} & \Rightarrow |T_2 \cap X^m| = 2\\ &\mathcal{S}_3 = \{␣, \texttt{,}, \texttt{.}\} & \Rightarrow |T_3 \cap X^m| = 2\end{aligned}$$

(b) Choice of the separators set $\mathcal{S}$.

$$\begin{aligned} P &= \{\texttt{7.2.2011}, \texttt{11-10-2013}, \texttt{24/1/2016}\}\\ N &= \{\texttt{last}, \texttt{2}, \texttt{33}\}\end{aligned}$$

(c) Corresponding classification problem (P, N).

Figure 3: Example of the execution of the tokenization procedure.

Second, we sort $\mathcal{S}_0$ in descending order according to the number of occurrences of each character. Third, we iterate the following steps starting from $i = 1$: (i) we construct the set $\mathcal{S}_i$ of the first i characters of $\mathcal{S}_0$, and (ii) we build the set T_i of tokens obtained by splitting s with the separators in $\mathcal{S}_i$. Finally, we assign $\mathcal{S}$ to the set $\mathcal{S}_i$ for which the number of tokens which are substrings to be extracted is the largest, i.e., $\mathcal{S} := \mathcal{S}_{i^*}$, with $i^* = \arg\max_i |T_i \cap X^m|$—in case of tie, we select the lowest i.

Figure 3 shows an example of the tokenization procedure on a problem instance concerning the extraction of dates: it can be seen that $\mathcal{S}$ is assigned to $\mathcal{S}_2 = \{␣, \texttt{,}\}$ and hence the dot character is not considered as as a separator because, otherwise, the substring `7.2.2011` of X^m would be split.

4.1.1. Improved tokenization

The procedure described above introduces two practically relevant limitations on the extraction.

First, if the token separators cause the desired extractions (i.e., substrings in $X^\star$) to be tokenized, those substrings cannot be extracted regardless of the effectiveness of the string classifier. For example, consider a problem instance in which the user is interested in extracting sequences of capitalized words: $s =$ `Andrew␣met␣his␣friend␣Alice␣Young,␣who␣everybody␣knew␣as␣Alice␣the␣kid` and $X^\star = \{$`Andrew`, `Alice␣Young`, `Alice`$\}$. In this case, the set $\mathcal{S}$ of separators might include (depending on the substrings in X^m and X^u) ␣ and `,`. The substring `Alice␣Young` would not be extracted because it is not even considered by a string classifier, being split in `Alice` and `Young`: note, however, substrings `Alice` and `Young` could be extracted *separately.*

Second, since tokens are analyzed by the string classifier in isolation, the *context* in which they appear cannot be taken into account: hence, the discriminative power of the string classifier may be severely hampered. For example, consider a problem instance in which the user is interested in extracting Twitter user names from tweets, i.e., mentions without the leading `@`: $s =$ `Nice␣research␣paper␣from␣@johnreds␣on␣wild␣#birds␣detection␣and␣classification.␣@units` with the set $X^\star = \{$`johnreds`, `units`$\}$. The string classifier would be learned on strings multisets $P = \{$`johnreds`, `units`$\}$ and $U = \{$`Nice`, `research`, `paper`, `from`, …$\}$: it would hardly be able to build an appropriate and discriminative model of strings in the two multisets, since what actually matters, from the point of view of the syntax, stays around those strings.

In order to overcome these limitations, we experimented with two tokenization variants that we describe below (the extraction step resulting from these variants does not fit the graphical overview in Figure 2; we do not provide additional overviews for the sake of brevity).

Multitoken. The first variant deals with the first limitation. In this variant, right after the building of the separators set $\mathcal{S}$, a numeric value n_tokens is determined as follows: first, each substring x_i in X^m is tokenized using the separators in $\mathcal{S}$ and the resulting number n_i of tokens is saved; then, n_tokens is set to the largest n_i. The building of P and N is not affected. During the extraction phase, instead of applying the string classifier only to each token of s, it is applied also to sequences of up to n_tokens consecutive tokens. For example, considering the problem instance concerning the capitalized words presented before, the string classifier would be applied to strings `Andrew`, `met`, `his`, …, but also to strings `Andrew␣met`, `met␣his`, …, `Alice␣Young`, …, because, in this case, n_tokens would have been set to 2.

Note that for problem instances where desired extractions are not tokenized, this variant does not engage (since $n_\text{tokens} = 1$) and hence neither effectiveness nor time efficiency are affected.

Context-aware. The second variant deals with the context-related limitation. The key idea for this variant is to build three string classifiers, one which operates on the string under analysis, one on a short preceding string, and one on a short succeeding string.

In details, we proceed as follows. In the learning phase, right after building the multisets P and N as described before, we also build the multisets P^before, P^after, N^before, and N^after. For each string x in P, we insert in P^before the string composed of the $n_\text{context} = 5$ characters preceding x in s and we insert in P^after the string composed of the n_context characters succeeding x in s; the same for N, N^before, and N^after. Then, three string classifiers are learned, one on P, N, one on P^before, N^before, and one on P^after, N^after. Since we assume that it is not known, a priori, if in the problem instance the context does matter, we designed this variant to perform a *validation procedure* during the learning phase: the procedure is aimed at figuring out to which degree the three string classifiers concur in discriminating substrings that are to be extracted from substrings that are not to be extracted. The procedure is performed as follows (we consider the case of P, N, the same applies to the other two cases): (i) a number $n_v = 3$ of pairs $P_1, N_1, \dots, P_{n_v}, N_{n_v}$ is built from P, N by discarding each time two different strings $x_{p,i} \in P$ and $x_{n,i} \in N$; (ii) for each pair, a string classifier is learned on the pair P_i, N_i and then applied to the discarded strings $x_{p,i}, x_{n,i}$; (iii) the accuracy of classification a is computed as the number of correct classifications divided by $2n_v$. The procedure results in three accuracies, a, a^{before}, and a^{after}, which capture to which degree the strings in the corresponding multisets are "separable". If $|P| \leq n_v$ or $|N| \leq n_v$, we skip the validation procedure and set $a = a^{\text{before}} = a^{\text{after}} = 1$. In all cases, the three classifiers for the multisets P, N, $P^{\text{before}}, N^{\text{before}}$, and $P^{\text{after}}, N^{\text{after}}$ are eventually learned on the whole multisets.

In the classification phase, the score assigned to an input string x is computed as weighted average of the scores obtained from the three classifiers applied to x, the string x_{before} of n_{context} characters preceding x in s, and the string x_{after} of n_{context} characters succeeding x in s, with the weights corresponding to the accuracies a, a^{before}, and a^{after}. As an optimization, if any accuracy is lower than 0.5, we do not apply the corresponding classifier.

4.2. Classifiers

We designed and evaluated a number of classifiers to be used in the extraction step (Section 4.1), which we describe in the following subsections. We performed some preliminary experimentation on classifiers based on Conditional Random Fields (CRF), but we found this method not adequate to our use case due to its long learning time. Indeed, other previous works also concluded that CRF take long learning times (Li et al., 2008; Bartoli et al., 2016e).

4.2.1. Markov chain classifier

The key idea for this string classifier is to fit two Markov chain models on the strings in P and N and then classify an input string by comparing its consistency with the two models. Indeed, Markov chains have already been used for classification of strings, such as, e.g., by Medvet et al. (2011); Bartoli et al. (2014b) for the classification of text blocks to be extracted from digitally acquired printed documents. In details, we proceed as follows.

We denote by n the order of the Markov chain model—n being a parameter of this string classification method. In the learning phase, two *transition matrices* $\mathcal{M}_P, \mathcal{M}_N \in [0,1]^{|\mathcal{A}|^n \times |\mathcal{A}|}$ are built on P, N respectively—$\mathcal{A}$ being the set of characters appearing in $P \cup N$ plus the character $\triangleright$, whose role is clarified below. Each row of the transition matrix corresponds to one string of n characters and each column corresponds to a single character.

We describe the procedure for building $\mathcal{M}_P$, the same apply to $\mathcal{M}_N$. First, we set each element of $\mathcal{M}_P$ to 0. Second, for each string $x \in P$, we (i) left pad x with $n-1$ occurrences of the character $\triangleright$ and then (ii) we slide a window of n characters on x and increment the elements of $\mathcal{M}_P$ accordingly: for each window $x_{[i,i+n[}$ of x characters from ith (included) to $(i+n)$th (excluded), we increment the element $\mathcal{M}_P^{x_{[i,i+n[}, x_{[i+n]}}$ of $\mathcal{M}_P$ corresponding to row $x_{[i,i+n[}$ and to column $x_{[i+n]}$. E.g., for $x =$ `banana` and $n = 2$, the elements $\mathcal{M}_P^{\triangleright\texttt{b},\texttt{a}}$, $\mathcal{M}_P^{\texttt{ba},\texttt{n}}$, and $\mathcal{M}_P^{\texttt{na},\texttt{n}}$ are incremented by 1 whereas the element $\mathcal{M}_P^{\texttt{an},\texttt{a}}$ is incremented by 2. Third, each row of the matrix is normalized such that (i) the sum of the elements whose values were equal to zero is equal to ϵ and (ii) the sum of all row elements is equal to 1. After the learning, each element of the transition matrix represents the state-transition probability observed on P, i.e., the probability that a given character follows a given string of n characters. The rationale for setting non zero values (by means of ϵ) for transitions which were not observed on P is to smooth the fitted model in order to avoid assigning zero probability to unseen strings. In all our experimentation, we set $\epsilon = 0.05$ and $n = 5$ after exploratory experimentation.

In the classification phase, the score assigned to an input string x is given by $c = \Pr(x; \mathcal{M}_P) - \Pr(x; \mathcal{M}_N)$, where $\Pr(x; \mathcal{M})$ is the probability that x is generated by a Markov chain defined by the transition matrix $\mathcal{M}$—e.g., $\Pr(\texttt{bag}; \mathcal{M}_P) = \mathcal{M}_P^{\triangleright\triangleright\texttt{b},\texttt{a}} \mathcal{M}_P^{\triangleright\texttt{ba},\texttt{g}}$.

We also considered a variant of this method in which, before the actual learning and classification procedures, each string is preprocessed as follows: each lowercase letter is replaced by `a`, each uppercase letter is replaced by `A`, each digit is replaced by `0`, each spacing character is replaced by `␣`, and each other character is replaced by `.`. For example, the `07-Feb-2011` becomes `00.Aaa.0000`. The rationale for this transformation is an attempt to capture the underlying structure of the strings, rather than their specific content.

4.2.2. Feature-based classifier

The key idea for this string classifier is to transform an input string in a vector of numeric features and then to process this vector using well-established classification algorithms. We considered two possible variants for obtaining a numeric vector $\vec{f}$ from a string x, one based on n-grams and one based on building blocks.

In the *n-grams* variant, strings are first preprocessed according to the procedure described at the end of the previous section. Then, $\vec{f}$ is defined in $[0,1]^{\sum_{k=1}^{n} 5^k}$—because the different characters after the preprocessing are 5—and each element of $\vec{f}$ is the ratio between the number of occurrences in x of a substring up to n characters and the length of x. For example, the only non zero elements for $x =$ `00.aaa.0000` (after preprocessing) with $n = 2$ are $f^{\texttt{0}} = \frac{6}{11}$, $f^{\texttt{a}} = \frac{3}{11}$, $f^{\texttt{.}} = \frac{2}{11}$, $f^{\texttt{00}} = \frac{4}{11}$, $f^{\texttt{aa}} = \frac{2}{11}$, and $f^{\texttt{0.}} = f^{\texttt{.a}} = f^{\texttt{a.}} = f^{\texttt{.0}} = \frac{1}{11}$. In all our experimentation, we set $n = 3$, hence $\vec{f} \in [0,1]^{155}$.

