Predicting the Effectiveness of Pattern-based Entity Extractor Inference

Alberto Bartoli, Andrea De Lorenzo, Eric Medvet, Fabiano Tarlao

DIA, University of Trieste, Italy

Abstract

An essential component of any workflow leveraging digital data consists in the identification and extraction of relevant *patterns* from a data stream. We consider a scenario in which an *extraction inference engine* generates an *entity extractor* automatically from examples of the desired behavior, which take the form of user-provided annotations of the entities to be extracted from a dataset. We propose a methodology for *predicting* the accuracy of the extractor that may be inferred from the available examples. We propose several prediction techniques and analyze experimentally our proposals in great depth, with reference to extractors consisting of regular expressions. The results suggest that reliable predictions for tasks of practical complexity may indeed be obtained quickly and without actually generating the entity extractor.

Keywords: String similarity metrics, Information extraction, Genetic programming, Hardness estimation

1. Introduction

An essential component of any workflow leveraging digital data consists in the identification and extraction of relevant *patterns* from a data stream. This task occurs routinely in virtually every sector of business, government, science, technology, and so on. In this work we are concerned with extraction from an unstructured text stream of entities that adhere to a *syntactic* pattern. We consider a scenario in which an extractor is obtained by tailoring a generic tool to a specific problem instance. The extractor may consist, e.g., of a regular expression, or of an expression in a more general formalism [1], or of full programs suitable to be executed by NLP tools [2, 3]. The problem instance is characterized by a dataset from which a specified entity type is to be extracted, e.g., VAT numbers, IP addresses, or more complex entities.

The difficulty of generating an extractor is clearly dependent on the specific problem. However, we are not aware of any methodology for providing a practically useful answer to questions of this sort: generating an extractor for describing IP addresses is more or less difficult than generating one for extracting email addresses? Is it possible to generate an extractor for drug dosages in medical recipes, or for ingredients in cake recipes, with a specified accuracy level? Does the difficulty of generating an extractor for a specified entity type depend on the properties of the text that is *not* to be extracted? Not only answering such questions may provide crucial insights on extractor generation techniques, it may also be of practical interest to end users. For example, a prediction of low effectiveness could be exploited by providing more examples of the desired extraction behavior; the user might even decide to adopt a manual approach, perhaps in crowdsourcing, for problems that appear to be beyond the scope of the extractor generation technique being used.

In this work we propose an approach for addressing questions of this sort systematically. We consider on a scenario of increasing interest in which the problem instance is specified by *examples* of the desired behavior and the target extractor is generated based on those examples automatically [4, 5, 6, 7, 8, 9, 10, 11, 12]. We propose a methodology for *predicting* the accuracy of the extractor that may be inferred by a given extraction inference engine from the available examples. Our prediction methodology does not depend on the inference engine internals and can in principle be applied to any inference engine: indeed, we validate it on two different engines which infer different forms of extractors.

The basic idea is to use string similarity metrics to characterize the examples. In this respect, an "easy" problem instance is one in which (i) strings to be extracted are "similar" to each other, (ii) strings not to be extracted are "similar" to each other, and (iii) strings to be extracted are not "similar" to strings not to be extracted. Despite its apparent simplicity, implementing this idea is highly challenging for several reasons.

To be practically useful, a prediction methodology shall satisfy these requirements: (a) the prediction must be reliable; (b) it must be computed without actually generating the extractor; (c) it must be computed very quickly w.r.t. the time taken for inferring the extractor. First and foremost, predicting the performance of a solution without actually generating the solution is clearly very difficult (see

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

also the related work section).