In the *building blocks* variant, $\vec{f}$ is defined in $\mathbb{N}^{120}$ and each element of $\vec{f}$ is the number of matches of a given regular expression of a set $\mathcal{R}$ within the string x. The set $\mathcal{R}$ of regular expressions is built by combining a seed set $\mathcal{R}_0$ of the following 10 regular expressions: `[A-Z]`, `[a-z]`, `[0-9]`, `\w`, `[^\w]`, `\s`, `[^\w\s]`, `[-@&#:;]`, `[\[\]()]`, and `[*+-/\\]`. For each regular expression $r \in \mathcal{R}_0$, three regular expressions r`++`, `^`r, and r`$` are present[1] in $\mathcal{R}$; moreover, for each pair $r_1, r_2 \in \mathcal{R}_0 \times \mathcal{R}_0$ such that $r_1 \neq r_2$, the regular expression $r_1 r_2$ is present in $\mathcal{R}$. It follows that $|\mathcal{R}| = 3 \cdot 10 + 10^2 - 10 = 120$.

In the classification phase, the score assigned to an input string x is given by the difference of the posteriori probabilities assigned by the classifier to the feature vector $\vec{f}$ obtained from x for the two classes, i.e., $c = \Pr(\vec{f}; P) - \Pr(\vec{f}; N) = 2\Pr(\vec{f}; P) - 1$.

We experimented with four classifications algorithms: Random Forest (Breiman, 2001), Naive Bayes (McCallum et al., 1998), AdaBoost (Freund & Schapire, 1995), and Artificial Neural Network—we used only Random Forest with n-grams and all the 4 algorithms with building blocks. We chose these algorithms after preliminary experimentation, which also allowed us to set the values of the most significant parameters for each algorithm. For Random Forest and AdaBoost, we set the number of trees to 100. For Naive Bayes, we used the multinomial model, which we found to be more effective, despite the findings of McCallum et al. (1998). For Artificial Neural Networks, we used a network with 1 intermediate layer of 20 nodes trained using the cross entropy error function.

4.2.3. Similarity-based classifier

The key idea for this string classifier is to classify an input string based on its average similarity to strings in P and N. We considered several variants for the actual string similarity index m being used: based on exploratory experimentation and previous studies (Cheatham & Hitzler, 2013a; Cohen et al., 2003; Bilenko et al., 2003; Bartoli et al., 2016c), we actually implemented three extractors using Jaccard, Jaro-Winkler, and Needleman-Wunsch similarity indexes. In particular, we chose these metrics because they represent, as shown by Bartoli et al. (2016c), an interesting trade-off between computational effort and effectiveness in evaluating as (dis)similar pairs of string which have the same (or different) extraction outcome.

The Jaccard similarity index between two strings is computed by considering each string as a set of characters or bigrams (as in our case) and is the ratio between the sizes of the intersection and the union of the two sets. The Jaro-Winkler index is a modified Jaro index in which similarity grows for strings that share a common prefix; in turn, Jaro similarity index takes into account matching characters, i.e., characters appearing in both strings at an offset smaller than a certain quantity, and transpositions, i.e., number of matching characters appearing in a different order in the two strings. Finally, the Needleman-Wunsch (Needleman & Wunsch, 1970) is a form of edit distance which assigns different costs to edit operations: we adjusted the output in order to use it as a similarity index rather than a distance, i.e., lower values mean dissimilar strings.

In the learning phase, we determine a threshold τ as follows. We denote with $m(x_1, x_2)$ the similarity among strings x_1 and x_2: we assume that $m(x_1, x_2) = 1$ if $x_1 = x_2$ and the closer $m(x_1, x_2)$ to zero, the less similar the two strings. First, for each string x in $P \cup N$, we compute a value $\Delta_m(x)$ defined as the difference between the average similarity of x to strings in P and to strings in N, i.e., $\Delta_m(x) = \frac{1}{|P|}\sum_{x' \in P} m(x, x') - \frac{1}{|N|}\sum_{x' \in N} m(x, x')$. Then, we choose the value for τ which maximizes the sum of the number of strings in P for which $\Delta_m(x) > \tau$ and the number of strings in N for which $\Delta_m(x) < \tau$. Note that, in an ideal case, $\Delta_m(x)$ is close to 1 for strings in P, $\Delta_m(x)$ is close to -1 for strings in N, and τ is close to 0.

In the classification phase, the score assigned to an input string x is given by $c = \Delta_m(x) - \tau$. Note that, differently from Markov chain and feature-based string classifier, the computational effort in the classification phase depends on the size of the multisets P and N.

5. Active learning schemes

Several schemes for pool-based active learning have been proposed (Settles, 2009). All such schemes try to capture the concept of *informativeness* of a candidate instance with respect to the learning process: they differ in how the informativeness is computed and in how tightly they are coupled with the underlying learning algorithm.

The most common active learning schemes are: uncertainty sampling, query-by-committee, expected model change, variance reduction, estimated error reduction. For many of them, variants based on density weighting have been proposed as well: e.g, query-by-committee with density estimation (McCallum & Nigam, 1998), uncertainty sampling with pre-computed density on text classification segmentation (Settles & Craven, 2008). Among all these schemes, based on previous studies and on the nature of our problem in terms of data and goals, we decided to consider only the first two (uncertainty sampling and query-by-committee), together with two other custom schemes which we designed for our case. We think that the others do not fit our scenario, mainly because they are computationally expensive.

In particular, variance reduction is intrinsically computationally expensive and, most importantly, needs to be tailored to the specific underlying classifier (the extractor, in our case)—e.g., it has been adapted to Conditional Random Fields by Settles & Craven (2008).

[1]The regular expressions `^`r and r`$` match a substring at the beginning or end, respectively, of the string which itself matches r.

Estimated error reduction requires the re-training of the underlying classifier for each possible answer to each candidate query and irrespective of the actual answer that will be given by the user, which is clearly expensive. Concerning expected model change, it is not straightforward to be generalized with respect to the underlying classifier: however, we were inspired by this scheme while designing one of the two active learning approaches which we propose.

In the next sections, we describe how we adapted uncertainty sampling and query-by-committee to our case and we present the two schemes we designed ad hoc. Finally, we discuss about the density weighting variant and how we applied it.

5.1. Uncertainty sampling

The key idea for this active learning scheme is to select, among the pool of available instances, the one for which the current model is most uncertain.

This scheme application to our case is straightforward, since the score c assigned by the extractor E to a substring x can be used to estimate the extractor uncertainty: the closer c to 0, the larger the uncertainty. Hence, in this scheme, the selected query x_q is the one for which $\operatorname{abs}(c)$ has the lowest value. If that substring overlaps a substring in $X^m \cup X^u$, i.e., it is already (partially) annotated, then the second most uncertain substring is selected, and so on.

5.2. Balancer

This active learning scheme bases on the assumption that a balanced learning set may result in a more effective classification: such assumption has been confirmed also for classification scenarios similar to the one here considered (Bartoli et al., 2014a). Hence, the queried substring is the one which, most likely, will make the current learning set more balanced: i.e., if $|X^m| > |X^u|$, the query is the one for which the system has largest confidence on the fact that it should not be extracted; otherwise, if $|X^m| < |X^u|$, the query is the one for which the system has largest confidence on the fact that it should be extracted.

In details, we proceed as follows. When a substring has to be selected from the pool, the sizes of the current learning sets $|X^m|$ and $|X^u|$ are considered. If $|X^m| > |X^u|$, the substring with the largest negative score c is selected as x_q—we recall that a large negative score means that the extractor is strongly confident that the substring should not be extracted: hence, it is the one which most likely will be marked by the user as not to be extracted. Similarly, if $|X^m| < |X^u|$, the substring with the largest positive score is selected. Otherwise, if the learning set is already balanced, i.e., if $|X^m| = |X^u|$, then the most uncertain substring is selected as query x_q, as for uncertainty sampling. In all cases, if the selected substring overlaps a substring in $X^m \cup X^u$, then the next most appropriate substring in the pool is selected according to the specific criterion.

5.3. Least similar instance

The key idea for this active learning scheme is to select a substring which is strongly different from the ones already available, i.e., those in $X^m \cup X^u$. The rationale for this choice is to provide the extractor with largely different examples and hence avoid "overfitting" the extraction behaviour on a specific subset of desired extractions. As an aside, choosing the most different substrings may stimulate large changes in the extractor model—in this sense, this method resembles the expected model change general active learning scheme.

In details, we proceed as follows. Let m be the string similarity metric being used. For each substring x in the pool, we compute its average similarity $\overline{m}(x)$:

$$\overline{m}(x) = \frac{1}{|X|} \sum_{x' \in X} m(x, x') \tag{1}$$

where X is the set X^m or X^u depending on whether the score c assigned to x is ≥ 0 or < 0, respectively. Then, we select as the query x_q the substring x with the lowest average similarity $\overline{m}(x)$. In other words, if the substring appears to be a desired extraction, we aim at minimizing its similarity with labeled desired extractions; otherwise, we aim at minimizing its similarity with labeled undesired extractions. As for the other schemes, if the selected substring overlaps a substring in $X^m \cup X^u$, then the next most appropriate substring in the pool is selected.

We considered the same string similarity metrics we used to build the extractor described in Section 4.2.3, for the same motivation. However, since we experimentally found that Jaccard outperformed the other two, we used only the former for the remaining evaluations.

5.4. Density weighting variant

The key idea for this variant is that the more representative a candidate query, the better. The representativeness of an instance may be measured as the average similarity with other instances in the pool, as suggested by Settles & Craven (2008) who calls it information density: if an instance lies in a dense region of the instance space, it is deemed representative. The degree of representativeness of a candidate query should then be combined with its informativeness.

In details, we proceed as follows. Let m be the string similarity metric being used. For each substring x in the pool, we compute its informativeness $i(x)$ according to one of the previous active learning schemes (e.g., $-\operatorname{abs}(c)$ for the uncertainty sampling). Then we compute its average similarity $\overline{m}(x)$ with the other substrings in the pool (see Section 5.3). In order to reduce the computational burden, we actually consider only the most informative 25 substrings in the pool (i.e., those with the greatest $i(x)$), which leads to $\frac{1}{2}25^2$ similarity computations. Finally, we compute a score for the substring as $i(x)\overline{m}(x)$ and select as query x_q the substring in the (reduced) pool with the largest score. If the selected substring overlaps a substring in $X^m \cup X^u$, then the next substring in the reduced pool is selected.

We considered this variant in combination with uncertainty sampling and balancer schemes and used the Jaccard metric. We did not combine the density weighting variant with the least similar substring scheme because they pursue conflicting goals (minimizing and maximizing similarity, respectively).

5.5. Query-by-committee

Query-by-committee consists in maintaining a set of alternative hypotheses (the committee) which are all based on the same learning data. The instance to be selected from the pool is the one for which the hypotheses most disagree.

Two key design choices have to be done for applying query-by-committee: (a) the composition of the committee and (b) the measure of disagreement. In our case, we use a committee $\mathcal{Q}$ composed of 7 extractors: Markov chain; AdaBoost with context-aware optimization; AdaBoost with multitoken and context aware optimizations; Artificial Neural Networks; similarity-based with Jaccar; similarity-based with Jaccard and context-aware optimization; similarity-based with Jaccard, multitoken and context-aware optimizations. The key idea for this choice was to select a set of extractors being, at the same time, effective (see Section 6.4) and based on different working principles. Concerning the measure of disagreement, we set the disagreement $d(x)$ of the committee on a substring x of s as the absolute difference between the number of extractors extracting x and half of the committee size, inverted in sign:

$$d(x) = -\operatorname{abs}\left(\left|\left\{E \in \mathcal{Q} : (x,c) \in E(s) \land c > 0\right\}\right| - \frac{|\mathcal{Q}|}{2}\right) \tag{2}$$

If the substring which maximizes $d(x)$ overlaps a substring in $X^m \cup X^u$, then the second substring with greatest disagreement is selected, and so on.

In the extraction phase, a set $\{(x,c)\}$ of substrings has to be extracted. When working according to query-by-committee, our system uses many extractors: we use only one of them to actually produce the set $\{(x,c)\}$. The key idea is to select the extractor which, on the current learning sets X^m and X^u, has the greatest discrimination ability between substrings known to be extracted and substrings known not to be extracted. We proceed as follows. First, for each extractor $E \in \mathcal{Q}$, we obtain the set C^m of desired extractions scores and the set C^u of undesired extractions scores: starting from $C^m = C^u = \emptyset$, for each $(x,c) \in E(s)$, we add c to C^m if $x \in X^m$, otherwise we add c to C^u—ideally, all scores in C^m should be positive and all scores in C^u should be negative. Second, we compute the *overlapness* $o(C^m, C^u)$ of the sets C^m and C^u, where

Tokenization	
P	w/o optimizations
M	Multitoken
C	Context-aware
B	Multitoken and context-aware
String classifier	
MC	Markov chain
pMC	Markov chain w/ preprocessing
AB	AdaBoost on building blocks
NN	Art. neural net. on building blocks
NB	Naive Bayes on building blocks
bRF	Random Forest on building blocks
nRF	Random Forest on n-grams
jS	Similarity-based w/ Jaccard
jwS	Similarity-based w/ Jaro-Winkler
nwS	Similarity-based w/ Needleman-Wunsch
Active learning scheme	
R	Random query
U	Uncertainty sampling
QbC	Query-by-committee
B	Balancer
L	Least similar instance
dU	Density-weighting uncertainty sampling
dB	Density-weighting balancer

Table 1: A summary of the abbreviations of the names for the considered variants of the three key components of our system. The name of a specific variant of the full system is given by the concatenation of the component variant names: e.g., dU/bRF-B is Random Forest on building blocks with both tokenization optimizations and the density-weighted uncertainty sampling scheme; QbC/* is the committee of 7 extractors described in Section 5.5.

the $o(A,B)$ for numeric sets A and B is defined as:

$$\begin{aligned} o(A,B) = & \frac{|\{a \in A : a \geq \min_{b \in B} b\}|}{|A|+|B|} \\ & + \frac{|\{b \in B : b \leq \max_{a \in A} a\}|}{|A|+|B|} \end{aligned} \tag{3}$$

Finally, we select as the actual extractor the one for which the overlapness is the lowest.

In other words, we select the extractor which proves to have, on the learning sets X^m and X^u, the greatest separation ability according to scores assigned to substrings known to be or not to be extracted. This measure for discrimination ability has indeed already been used in the context of text extraction as the main component of fitness function for a string similarity metric synthesis method based on Grammatical Evolution (Bartoli et al., 2016d).

For the sake of presentation clarity, in the next sections we abbreviate the names of the variants of our system (resulting by combining different options for the tokenization, classification, and active learning scheme) as shown in Table 1.

6. Experimental evaluation

6.1. Overview

In this work we are concerned with the identification of text items which follow a certain pattern to be specified by the user by means of examples; all the relevant items have to be identified very quickly, within a time budget that we assume to be bounded within a few minutes. We chose this bound based on the interactive use cases that we believe are most relevant, as outlined in the introduction. Solutions to the problem may be assessed in terms of accuracy (i.e., whether the identified items indeed follow the pattern and no relevant items are missed) and in terms of the time required for identifying all the relevant items.

It is important to emphasize that these performance indexes, defined precisely in the next sections, complement each other and assessing a solution based on only one of them would be misleading. For example, an approach with very high accuracy that takes hours to complete would be unsuitable. On the contrary, an approach that is extremely fast but delivers very low accuracy would be unsuitable as well.

Since the time for identifying all the relevant items is a crucial issue, we quantified this index very carefully and took all the following components into account: (1) the time it takes to the user for selecting and annotating the initial examples; (2) the time it takes to the system for selecting queries to be submitted to the user; (3) the time it takes to the user for answering queries; and, (4) the time it takes to the system for identifying all the items that follow the currently specified pattern. We executed a number of experiments for assessing all our design variants in terms of accuracy and overall execution time, in the sense described above.

In order to gain further insights into our results, we analyzed also the behavior of an earlier proposal developed by our research group for constructing a pattern by means of examples (Bartoli et al., 2016b,e). The cited work describes a tool based on a *passive learning* framework, in which the user is required to analyze the full input text and specify all the examples to be input to the system at once, without any interaction with the system. This passive learning tool delivers state-of-the-art accuracy, but has been designed with different requirements and, in particular, is unfit for scenarios in which the results must be available within a few minutes. As we shall see, the approach proposed in this work delivers results with a latency that is order or magnitudes smaller than that of the passive learning tool, with accuracy that is smaller but roughly comparable. In other words, our proposal complements the tool previously proposed by Bartoli et al. (2016b,e) and provides a practical solution for scenarios that call for a different trade off between accuracy and execution time.

6.2. User annotation time

As discussed in the previous section, we devoted special effort in modelling the overall execution time accurately. Concerning the *annotation time* t_a spent by the user, we followed the same procedure of Bartoli et al. (2017) and proceeded as follows.

We observed in preliminary experimentation that the annotation time for a given query strongly depends on the type of answer, i.e., binary answer vs. edit answer (Section 3). We also observed that the time spent for annotating a query constructed by the system tends to be very different from the time required for fully annotating a long piece of text, as in passive learning (Bartoli et al., 2016e). We hence constructed three different models for the annotation time t_a and we fitted these models on real observations, as follows.

We involved a set of 10 users with varying skills and asked them to annotate a broad variety of datasets on a webapp that we developed for this purpose. We recorded user actions in two different scenarios: (i) when answering queries constructed by a mockup of our system, in order to emulate interaction with a system based on active learning; and, (ii) when fully annotating a dataset, in order to emulate the annotation required by a passive learning tool. In the former case, each user was asked to answer a number (≈ 20) of queries whose expected answers were roughly equally distributed among positive binary answers, negative binary answers, and edit answers. The web application was instrumented for taking note, for each query, of the time the user took to answer, the answer length, and the number of desired extractions in the query. Based on the collected data, we fitted several models for the annotation time and chose those that proved to be more accurate. The resulting models for t_a (expressed in seconds) are the following:

$$t_a^{\text{binary}} = 0.02\ell(X_q^m \cup X_q^u) + 2.0 \tag{4}$$

$$t_a^{\text{edit}} = 3.4|X_q^m| + 0.01\ell(X_q^m \cup X_q^u) + 3.1 \tag{5}$$

$$t_a^{\text{passive}} = 3.4|X^m| + 0.003\ell(X^m \cup X^u) \tag{6}$$

where $\ell(X_q^m \cup X_q^u)$ is the overall length of the answer, $|X_q^m|$ is the number of desired extractions in the answer, $|X^m|$ is the number of desired extractions in the fully annotated dataset, and $\ell(X^m \cup X^u)$ is the overall length of the fully annotated dataset. These figures correspond, in active learning, to an annotation time of ≈ 2.5 s for binary queries and ≈ 7 s for edit queries, assuming that queries lenght is in the order of few tens of characters.

6.3. Data

We considered 10 datasets, each representing a different use case, i.e., each consisting in different text items of interest. All the datasets have already been used in previous studies[2]. Some of those datasets were obtained from

[2]All the datasets developed by our group and that do not contain personally identifiable information are available on our web

earlier publications not published by our research group. We briefly describe them here.

- *BibTex-Author.* A set of bibtex entries from which individual authors names should be extracted. Used by Bartoli et al. (2016e).
- *Bills-Date.* A set of US Congress bills from which dates in 9 different formats should be extracted. Used by Bartoli et al. (2016e, 2015a).
- *Email-Phone.* A set of email excerpts from which phone numbers should be extracted. Used by Bartoli et al. (2016e,a, 2014a); Brauer et al. (2011); Li et al. (2008).
- *Headers-ForToEmail.* A set of email headers from which email addresses occurring right after the strings `for:` or `to:` (possibly capitalized) should be extracted. Used by Bartoli et al. (2016e, 2014a).
- *HTML-href.* A set of HTML documents from which `href` attributes (both name and value) should be extracted. Used by Cetinkaya (2007); Bartoli et al. (2016e,a, 2014a).
- *Log-IP.* A set of log entries of a firewall software from which IP addresses should be extracted. Used by Bartoli et al. (2016e,a, 2014a).
- *Twitter-Hashtag+Citation.* A set of Twitter posts from which hashtags (e.g., `#machinelearning`) and citations (e.g., `@MaleLabTs`) should be extracted. Used by Bartoli et al. (2016e,a, 2014a).
- *Twitter-URL.* A set of Twitter posts from which URLs should be extracted. Used by Bartoli et al. (2016e,a, 2014a).
- *Twitter-Usernames.* A set of Twitter posts from which user names in citations (e.g., just `MaleLabTs` in the substring `@MaleLabTs`) should be extracted. Used by Bartoli et al. (2016e).
- *Web-URL.* A set of web pages (in HTML) from which URLs should be extracted. Used by Bartoli et al. (2016e,a, 2014a); Li et al. (2008).

Table 2 shows salient information about the datasets. It shows the average length $\overline{\ell(x^\star)}$ of a desired extraction, the average length $\ell(s)$ of a text portion including 100 desired extractions, the ratio $\rho^\star$ between the number of characters to be extracted and all characters, and the average time a user would take to fully annotate a text portion including 100 desired extractions, according to the model of Eq. 6. It can be noted that there are significant differences (0.02–0.24) among the datasets concerning $\rho^\star$: in other words, desired extractions are more or less *sparse*. For instance, 100 desired extraction takes ≈ 5500 characters for Log-MAC+IP and $\approx 136\,000$ characters for Web-URL.

site: `http://machinelearning.inginf.units.it/data-and-tools/` `annotated-strings-for-learning-text-extractors`.

Extractor	$\overline{\ell(x^\star)}$	ρ	$\ell(s)$	t_a^{passive}
BibTex-Auth.	15.5	0.17	9185	368
Bills-Date	10.6	0.02	50 693	492
Email-Phone	13.2	0.04	35 140	445
Headers-learn.	23.5	0.03	76 924	571
HTML-href	62.9	0.09	73 250	560
Log-IP	12.9	0.24	5462	356
Twit.-Hash.+Cit.	11.2	0.11	10 251	371
Twit.-URL	19.9	0.07	28 490	425
Twit.-Usern.	11.0	0.08	13 283	380
Web-URL	52.2	0.04	135 412	746

Table 2: Salient information about the datasets. Annotation time t_a is expressed in seconds.