Second, it is not clear to which degree a string similarity metric can capture the actual difficulty in inferring an extractor for a given problem instance. Consider, for instance, the Levenshtein distance (string edit distance) applied to a problem instance in which entities to be extracted are dates. Two dates (e.g., `2-3-1979` and `7-2-2011`, whose edit distance is 6) could be as distant as a date and a snippet not to be extracted (e.g, `2-3-1979` and `19.79$`, whose edit distance is 6 too); yet dates could be extracted by an extractor in the form of regular expression that is very compact, does not extract any of the other snippets and could be very easy to generate (`\d+-\d+-\d+`). However, many string similarity metrics exist and their effectiveness is tightly dependent on the specific application [13, 14]. Indeed, one of the contributions of our proposal is precisely to investigate which metric is the most suitable for assessing the difficulty of extractor inference.

Third, the number of snippets in an input text grows quadratically with the text size and becomes huge very quickly—e.g., a text composed of just 10^5 characters includes $\approx 10^{10}$ snippets. It follows that computing forms of similarity between all pairs of snippets may be feasible for snippets that are to be extracted but is not practically feasible for snippets that are *not* to be extracted.

We propose several prediction techniques and analyze experimentally our proposals in great depth, with reference to a number of different similarity metrics and of challenging problem instances. We validate our techniques with respect to a state-of-the-art extractor generator[1] approach that we have recently proposed [9, 5, 6]; we further validate our predictor on a worse-performing alternative extractor generator [15] which works internally in a different way. The results are highly encouraging suggesting that reliable predictions for tasks of practical complexity may indeed be obtained quickly.

2. Related work

Although we are not aware of any work specifically devoted to predicting the effectiveness of a pattern-based entity extractor inference method, there are several research fields that addressed similar issues. The underlying common motivation is twofold: inferring a solution to a given problem instance may be a lengthy procedure; and, the inference procedure is based on heuristics that cannot provide any optimality guarantees. Consequently, lightweight methods for estimating the quality of a solution before actually generating that solution are highly desirable.

In *combinatorial optimization* a wealth of research efforts have been devoted to the problem of estimating the difficulty of a given problem instance [16]. Such efforts may be broadly categorized in two classes: identifying features of a problem instance which may impact difficulty in terms of quality of a solution; and, identifying problem instance-independent features that may help in characterizing the difficulty of a task in general.

The work in [17] considers a specific class of combinatorial optimization tasks (TSP: travelling salesman problem) and follows a different line of research aimed at identifying features of a problem instance that may be helpful in choosing from a portfolio of algorithms the best one for that instance. The cited work actually considers only two such algorithms and assesses the ability of several classifiers, trained on a number of problem instances, to predict the relative performance of these two algorithms.

A recent proposal in this area followed a common approach consisting in the generation of a number of problem instances for a specific problem class (TSP) by means of a parametrized generation method [18]. A regressor for the solutions was then generated by using features of each problem instance that included values for the generation parameter. Our approach is similar except that we address a radically different task, thereby calling for radically different features.

An indirect but strong indication that the problem that we are addressing is amenable only to heuristic solutions without any formal guarantee is provided in [19], which considers optimization problems and proves that predictive measures that run in polynomial time do not exist.

Problem difficulty prediction is a very important research topic in *evolutionary computation*: an excellent survey can be found in [20]. The cited work presents a general method for estimating performance of evolutionary program induction algorithms with an experimental evaluation on two important classes of tasks, i.e., symbolic regression and Boolean function induction. The method is based on regressors trained on features extracted from problem instances. Features are defined over forms of distances computed over input-output pairs of the problem instance. We are not aware of any instantiation of this method for application domains involving string similarity computations, where there are many metrics that can be used and their effectiveness is tightly dependent on the specific task (e.g., [21, 22]).

A systematic analysis of a number of measures aimed at characterizing the difficulty of a *classification* problem is presented in [23]. In principle, this analysis could be applied also to text extraction problems, because such problems require classifying each individual character in a stream depending on whether the character is to be extracted. On the other hand, the cited work focuses on the geometrical properties of classification, considering measures that may highlight the separation between classes in the measurement space. Text extraction problems are generally not suitable to interpretations of this kind.