6.4. Extraction assessment

6.4.1. Procedure

We performed a first experimental campaign aimed at assessing effectiveness and time efficiency of each extractor *without* active learning. That is, we constructed a set of desired extractions, a set of undesired extractions and then we synthesized and applied each extractor only once (i.e., we executed only step 1 of Section 3). We considered 29 extractors, resulting from a number of combinations between tokenization procedures and classifiers. This campaign allowed us to reduce the number or extractors to consider and thus obtain a manageable number of design alternatives for the full system, i.e., with active learning. The second campaign aimed at assessing the full system is described in the next section.

We proceeded as follows. For each dataset, we built a number of problem instances by selecting a random substring s of the dataset text such that the number of desired extractions in s was exactly $|X^\star| = 100$; then we annotated an initial portion of s such that the time taken for the (simulated) annotation was t_a, according to the model of Eq. 6 for t_a^{passive}. We repeated this procedure for $t_a \in \{30\,\text{s}, 60\,\text{s}, 90\,\text{s}, 120\,\text{s}\}$ and, for each t_a value, 30 times by varying the substring s: we hence built 120 problem instances $(s, X^m, X^u, X^\star)$ for each dataset.

Then, for each extractor, we learned the extractor on (s, X^m, X^u) and then applied it to s. Finally, we measured the extractor effectiveness in extracting all and only the substrings in $X^\star$. The effectiveness has been computed as F-measure, which is the harmonic mean of precision $\frac{|X \cap X^\star|}{|X|}$ and recall $\frac{|X \cap X^\star|}{|X^\star|}$, where X is the set of substrings extracted by the learned extractor—since F-measure and precision cannot be computed if the extractor does not extract any substring, we set F-measure and precision to zero in that case. We also measured the learning time t_l, i.e., the time the extractor took to be learned on (s, X^m, X^u), and the extraction time t_e, i.e., the time the extractor took to analyze s and extract the substrings in X. We recall that both the learning and extraction times are worth to be investigated, because the iterative procedure described in Section 3 involves, for each iteration, both the steps. All the experiments have been carried out on a workstation equipped with 8 GB and a Intel Core2 Quad CPU 2.5 GHz.

In order to place our results in perspective, we also executed the state-of-the-art passive learning tool (Bartoli et al., 2016e) on the same datasets with a similar procedure: we considered a single substring s of the dataset (instead of 30) built with $t_a = 60$ s and we ran the cited tool 5 times on the instance constructed from s—the latter repetition due to the stochastic nature of the tool proposed by Bartoli et al. (2016e), which is based on Genetic Programming.

6.4.2. Results and discussion: overview

The results of the first experimental campaign are shown in Table 3, Table 4, Table 5, and Figure 5. In particular, Table 3 shows the effectiveness (F-measure) and efficiency (t_l, t_e, and $t_l + t_e$) for each extractor, averaged across datasets and repetitions, obtained for $t_a = 60$ s, i.e., when the user spends one minute for annotating the data (in passive learning mode). Several interesting observations can be done.

First, six extractors obtain an average F-measure larger or equal to 0.60—the best extractor (AB-B) scoring 0.66—while requiring a short time for learning and extracting (from ≈ 170 ms to ≈ 2050 ms for $t_l + t_e$). This result fits our scenario presented in Section 3. To place these figures in a perspective, we also show the F-measure obtained by a state-of-the-art method for syntax-based entity extraction (Bartoli et al., 2016e) (last row of Table 3). The cited method obtains an average F-measure of 0.82, taking a learning time which is 1000 times longer than the one taken by AB-B—half-an-hour against two seconds.

Second, there is a noticeable variation in both effectiveness (F-measure) and time efficiency (t_l and t_e) among extractors: for $t_a = 60$ s, F-measure spans between 0.17 to 0.66 (nRF and AB-B, respectively), t_l spans between 6 ms and 1942 ms (MC and AB-B, respectively), and t_e spans between 20 ms and 1431 ms (NB and jS-B, respectively). Moreover, it can be seen that (i) there exists a trade-off between the effectiveness and time efficiency and (ii) some methods define a Pareto frontier in the space of the two goals. These findings are further supported by Figure 4, which compares the extractors by plotting effectiveness (F-measure) vs. time efficiency ($t_l + t_e$). More in detail, results of Table 3 and Figure 4 suggest that Markov chain extractors are (in general) faster, yet they performs poorly in terms of F-measure. On the other hand, feature-based and similarity-based extractors obtain very good F-measure figures, but take longer to be learned or to be applied, respectively, in particular in the context-aware (-C) variant—mainly due to the validation procedure (see Section 4.1.1).

Extr.	Fm	Prec.	Rec.	t_l	t_e	$t_l + t_e$
MC	0.2 ±0.03	0.15	0.64	6	53	59
pMC	0.31±0.04	0.24	0.67	8	63	71
pMC-C	0.43±0.09	0.38	0.62	19	122	141
pMC-M	0.32±0.05	0.25	0.75	8	263	271
pMC-B	0.47±0.1	0.41	0.7	19	305	324
AB	0.61±0.08	0.74	0.58	130	41	170
AB-C	0.65±0.1	0.81	0.61	1940	59	1999
AB-M	0.59±0.1	0.62	0.6	130	78	208
AB-B	0.66±0.11	0.71	0.66	1942	106	2048
NN	0.52±0.08	0.81	0.49	291	54	345
NB	0.4 ±0.05	0.32	0.65	16	20	36
bRF	0.58±0.1	0.69	0.59	70	86	157
bRF-C	0.62±0.12	0.78	0.6	1025	121	1145
bRF-M	0.57±0.11	0.58	0.6	71	156	227
bRF-B	0.63±0.13	0.69	0.64	1024	214	1238
nRF	0.17±0.01	0.99	0.09	1518	854	2372
jS	0.52±0.07	0.71	0.5	24	173	196
jS-C	0.56±0.09	0.66	0.55	92	333	425
jS-M	0.58±0.08	0.71	0.55	23	1236	1259
jS-B	0.64±0.11	0.73	0.62	15	1431	1446
jwS	0.47±0.09	0.44	0.57	10	163	173
nwS	0.51±0.08	0.53	0.61	27	166	192
Base.	0.82			2.01E6	50	2.01E6

Table 3: F-measure, learning time t_l, extraction time t_e, and overall time $t_l + t_e$ obtained for $t_a = 60$ s with each extractor. Values are averaged across repetitions and datasets: for F-measure, the average across datasets of the standard deviation across repetitions on the same dataset is also shown. Times are expressed in ms. Base stands for the baseline by Bartoli et al. (2016e).

Third, the method variants proposed to cope with the limitations of tokenization (i.e., multitoken and context-aware—see Section 4.1.1) appear to work. We present here the results obtained by combining them with a limited set of extractors, but we verified that the findings are more general. In many cases, the optimized extractors obtain better F-measure than the corresponding unoptimized extractors: in particular, for pMC and jS, both -M and -C outperform the unoptimized extractors, and -B outperform -M, -C and unoptimized. On the other hand, these optimizations impact negatively on both the learning time t_l (up to 15 times longer) and the extraction time t_e (up to 8 times longer).

Among the 22 extractor considered in Table 3, we chose a subset of 7 which we thought were representative of different ways of facing the trade-off between effectiveness and efficiency. On this subset, we conducted further analysis. Table 4 shows the difference of average F-measure, t_l and t_e among pair of extractors of this subset. The absolute value of the difference is shown along with a graphical indication of the statistical significance (in terms of the p-value) according to the Wilcoxon signed rank test, which we performed considering the 300 repetitions which we

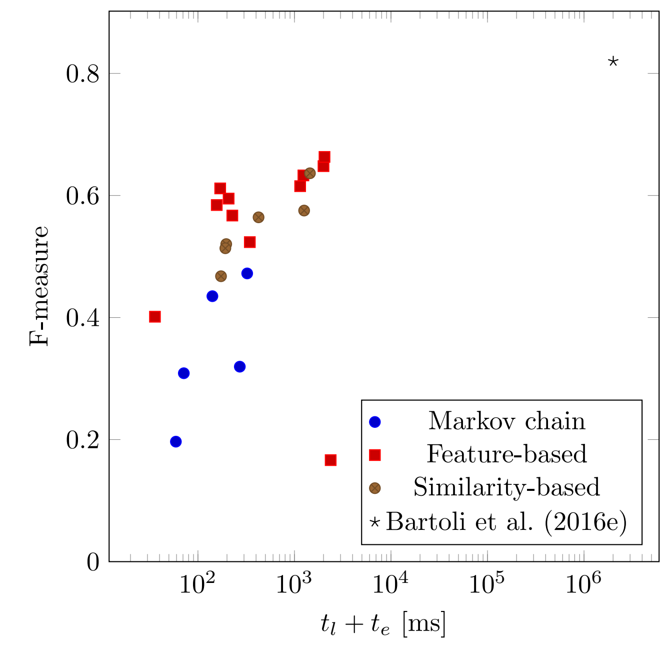


Figure 4: Effectiveness (F-measure) vs. time efficiency ($t_l + t_e$) of the extractrors for $t_a = 60\,\text{s}$. Values are averaged across repetitions and datasets.

performed for each extractor of the pair (with $t_a = 60\,\text{s}$). It can be seen that most of the conclusions drawn are confirmed, with the notable exceptions concerning the differences in F-measure between AB-B, jS-B and AB-C, jS-B which are not significant.

6.4.3. Impact of the learning data size

The impact of the amount of learning data on extractors effectiveness and efficiency can be appreciated in Figure 5. The figure plots F-measure, t_l, and t_e against t_a, one curve for extractor (in a subset of 7 relevant extractors) and one point for the baseline given by Bartoli et al. (2016e). It can be seen that t_a does impact on the F-measures for all the extractors: the larger t_a, the better the F-measure. The largest improvement is obtained when going from 30 s to 60 s. This finding confirms that the proposed extractors are appropriate for the considered scenario in which the user wants to spend short time for annotating the data, that is, when "little" information is available for learning. In this respects, it is worth to note that in our experiments the chosen values for t_a, i.e., 30 s, 60 s, 90 s, and 120 s, corresponded to a number $|X^m|$ of annotated desired extractions available for learning of 5, 9, 14, and 18, respectively, and to an overall number $\ell(X^m \cup X^u)$ of annotated characters of 1730, 3547, 5023, and 7192, respectively. Concerning times, it can be noticed that t_a has different impacts depending on the extractor family. For Markov chain and feature-based extractors, t_l grows roughly linearly with t_a, whereas t_e appears to be weakly affected by t_a. For similarity-based extractors, t_l grows quadratically with t_a, whereas t_e grows roughly linearly with t_a: this finding is consistent with the way

	Extr.	pMC-B	AB-C	AB-B	NN	jS	jS-M
F-measure	AB-C	0.18‡					
	AB-B	0.19‡	0.02*				
	NN	0.05†	-0.12‡	-0.14‡			
	jS	0.05*	-0.13‡	-0.14‡	-0.00		
	jS-M	0.10‡	-0.07*	-0.09‡	0.05†	0.05‡	
	jS-B	0.16‡	-0.01	-0.03	0.11‡	0.12‡	0.06‡
t_l	AB-C	1921‡					
	AB-B	1923‡	1				
	NN	272‡	-1649‡	-1651‡			
	jS	5‡	-1917‡	-1918‡	-267‡		
	jS-M	4‡	-1917‡	-1918‡	-268‡	0·	
	jS-B	-4†	-1925‡	-1926‡	-275‡	-8‡	-8‡
t_e	AB-C	-246‡					
	AB-B	-198‡	48‡				
	NN	-251‡	-5	-52‡			
	jS	-132	114‡	66‡	119‡		
	jS-M	931‡	1177‡	1130‡	1182‡	1063‡	
	jS-B	1126‡	1372‡	1325‡	1377‡	1258‡	195‡

Table 4: Differences of average F-measure, t_l, and t_e between pair of 7 selected extractors, obtained for $t_a = 60\,\text{s}$. For each pair, the statistical significance is shown: ·: $p < 0.1$, *: $p < 0.05$, †: $p < 0.01$, ‡: $p < 0.001$, $p \geq 0.1$ without any subscript.

similarity-based extractors work during the classification phase, when each token is compared against all strings available for learning.