Performance prediction is an important research topic in *information retrieval*, aimed at assessing effectiveness

[1] A web based version is available on `http://regex.inginf.units.it/`; the source code is published on `https://github.com/MaLeLabTs/RegexGenerator`.

of a query before or during early stages of retrieval [24, 25, 26, 27]. Methods in this area generally require an indexing phase of the document corpus and then emit a prediction for a query based on a quick comparison between query terms and various indexed structures [28] (a corpus-independent approach is proposed in [29]).

As mentioned above, the effectiveness of a given string similarity metrics is usually highly dependent on the specific class of task. For this reason, we apply our proposal on a number of different metrics following an approach taken in other application domains. Several preprocessing strategies in combination with a variety of similarity metrics were assessed with reference to *ontology alignment* task [13]. The focus was finding the combination which exhibits best performance on a wide selection of problem instances representative of the ontology alignment task. A number of similarity metrics proposed by various research communities were applied to the task of matching entity names to database records [14]. The focus was finding the metric most suitable to the specific task. The key difference from our approach is that we investigate different string metrics as a tool for predicting the quality of a solution. The solution itself, i.e., the extractor tailored to a specific task instance, is built with a method which does not use string metrics in any way.

The availability of an estimate of costly data elaborations may be desirable also when dealing with data quality. For instance, the authors of [30] propose a method for estimating the number of duplicates in a dataset, before actually applying more complex specific duplicate detection algorithms. As in our case, a key requirement for the practicality of their proposal is that the estimation procedure has to run much faster than the actual algorithm.

3. Problem statement and motivations

3.1. Pattern-based entity extraction

The application problem consists in extracting entities that follow a syntactic pattern from a potentially large text. Extraction is performed by means of an *extractor* tailored to the specific pattern of interest. We consider a scenario in which the extractor is generated automatically by an *extraction inference engine*, based on examples of the desired behavior in the form of snippets to be extracted (i.e., the entities) and of snippets not to be extracted. Such examples usually consist of user-provided annotations on the text to be processed by the extractor.

A *snippet* X of a string s is a substring of s, identified by the starting and ending indexes in s. We denote by $\mathcal{X}$ the set of all snippets of string s. An *example* (s, X) is a string s associated with a (possibly empty) set of non-overlapping snippets $X \subset \mathcal{X}$. We do not make any assumption on the length or internal structure of string s, which may be, e.g., a text line, or an email message, or a collection of email messages, or a log file and so on. Set X represents all and only the *desired extractions* from s, i.e., snippets in $\mathcal{X} \setminus X$ are *not* to be extracted.

The extractor inference engine takes an example (s, X) as input and outputs an extractor e whose extraction behavior is consistent with the provided example—e should extract from s only the desired extractions X. Furthermore, e should capture the *pattern* describing the extractions, thereby *generalizing* beyond the actual examples. In other words, (s, X) constitutes an incomplete specification of the extraction behavior of an ideal and unknown extractor $e^\star$ and the extractor inference engine should aim at inferring an extractor with the same extraction behavior of $e^\star$.

To quantify the quality of a solution e, another example (s', X') is used such that both (s, X) and (s', X') represent the extraction behavior of $e^\star$. The extraction behavior of e is compared against that of $e^\star$ in terms of its F-measure (harmonic mean of precision and recall) on (s', X'): F-measure is 1 if and only if e extracts all and only the snippets X' from s'. We emphasize that (s, X) is the input of the extraction inference engine, that is, (s', X') is *not* required for actually generating e. We use (s', X') only for assessing the quality of a generated extractor.

Let $(s, X), (s', X')$ be a pair representing the extraction behavior of the target unknown extractor $e^\star$. We define the tuple (s, X, s') to be a *problem instance*. Let f' be the F-measure on (s', X') of the extractor e generated from (s, X) by the extraction inference engine. We define the tuple (s, X, s', X', f') to be a *solved problem instance*.