6.4.4. Differences among datasets

Concerning the effectiveness on each dataset, results of Table 5 show that for 3 on 10 datasets the best F-measure is larger than 0.90 and for 8 is larger than 0.60. For BibTex-Author and Bills-Date the best F-measure is rather low (0.52 and 0.45, respectively). Looking at the raw data, we observed that in the former dataset there is not a sharp distinction between the content of substrings to be extracted and not to be extracted.

The Bills-Date dataset is instead intrinsically difficult due to the presence of several different date formats. Indeed, also the baseline method struggles into finding an effective regular expression for extracting dates (obtaining an F-measure of 0.21): interestingly, several of our extractors outperforms the baseline on Bills-Date.

Besides providing a more detailed view of the extractor effectiveness, values shown in Table 5 allow appreciating the impact of multitoken and context-aware optimizations. The former is beneficial for datasets BibTex-Author and Email-Phone, as expected—in both cases, spaces, e.g., occur both inside desired extractions and as boundaries of desired extractions. The latter is beneficial for datasets Headers-ForToEmail, in particular for similarity-based ex-

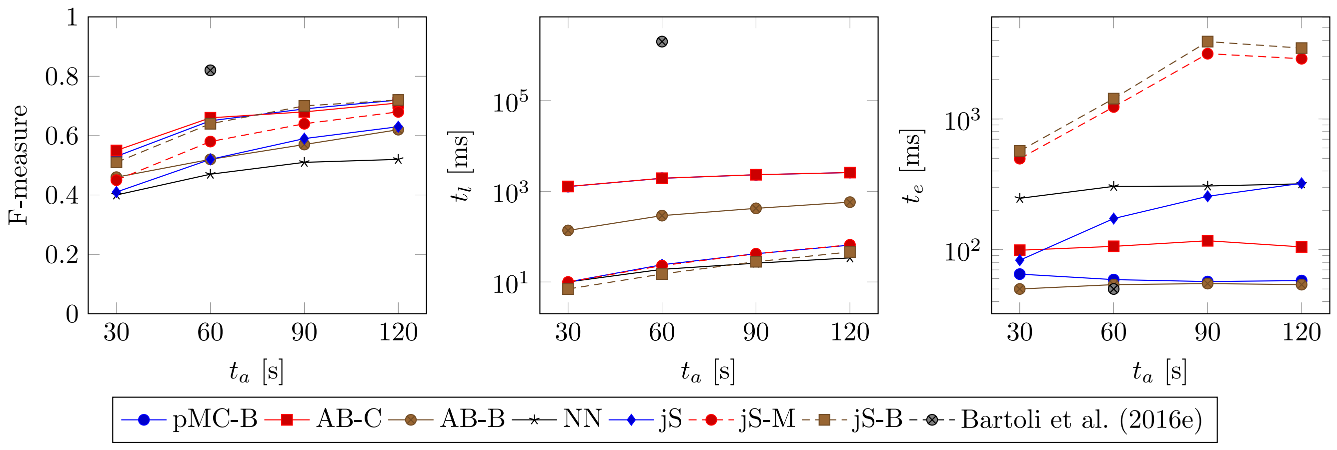


Figure 5: F-measure, learning time t_l, extraction time t_e obtained for different values of t_a with four selected extractors (see text) and with the baseline by Bartoli et al. (2016e): the vertical axis has logarithmic scale for t_l and t_e.

Extr.	BibTex-Author	Bills-Date	Email-Phone	Headers-ForToEmail	HTML-href	Log-IP	Twitter-Hashtag+Cit.	Twitter-URL	Twitter-Username	Web-URL	*Average*
MC	0.07	0.04	0.07	0.08	0.16	0.92	0.29	0.13	0.17	0.03	*0.20*
pMC	0.07	0.13	0.21	0.11	0.15	0.89	0.70	0.48	0.24	0.10	*0.31*
pMC-C	0.12	0.19	0.25	0.43	0.50	0.98	0.70	0.52	0.53	0.12	*0.43*
pMC-M	0.34	0.11	0.06	0.11	0.14	0.89	0.70	0.48	0.24	0.10	*0.32*
pMC-B	0.50	0.12	0.33	0.43	0.50	0.98	0.70	0.52	0.53	0.12	*0.47*
AB	0.19	0.44	0.48	0.30	0.65	**1.00**	0.87	0.95	0.60	0.64	*0.61*
AB-C	0.19	0.44	0.48	0.40	0.68	**1.00**	0.86	0.95	**0.83**	0.64	*0.65*
AB-M	0.15	0.18	0.67	0.28	0.63	**1.00**	0.86	0.95	0.60	0.64	*0.59*
AB-B	0.42	0.18	**0.67**	0.41	0.67	**1.00**	0.87	0.95	0.82	0.65	***0.66***
NN	0.19	**0.45**	0.41	0.14	0.24	**1.00**	**0.91**	0.97	0.69	0.23	*0.52*
NB	0.08	0.14	0.20	0.18	0.33	0.82	0.76	0.66	0.59	0.24	*0.40*
bRF	0.18	0.41	0.44	0.31	0.61	**1.00**	0.77	0.95	0.54	0.62	*0.58*
bRF-C	0.19	0.40	0.45	0.48	0.58	**1.00**	0.76	0.95	0.76	0.58	*0.62*
bRF-M	0.13	0.16	0.58	0.32	0.60	**1.00**	0.77	0.95	0.55	0.62	*0.57*
bRF-B	0.48	0.17	0.61	0.47	0.56	1.00	0.77	0.95	0.75	0.58	*0.63*
nRF	0.19	0.15	0.17	0.14	0.15	0.20	0.19	0.17	0.19	0.11	*0.17*
jS	0.19	0.32	0.27	0.42	**0.78**	0.99	0.35	0.98	0.24	**0.66**	*0.52*
jS-C	0.13	0.27	0.28	**0.64**	0.75	0.99	0.34	0.98	0.68	0.58	*0.56*
jS-M	0.40	0.28	0.67	0.42	0.77	0.99	0.35	0.98	0.24	**0.66**	*0.58*
jS-B	**0.52**	0.25	0.64	**0.64**	0.74	0.99	0.34	0.98	0.68	0.58	*0.64*
jwS	0.12	0.21	0.25	0.26	0.68	0.85	0.62	0.97	0.25	0.48	*0.47*
nwS	0.16	0.39	0.43	0.23	0.48	0.98	0.61	**0.99**	0.40	0.47	*0.51*
Bartoli et al. (2016e)	0.79	0.21	0.91	0.58	0.91	1.00	0.94	0.98	1.00	0.84	*0.82*

Table 5: F-measure obtained for $t_a = 60$ s with each extractor and dataset (values are averaged across repetitions). The best figures among our methods for each dataset are highlighted in bold.

tractors, and Twitter-Username, for which the improvement is large (up to $\approx +100\%$).

Finally, Table 5 gives an high level indication of which kinds of classifiers are suitable to which datasets. In particular, it can be seen that similarity-based classifiers appear to outperform feature-based classifiers on datasets in which the extractions are long, and the opposite. Indeed, the 5 datasets with the largest average length $\overline{\ell(x^\star)}$ of a desired extraction (see Table 2) are exactly the ones in which the best F-measure is achieved by a similarity-based classifier.

6.5. *Full system assessment*

6.5.1. *Procedure*

In the second experimental campaign we assessed our system as a whole, limiting to a set of combinations of extractors and active learning schemes selected based on the results in the first campaign. We chose the following 7 extractors—pMC-B, AB-C, AB-B, NN, jS, jS-M, jS-B—the sames we chose to build QbC. We combined these extractors with the 6 active learning variants presented in Section 5 (U, QbC, B, S, dU, and dB) and with an additional baseline scheme which selects as query x_q a random substring of s such that (i) its length is equal to the average length of substrings in the current learning sets X^m and X^u and (ii) it does not overlap any substring in $X^m \cup X^u$. We denote by R this active learning scheme; as an optimization, which also implies an advantage of R w.r.t. the other active learning schemes, the extractor is learned only once with R, just after the last user answer.

For each dataset, we built a number of problem instances $(s, x_0^m, x_0^u, X^\star)$ as follows: (i) we selected a random substring s of the dataset text such that the number $|X^\star|$ of desired extractions in s was exactly 100; (ii) we randomly selected x_0^m in $X^\star$; and (iii) we randomly selected as x_0^u a substring in s, not overlapping with any substring in $X^\star$ and such that $\ell(x_0^u) = \ell(x_0^m)$. We repeated this procedure 30 times for each dataset.

Finally, we applied the 43 variants of our method (42 resulting from the combination of the 7 extractors with the 6 active learning schemes—without QbC and with R—plus QbC) to each problem instance. We simulated the user by means of a program which always answered correctly to the query and required a time t_a for each query as defined in Eqq. 4 and 5. We stopped the execution after a (simulated) elapsed time of 60 s, i.e., upon the first query for which overall annotation and learning times exceeded one minute.

In order to place our results in perspective, we also executed a recent proposal for active learning of extractors of syntax-based entities (Bartoli et al., 2016a). The cited work is internally based on the work of Bartoli et al. (2016e) and hence uses Genetic Programming to build regular expressions tailored to learning examples. The active learning scheme of (Bartoli et al., 2016a) is a form of query-by-committee. We applied the cited approach on the same datasets with the same procedure used to assess our methods, with the exception of the termination criterion: for the tool by Bartoli et al. (2016a), due to its long execution time, we took into account *only* the annotation time t_a, rather than the elapsed time $t_a + t_l$, when comparing against the 60 s time budget. Moreover, due to the stochastic nature of Genetic Programming, we run the cited tool 5 times on each instance $(s, x_0^m, x_0^u, X^\star)$ (in a subset of 3 on the 30 instances) and averaged the results.

6.5.2. *Results and discussion: overview*

Table 6 shows the main results, averaged across datasets and repetitions: it shows the F-measure, the number $|X^m|$ ($|X^u|$) of annotated desired (undesired) extractions, the overall number $\ell(X^m \cup X^u)$ of annotated characters, the number #Q of queries, the percentage %BA of queries with a binary answer, the overall time $\sum t_a$ spent by the simulated user for annotating, and the overall time $\sum t_l$ spent by our system in executing the active learning procedure. All those figures refer to the end of the execution, i.e., after one minute. Some interesting observations can be done.

First, the proposed system, in its best variant, appears to meet the requirements: it is possible to achieve a good F-measure (0.67, for dU/AB-B) within the tight time budget of 60 s. To place this result in a perspective, recall that in the best case of passive learning, AB-B obtained a slightly lower F-measure (0.66 vs. 0.67, the difference being not statistically significant according to the Wilcoxon signed rank task, for which $p = 0.33$) with a learning set of substrings corresponding to 3547 characters; the 16 queries of dU/AB-B correspond instead to ≈ 431 characters. In other words, a neat decrease in the need for learning information is achieved by means of the proposed interactive learning system, without affecting the extraction effectiveness. With respect to the baseline (Bartoli et al., 2016a), Table 6 shows that our best variant obtains an F-measure which is only slightly smaller (0.67 vs. 0.69), but it takes a remarkably shorter learning time t_l (9.4 s vs. 558.2 s): the latter figure corresponds to an overall time $t_a + t_l$ which abundantly exceeds the 1 min budget. From another point of view, the time taken by the baseline between two consecutive queries is ≈ 40 s, which definitely does not allow to use that tool interactively: our best method, instead, takes around ≈ 0.5 s to propose a new query to the user.

Second, with the best extractors (AB-B and jS-B, and variants), all the considered active learning schemes (with the exception of S) outperform the random baseline (R) in F-measure: the uncertainty sampling scheme (U) and its density weighting variant (dU) obtain the best results, with an average increase of +75% over the baseline. This finding is corroborated by Table 7, which shows the absolute difference in the average F-measure along with the statistical significance (with Wilcoxon signed rank test), for the two best extractors (AB-B and jS-B) coupled with the two best active learning schemes (dU and U) and the baseline (R). It can be seen that dU/AB-B is significantly

AL	Extr.	Fm	Prec.	Rec.	$\lvert X^m\rvert$	$\lvert X^u\rvert$	ℓ	#Q	%BA	$\sum t_a$	$\sum t_l$
R	pMC-B	0.35±0.13	0.3	0.52	4	20	482	20	0.84	62	0
R	AB-C	0.4 ±0.16	0.51	0.44	4	20	482	20	0.84	62	0
R	AB-B	0.43±0.17	0.49	0.49	4	20	482	20	0.84	62	0
R	NN	0.41±0.13	0.55	0.45	4	20	482	20	0.84	62	0
R	jS	0.32±0.12	0.78	0.28	4	20	482	20	0.84	62	0
R	jS-M	0.35±0.14	0.74	0.31	4	20	482	20	0.84	62	0
R	jS-B	0.38±0.18	0.76	0.34	4	20	482	20	0.84	62	0
U	pMC-B	0.38±0.13	0.36	0.54	5	20	364	21	0.89	58	3.8
U	AB-C	0.6 ±0.14	0.78	0.59	7	14	407	18	0.84	53.2	9
U	AB-B	0.62±0.16	0.68	0.64	7	14	427	17	0.83	53	9.3
U	NN	0.48±0.1	0.76	0.49	8	15	473	19	0.81	60.9	1.3
U	jS	0.53±0.08	0.75	0.5	11	11	436	19	0.76	60.6	1.5
U	jS-M	0.58±0.1	0.78	0.54	11	11	434	18	0.74	59.9	2.4
U	jS-B	0.61±0.16	0.83	0.55	11	11	406	19	0.79	58.4	3.8
QbC		0.37±0.15	0.36	0.63	7	10	322	13	0.77	42.9	20.4
B	pMC-B	0.33±0.1	0.31	0.51	11	12	408	21	0.89	58.5	3.3
B	AB-C	0.56±0.12	0.71	0.56	10	11	428	18	0.85	53.2	9
B	AB-B	0.6 ±0.15	0.66	0.62	10	11	432	17	0.84	52.9	9.4
B	NN	0.48±0.08	0.71	0.52	10	13	584	19	0.83	60.7	1.2
B	jS	0.48±0.11	0.76	0.45	11	12	375	20	0.79	60.4	1.4
B	jS-M	0.53±0.13	0.79	0.49	11	13	379	19	0.79	59.9	2
B	jS-B	0.55±0.17	0.83	0.48	11	12	385	20	0.84	58.1	3.9
S	pMC-B	0.25±0.11	0.27	0.33	2	22	247	23	0.93	55.5	6.3
S	AB-C	0.26±0.16	0.53	0.23	2	22	236	23	0.92	54.8	7.3
S	AB-B	0.27±0.16	0.48	0.26	2	21	234	22	0.92	53.8	8.3
S	NN	0.32±0.12	0.36	0.4	2	24	246	24	0.93	58	3.8
S	jS	0.23±0.1	0.76	0.19	3	23	246	24	0.92	58.4	3.3
S	jS-M	0.25±0.11	0.7	0.21	3	23	244	24	0.92	57.5	4.3
S	jS-B	0.24±0.15	0.79	0.2	3	22	247	23	0.93	56	5.9
dU	pMC-B	0.35±0.12	0.31	0.61	6	17	427	20	0.85	58.3	3.6
dU	AB-C	0.63±0.12	0.75	0.65	8	12	417	17	0.82	53.3	9
dU	AB-B	0.67±0.15	0.7	0.7	8	12	431	16	0.8	52.8	9.4
dU	NN	0.49±0.09	0.67	0.52	9	13	433	19	0.77	60.5	1.4
dU	jS	0.52±0.08	0.79	0.49	11	11	438	19	0.77	60.3	1.7
dU	jS-M	0.57±0.1	0.82	0.53	11	12	439	18	0.74	59.7	2.9
dU	jS-B	0.63±0.13	0.81	0.6	12	10	416	18	0.77	58	4.2
dB	pMC-B	0.32±0.09	0.26	0.57	11	12	453	20	0.85	58.3	3.5
dB	AB-C	0.59±0.11	0.7	0.61	10	11	418	17	0.84	52.8	9.4
dB	AB-B	0.61±0.13	0.62	0.67	10	11	428	17	0.81	52.4	10
dB	NN	0.5 ±0.09	0.66	0.55	10	12	502	19	0.81	60.5	1.4
dB	jS	0.47±0.1	0.79	0.43	10	13	398	20	0.81	60.3	1.5
dB	jS-M	0.52±0.11	0.82	0.48	10	13	401	20	0.78	59.7	2.5
dB	jS-B	0.58±0.14	0.77	0.53	11	12	405	19	0.81	57.9	4.2
Bartoli et al. (2016a)		0.69			6	8	452	13	0.85	60.0	558.2

Table 6: Results of the full system variants. Values are averaged across repetitions and datasets: for F-measure, the average across datasets of the standard deviation across repetitions on the same dataset is also shown. $\sum t_a$ and $\sum t_l$ are expressed in seconds. ℓ columns shows $\ell(X^m \cup X^u)$ values.

AL	Extr.	R jS-B	U AB-B	U jS-B	dU AB-B	dU jS-B
R	jS-B	-0.05*				
U	AB-B	0.19‡	0.24‡			
U	jS-B	0.17‡	0.22‡	-0.01		
dU	AB-B	0.23‡	0.28‡	0.04†	0.06*	
dU	jS-B	0.20‡	0.25‡	0.01	0.02$^{\cdot}$	-0.04

Table 7: Differences of average F-measure, between pair of 6 selected methods. For each pair, the statistical significance is shown: $\cdot$: $p < 0.1$, $*$: $p < 0.05$, $\dagger$: $p < 0.01$, $\ddagger$: $p < 0.001$, $p \geq 0.1$ without any subscript.

better ($p \leq 0.05$) than all the other methods with the exception of dU/jS-B: we argue that the small gap between the two methods is favored by the fact that jS-B is in general faster in learning, hence allowing for more time devoted to user annotation with repsect to dU/AB-B—indeed, Table 6 shows that, on average, the latter poses two queries less than dU/jS-B (16 vs. 18). Concerning the other two extractors (pMC-Ms and NN), it appears from the results in Table 6 that their learning effectiveness is not clearly positively affected by any active learning scheme: in particular, the former delivers an F-measure which is likely too low, regardless of the annotated data it can learn on. According to the F-measure, QbC/* appears to be not adequate for the considered scenario: its effectiveness is roughly the same of the average extractor when coupled with the baseline active learning scheme (R). Moreover, QbC/* takes the longest time to learn ($t_l = 20.4\,\mathrm{s}$), a figure which is consistent with the scheme working principle which requires to train, before each query, 7 competing extractors. Finally, Table 6 shows that the least similar instance active learning scheme S performs poorly, regardless of the extractor. We explain this finding by the fact that undesired extractions have larger entropy (i.e., they are more dissimilar) than extractions: this leads the system to builds queries which will be answered by the user as undesired extractions, making the learning set $X^m \cup X^u$ unbalanced and, eventually, the extractor ineffective.

Third, it can be seen that in most cases the available time has been used almost entirely for exploiting user feedback, i.e., $\sum t_a \gg \sum t_l$. Moreover, queries caused in general binary answers (%BA $\approx$ 80%) rather than edit answers.

Fourth, it can be observed that F-measure tends to be greater when the system allowed the user to provide a balanced learning information. This consideration applies mainly to B and dB active learning schemes, which were indeed designed right with this aim. Moreover, the balancing is near optimal also for combinations of U and dU schemes with similarity-based extractors.

6.5.3. Differences among datasets

Table 8 shows the F-measure of our variants and the baseline for the 10 datasets, averaged across repetitions.

It can be seen that the uncertainty-based scheme (U) is the best scheme in the largest number of datasets and improves over the random scheme on the vast majority of cases. It can also be observed that the improvement is greater for those datasets in which the desired extractions are sparser (Headers-ForToEmail, Email-Phone, and Web-URL): looking at raw results, we also verified that in those cases the percentage $\%|X^m|$ of labeled desired extractions was remarkably low with R than with the other schemes—i.e., learning set built with R were highly unbalanced towards undesired extractions. This finding is further corroborated by Figure 6, which shows, for one repetition of the application of the variants R/jS-B and dU/jS-B on two datasets, how the F-measure on the full s varies over the time. In the Log-IP dataset (Figure 6a), where desired extractions are dense ($\rho = 0.24$), the R scheme is able to provide informative data to the extractor by simply querying random substrings; on the other hand, in the Web-URL dataset (Figure 6b), where desired extractions are sparse ($\rho = 0.04$), few improvements are achieved over the time by R which continuously query substrings which correspond to undesired extractions, whereas dU is instead able to query the user with informative substrings. As an aside, it can also be observed that in the dense Log-IP dataset, R tends to query substrings requiring edit answers (which hence take longer for the user), whereas U tends to result in binary answers: this is the reason for which marks are more spaced for R/jS-B curve than for dU/jS-B curve in Figure 6a.

Finally, Figure 6 also shows how the F-measures increases with t_a, i.e., as the user provides more learning data. In particular, Figure 6a shows that in the Log-IP dataset (as in other cases) a perfect (or very high) effectiveness can be achieved with our active learning system well before one minute. On the other hand, in passive learning the user is required to spend the full minute in annotating the text before being able to see which are the found text items.

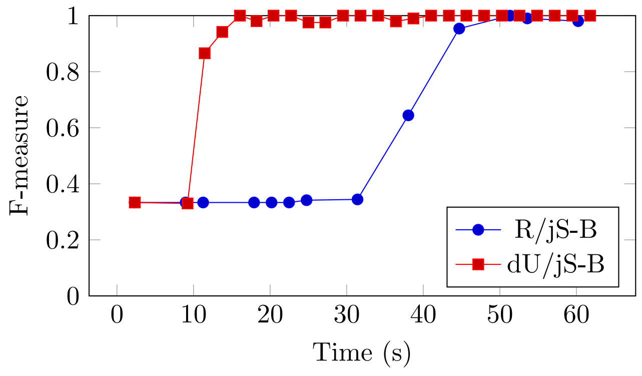
6.5.4. Larger data and longer time

We designed our system focusing on a use case in which the user has a time budget of one minute for extracting approximately 100 text items of interest from a text of up to tens of thousands characters.

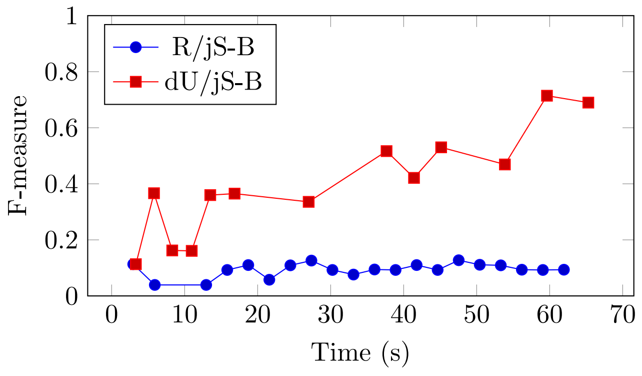
In this section, we present the results of another experimental campaign we performed for investigating about the effectiveness of our proposal in a scenario in which the text and the time budget are longer. To this end, we selected 6 variants of our method (resulting from the combination of AB-B and jS-B extractors with R, U, and dU active learning schemes) and repeated the procedure of Section 6.5 with the following changes: we built problem instances such that $|X^\star|$ was exactly 500 (instead of 100), we con-

AL	Extr.	BibTex-Author	Bills-Date	Email-Phone	Headers-ForToEmail	HTML-href	Log-IP	Twitter-Hashtag+Cit.	Twitter-URL	Twitter-Username	Web-URL	*Average*
R	pMC-B	0.40	0.07	0.18	0.27	0.29	0.84	0.56	0.38	0.36	0.14	*0.35*
R	AB-C	0.10	0.18	0.21	0.21	0.37	0.81	0.60	0.77	0.53	0.25	*0.40*
R	AB-B	0.37	0.13	0.27	0.22	0.39	0.81	0.61	0.76	0.54	0.25	*0.43*
R	NN	0.09	0.19	0.19	0.11	0.18	0.93	0.70	0.90	0.47	0.31	*0.41*
R	jS	0.10	0.11	0.13	0.22	0.35	0.92	0.12	0.92	0.08	0.27	*0.32*
R	jS-M	0.15	0.10	0.40	0.22	0.35	0.92	0.12	0.92	0.08	0.27	*0.35*
R	jS-B	0.51	0.06	0.26	0.38	0.46	0.89	0.15	0.71	0.25	0.15	*0.38*
U	pMC-B	0.48	0.07	0.23	0.25	0.29	0.99	0.67	0.38	0.38	0.12	*0.38*
U	AB-C	0.12	0.33	0.42	0.33	0.60	**1.00**	0.80	0.96	0.84	0.59	*0.60*
U	AB-B	0.36	0.21	0.51	0.37	0.58	**1.00**	0.82	0.96	0.83	0.58	*0.62*
U	NN	0.13	0.37	0.32	0.05	0.21	0.87	0.97	0.99	0.65	0.28	*0.48*
U	jS	0.12	0.30	0.24	0.22	**0.96**	**1.00**	0.62	**1.00**	0.17	0.70	*0.53*
U	jS-M	0.35	0.21	**0.66**	0.22	0.91	**1.00**	0.62	**1.00**	0.17	0.70	*0.58*
U	jS-B	0.48	0.17	0.51	0.48	0.85	0.97	0.33	0.99	0.82	0.48	*0.61*
QbC		0.39	0.14	0.29	0.15	0.21	0.99	0.65	0.21	0.46	0.21	*0.37*
B	pMC-B	0.44	0.06	0.12	0.11	0.19	0.97	0.52	0.34	0.42	0.11	*0.33*
B	AB-C	0.12	0.34	0.34	0.40	0.46	**1.00**	0.70	0.96	0.79	0.45	*0.56*
B	AB-B	0.38	0.25	0.47	0.45	0.50	**1.00**	0.72	0.95	0.79	0.51	*0.60*
B	NN	0.13	0.31	0.31	0.05	0.18	**1.00**	0.93	0.99	0.56	0.31	*0.48*
B	jS	0.12	0.27	0.22	0.20	0.87	0.99	0.41	0.95	0.17	0.62	*0.48*
B	jS-M	0.32	0.18	0.61	0.20	0.85	0.99	0.41	0.95	0.17	0.62	*0.53*
B	jS-B	0.51	0.18	0.43	0.43	0.76	0.97	0.29	0.93	0.60	0.35	*0.55*
S	pMC-B	0.30	0.05	0.14	0.16	0.14	0.87	0.22	0.23	0.30	0.10	*0.25*
S	AB-C	0.11	0.15	0.11	0.25	0.22	0.76	0.13	0.19	0.40	0.21	*0.26*
S	AB-B	0.32	0.09	0.13	0.25	0.23	0.73	0.13	0.17	0.40	0.22	*0.27*
S	NN	0.06	0.16	0.13	0.12	0.16	0.85	0.33	0.85	0.31	0.22	*0.32*
S	jS	0.08	0.08	0.06	0.20	0.12	0.98	0.03	0.51	0.07	0.19	*0.23*
S	jS-M	0.14	0.07	0.15	0.20	0.12	0.98	0.03	0.51	0.07	0.19	*0.25*
S	jS-B	0.44	0.06	0.10	0.33	0.27	0.86	0.03	0.04	0.18	0.12	*0.24*
dU	pMC-B	0.45	0.09	0.15	0.25	0.27	0.91	0.68	0.25	0.36	0.10	*0.35*
dU	AB-C	0.11	0.38	0.45	0.44	0.69	**1.00**	0.87	0.94	0.86	0.55	*0.63*
dU	AB-B	0.37	0.31	0.56	0.49	0.71	**1.00**	0.87	0.92	0.87	0.57	***0.67***
dU	NN	0.13	0.34	0.26	0.07	0.28	0.92	**0.97**	0.98	0.64	0.32	*0.49*
dU	jS	0.14	0.31	0.23	0.22	0.96	1.00	0.54	0.98	0.11	**0.75**	*0.52*
dU	jS-M	0.30	0.21	0.66	0.22	0.96	1.00	0.54	0.98	0.11	0.75	*0.57*
dU	jS-B	0.50	0.16	0.42	0.55	0.90	0.99	0.33	0.99	**0.88**	0.58	*0.63*
dB	pMC-B	0.41	0.07	0.10	0.13	0.19	0.89	0.61	0.30	0.39	0.09	*0.32*
dB	AB-C	0.11	**0.38**	0.36	0.44	0.63	**1.00**	0.75	0.94	0.81	0.45	*0.59*
dB	AB-B	0.41	0.27	0.49	0.45	0.62	**1.00**	0.67	0.94	0.82	0.42	*0.61*
dB	NN	0.12	0.37	0.28	0.09	0.25	**1.00**	0.93	0.98	0.60	0.35	*0.50*
dB	jS	0.13	0.28	0.21	0.21	0.92	0.97	0.36	0.93	0.11	0.61	*0.47*
dB	jS-M	0.29	0.21	0.64	0.21	0.92	0.97	0.36	0.93	0.11	0.61	*0.52*
dB	jS-B	**0.52**	0.17	0.38	**0.58**	0.80	0.98	0.29	0.93	0.71	0.41	*0.58*
Bartoli et al. (2016a)		0.61	0.34	0.76	0.24	0.68	0.97	0.86	0.97	0.88	0.54	*0.69*

Table 8: F-measure for each dataset (values are averaged across repetitions). The best figures among our methods for each dataset are highlighted in bold.



(a) Log-IP ($\rho^\star = 0.24$).



(b) Web-URL ($\rho^\star = 0.04$).

Figure 6: Learning curves for one repetition of the application of two variants on two datasets.

sidered an elapsed time of 300 s (instead of 60 s), and we built 10 instances for each dataset (instead of 30).

Table 9 shows the results of this experimental campaign: two key observations may be done.

First, it can be seen that the F-measure is larger than in the reference use case for all the 6 variants. For the best method (again dU/AB-B) F-measure reaches 0.73, being 0.66 in the reference use case. It can also be noted that the improvement is larger for the R scheme. This finding confirms that in the reference use case, when the time budget is tight, having a good active learning scheme is fundamental for achieving a good effectiveness.

Second, in this scenario the differences in F-measure among the four best variants (based on U, dU and AB-B, jS-B) look vanishing. On the other hand, the difference in the learning time t_l between AB-B and jS-B is now larger and opposite with respect to the reference use case: this is a consequence of how the two extractors work and, in particular, of the fact that, for similarity-based extractors, the learning time grows quadratically with the amount of learning data (see Section 6.4). In practice, the average time between two consecutive queries is ≈ 1.4 s for dU/AB-B and ≈ 1.7 s for dU/jS-B. It is fair to claim that in both cases, despite being larger than the respective figures in the reference use case (≈ 0.5 s and ≈ 0.2 s, respectively), those times are sufficiently small to enable interactive usage of our methodology.

7. Concluding remarks

There are several use cases of practical interest, broadly related to document analysis and exploration, in which the problem of identifying very quickly text items which follow a certain pattern is not addressed by existing technologies well. We have proposed an interactive framework in which the user provides only examples of the items he is interested in; the system identifies items similar to those provided by the user and progressively refines the similarity criterion by submitting selected queries to the user, in an active learning fashion.

Actually implementing the proposed framework is challenging because the requirement for interactive execution places severe constraints on the algorithms that can be used for: (i) inferring a general pattern from the available examples; (ii) selecting from the input document the next query to be submitted to the user; (iii) allowing the user to answer the query; and, (iv) actually finding all occurrences of the current pattern within the document.

We have analyzed a number of different designs and assessed them experimentally in depth, by carefully modelling the user time involved in the active learning interaction. The key outcome of our work is that our proposal is indeed practically feasible, as several of the design options that we have analyzed are able to achieve accuracy comparable to that obtained with a state-of-the-art approach for constructing a regular expression by means examples of the desired behavior, while requiring an execution time that is orders of magnitude shorter and that is sufficiently short to enable interactive execution.

Acknowledgements

This work has been partially supported by the University of Trieste - Finanziamento di Ateneo per progetti di ricerca scientifica - FRA 2016.

References

Angeli, G., Tibshirani, J., Wu, J., & Manning, C. D. (2014). Combining distant and partial supervision for relation extraction. In *EMNLP* (pp. 1556–1567).

Bajaj, K., Pattabiraman, K., & Mesbah, A. (2015). Synthesizing web element locators (t). In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on* (pp. 331–341). IEEE.

Bartoli, A., Davanzo, G., De Lorenzo, A., Medvet, E., & Sorio, E. (2014a). Automatic synthesis of regular expressions from examples. *Computer*, *47*, 72–80.

Bartoli, A., Davanzo, G., Medvet, E., & Sorio, E. (2014b). Semisupervised wrapper choice and generation for print-oriented documents. *IEEE Transactions on Knowledge and Data Engineering*, *26*, 208–220.

AL	Extr.	Fm	Prec.	Rec.	$\lvert X^m \rvert$	$\lvert X^u \rvert$	ℓ	#Q	%BA	$\sum t_a$	$\sum t_l$
R	AB-B	0.61±0.11	0.59	0.67	15	97	2361	98	0.85	301.8	0
R	jS-B	0.64±0.12	0.73	0.63	15	97	2361	98	0.85	301.8	0
U	AB-B	0.71±0.1	0.76	0.72	32	48	1367	69	0.83	207.6	95.2
U	jS-B	0.71±0.11	0.86	0.67	47	28	1366	68	0.78	201.3	125.2
dU	AB-B	0.73±0.11	0.75	0.75	39	42	1371	67	0.78	210.2	93
dU	jS-B	0.71±0.1	0.83	0.7	48	26	1345	66	0.76	200	115.2

Table 9: Results of 4 selected full system variants with larger data ($\lvert X^\star \rvert = 500$) and longer elapsed time (300 s). Values are averaged across repetitions and datasets: for F-measure, the average across datasets of the standard deviation across repetitions on the same dataset is also shown. $\sum t_a$ and $\sum t_l$ are expressed in seconds. ℓ columns shows $\ell(X^m \cup X^u)$ values.

Bartoli, A., De Lorenzo, A., Medvet, E., & Tarlao, F. (2015a). Learning text patterns using separate-and-conquer genetic programming. In *Genetic Programming* (pp. 16–27). Springer.

Bartoli, A., De Lorenzo, A., Medvet, E., & Tarlao, F. (2016a). Active learning approaches for learning regular expressions with genetic programming. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing* (pp. 97–102). ACM.

Bartoli, A., De Lorenzo, A., Medvet, E., & Tarlao, F. (2016b). Can a machine replace humans in building regular expressions? a case study. *IEEE Intelligent Systems*, *31*, 15–21. doi:10.1109/MIS.2016.46.

Bartoli, A., De Lorenzo, A., Medvet, E., & Tarlao, F. (2016c). Predicting the effectiveness of pattern-based entity extractor inference. *Applied Soft Computing*, *46*, 398–406.

Bartoli, A., De Lorenzo, A., Medvet, E., & Tarlao, F. (2016d). Syntactical similarity learning by means of grammatical evolution. In J. Handl, E. Hart, P. R. Lewis, M. López-Ibáñez, G. Ochoa, & B. Paechter (Eds.), *Parallel Problem Solving from Nature – PPSN XIV: 14th International Conference, Edinburgh, UK, September 17-21, 2016, Proceedings* (pp. 260–269). Cham: Springer International Publishing.

Bartoli, A., De Lorenzo, A., Medvet, E., & Tarlao, F. (2017). Active learning of regular expressions for entity extraction. *IEEE Transactions on Cybernetics*, . doi:10.1109/TCYB.2017.2680466.

Bartoli, A., Lorenzo, A. D., Medvet, E., & Tarlao, F. (2015b). Data quality challenge: Toward a tool for string processing by examples. *Journal of Data and Information Quality (JDIQ)*, *6*, 13.

Bartoli, A., Lorenzo, A. D., Medvet, E., & Tarlao, F. (2016e). Inference of regular expressions for text extraction from examples. *IEEE Transactions on Knowledge and Data Engineering*, *28*, 1217–1230.

Bilenko, M., Mooney, R., Cohen, W., Ravikumar, P., & Fienberg, S. (2003). Adaptive name matching in information integration. *IEEE Intelligent Systems*, *18*, 16–23.

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

Breiman, L. (2001). Random forests. *Machine learning*, *45*, 5–32.

Budlong, E., Pine, C., Zappavigna, M., Homer, J., Proefrock, C., Gucwa, J., Crystal, M., & Weischedel, R. M. (2013). Interactive information extraction and navigation to enable effective link analysis and visualization of unstructured text. In *IAAI*.

Carlson, A., Betteridge, J., Kisiel, B., Settles, B., Hruschka Jr, E. R., & Mitchell, T. M. (2010). Toward an architecture for never-ending language learning. In *AAAI* (p. 3). volume 5.

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

Chandrasekar, C. et al. (2013). An optimized approach of modified bat algorithm to record deduplication. *International Journal of Computer Applications*, *62*.

Cheatham, M., & Hitzler, P. (2013a). String similarity metrics for ontology alignment. In *The Semantic Web–ISWC 2013* (pp. 294–309). Springer.

Cheatham, M., & Hitzler, P. (2013b). String similarity metrics for ontology alignment. In *International Semantic Web Conference*.

Chin, W.-S., Zhuang, Y., Juan, Y.-C., Wu, F., Tung, H.-Y., Yu, T., Wang, J.-P., Chang, C.-X., Yang, C.-P., Chang, W.-C. et al. (2014). Effective string processing and matching for author disambiguation. *The Journal of Machine Learning Research*, *15*, 3037–3064.

Cochran, R. A., D'Antoni, L., Livshits, B., Molnar, D., & Veanes, M. (2015). Program boosting: Program synthesis via crowd-sourcing. In *ACM SIGPLAN Notices* (pp. 677–688). ACM volume 50.

Cohen, W., Ravikumar, P., & Fienberg, S. (2003). A comparison of string metrics for matching names and records. In *Kdd workshop on data cleaning and object consolidation* (pp. 73–78). volume 3.

Culotta, A., & McCallum, A. (2005). Joint deduplication of multiple record types in relational data. In *Proceedings of the 14th ACM international conference on Information and knowledge management* (pp. 257–258). ACM.

Dalvi, B., Bhakthavatsalam, S., Clark, P., Clark, P., Etzioni, O., Fader, A., & Groeneveld, D. (2016). Ike-an interactive tool for knowledge extraction. In *5th AKBC Workshop*.

De Lorenzo, A., Medvet, E., & Bartoli, A. (2013). Automatic string replace by examples. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation* GECCO '13 (pp. 1253–1260). New York, NY, USA: ACM.

Donmez, P., & Carbonell, J. G. (2008). Proactive learning: cost-sensitive active learning with multiple imperfect oracles. In *Proceedings of the 17th ACM conference on Information and knowledge management* (pp. 619–628). ACM.

Etzioni, O., Cafarella, M., Downey, D., Kok, S., Popescu, A.-M., Shaked, T., Soderland, S., Weld, D. S., & Yates, A. (2004). Web-scale information extraction in knowitall:(preliminary results). In *Proceedings of the 13th international conference on World Wide Web* (pp. 100–110). ACM.

Etzioni, O., Cafarella, M., Downey, D., Popescu, A.-M., Shaked, T., Soderland, S., Weld, D. S., & Yates, A. (2005). Unsupervised named-entity extraction from the web: An experimental study. *Artificial intelligence*, *165*, 91–134.

Finkel, J. R., Grenager, T., & Manning, C. (2005). Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd annual meeting on association for computational linguistics* (pp. 363–370). Association for Computational Linguistics.

Fisher, K., Walker, D., Zhu, K. Q., & White, P. (2008). From dirt to shovels: fully automatic tool generation from ad hoc data. In *ACM SIGPLAN Notices* (pp. 421–434). ACM volume 43.

Freund, Y., & Schapire, R. E. (1995). A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory* (pp. 23–37). Springer.

Gualtieri, M. (2011). *Empowering The "Business Developer"*. Technical Report Forrester Research.

Gulwani, S. (2016). Programming by examples: Applications, algorithms, and ambiguity resolution. In N. Olivetti, & A. Tiwari (Eds.), *Automated Reasoning: 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 – July 2, 2016, Proceedings* (pp. 9–14). Cham: Springer International Publishing.

Haertel, R., Ringger, E. K., Felt, P., & Seppi, K. (2015). An analytic and empirical evaluation of return-on-investment-based active learning. In *The 9th Linguistic Annotation Workshop held in conjuncion with NAACL 2015* (p. 11).

Haertel, R., Seppi, K. D., Ringger, E. K., & Carroll, J. L. (2008). Return on investment for active learning. In *Proceedings of the NIPS Workshop on Cost-Sensitive Learning*. volume 72.

Hu, H., Zheng, K., Wang, X., & Zhou, A. (2015). Gfilter: A general gram filter for string similarity search. *IEEE Transactions on Knowledge and Data Engineering*, *27*, 1005–1018.

Jiang, Y., Li, G., Feng, J., & Li, W.-S. (2014). String similarity joins: An experimental evaluation. *PVLDB*, *7*, 625–636.

Jin, L., Li, C., & Mehrotra, S. (2003). Efficient record linkage in large data sets. In *Database Systems for Advanced Applications, 2003.(DASFAA 2003). Proceedings. Eighth International Conference on* (pp. 137–146). IEEE.

Le, V., & Gulwani, S. (2014). Flashextract: a framework for data extraction by examples. In *ACM SIGPLAN Notices* (pp. 542–553). ACM volume 49.

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

McCallum, A., & Li, W. (2003). Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4* (pp. 188–191). Association for Computational Linguistics.

McCallum, A., Nigam, K. et al. (1998). A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization* (pp. 41–48). Citeseer volume 752.

McCallum, A. K., & Nigam, K. (1998). Employing em in pool-based active learning for text classification. In *Proceedings of ICML-98, 15th International Conference on Machine Learning*.

Medvet, E., Bartoli, A., & Davanzo, G. (2011). A probabilistic approach to printed document understanding. *International Journal on Document Analysis and Recognition (IJDAR)*, *14*, 335–347.

Needleman, S. B., & Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, *48*, 443–453.

Pennacchiotti, M., & Pantel, P. (2009). Entity extraction via ensemble semantics. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1* (pp. 238–247). Association for Computational Linguistics.

Settles, B. (2009). *Active Learning Literature Survey*. Computer Sciences Technical Report 1648 University of Wisconsin–Madison.

Settles, B., & Craven, M. (2008). An analysis of active learning strategies for sequence labeling tasks. In *Proceedings of the conference on empirical methods in natural language processing* (pp. 1070–1079). Association for Computational Linguistics.

Settles, B., Craven, M., & Friedland, L. (2008). Active learning with real annotation costs. In *Proceedings of the NIPS workshop on cost-sensitive learning* (pp. 1–10).

Yates, A., Cafarella, M., Banko, M., Etzioni, O., Broadhead, M., & Soderland, S. (2007). Textrunner: open information extraction on the web. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations* (pp. 25–26). Association for Computational Linguistics.

Yessenov, K., Tulsiani, S., Menon, A., Miller, R. C., Gulwani, S., Lampson, B., & Kalai, A. (2013). A colorful approach to text processing by example. In *Proceedings of the 26th annual ACM symposium on User interface software and technology* (pp. 495–504). ACM.

Yu, M., Li, G., Deng, D., & Feng, J. (2016). String similarity search and join: a survey. *Frontiers of Computer Science*, *10*, 399–417.

Zhang, Z. (2008). Mining relational data from text: From strictly supervised to weakly supervised learning. *Information Systems*, *33*, 300–314.