3.2. Effectiveness prediction

With reference to a pair $(s, X), (s', X')$ representing the extraction behavior of the target unknown extractor $e^\star$, let e denote the extractor generated by the extractor inference engine. The prediction problem consists in predicting F-measure f' of e on (s', X') based solely on (s, X, s')—that is, neither e nor X' are available for constructing the prediction. Prediction reliability may be measured in terms of the prediction error on a set of solved problem instances: we present the specific indexes that we used to this purpose in Section 5.2. In general, we are interested in minimizing the prediction error, i.e., the difference between the predicted value $\hat{f}'$ and the actual value f', which, clearly, is not available while computing the prediction.

This problem statement models the practical scenario in which the user has annotated some text for generating the extractor e and is interested in assessing the quality that will be obtained by applying e on a given unannotated text, *before* actually generating e.

In order to train the predictor, we assume that a set of solved problem instances are available. Note that knowledge of e for solved problem instances is not required.

Our overall framework do not make any assumptions on the implementation of the extractor inference engine or on the nature of the text extractor itself. However, our interest in this problem as well as the detailed experimental assessment in this work are based on evolutionary generation of extractors consisting of *regular expressions*.

In particular, we will consider solved problem instances obtained from the extraction inference engine in [9]. We chose this engine because it represents the state-of-the-art for the usage scenario considered in this paper. However, we further validate our framework by applying it also on a different extraction engine which generates extractors in the form of Deterministic Finite Automata (DFA) [15].

4. Our prediction method

Our prediction method consists of three steps. First, we transform the input (s, X, s') in an intermediate representation which is suitable to be processed using string similarities. Second, we extract a set of numerical features consisting in several statistics of similarities among strings of the intermediate representation. Finally, we apply a regressor to the vector of features and obtain an estimate $\hat{f}'$ of the F-measure f' which an extractor would have on X'.

In the following sections, we describe each step in detail. Figure 1 shows an overview of the prediction method.

4.1. Tokenization

We transform the input (s, X, s') into a triplet (P, N, U) whose elements are multisets of strings among which similarities can be computed. Multiset P (*positives*) includes all the snippets in X (i.e., all the desired extractions). Multisets N (*negatives*) and U (*unlabeled*) are obtained after splitting s and s' in *tokens*, according to the tokenization procedure described below. In particular, N includes all the tokens of s which do not overlaps snippets in X while U includes all the tokens of s'.

The aim of tokenization is to split strings in tokens whose length and content is "appropriate" with respect to the specific problem instance. To this end, we construct a set $\mathcal{S}$ of characters acting as token separators as follows. First, we construct the set of characters $\mathcal{S}_0$ including each character immediately preceding or immediately following each snippet in X. 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 $T_i \subset \mathcal{X}$ 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 snippets to be extracted is maximal, i.e., $\mathcal{S} := \mathcal{S}_{i^*}$, with $i^* = \arg\max_i |T_i \cap X|$—in case of tie, we choose the set $\mathcal{S}_i$ with smallest size.

Having determined the set of characters $\mathcal{S}$ acting as token separators, we split s and s' in tokens accordingly. Figure 2 shows an example of the tokenization procedure applied to a problem instance concerning the extraction of dates. In particular, Figure 2b shows the iterative procedure used to build the set of separators $\mathcal{S}$: in this case, $\mathcal{S}$ is assigned to $\mathcal{S}_2 = \{\text{␣}, ;\}$; the dot character is not considered as as a separator because in that case the snippet `11.10.2013` would be split.

$$
\begin{aligned}
s &= \texttt{The␣file␣has␣been␣sent␣on␣11.10.2013␣and␣}\\
&\quad\ \texttt{has␣been␣received␣on␣15-10-2013;␣the␣cont}\\
&\quad\ \texttt{ent␣has␣been␣written␣on␣7/2/2011.}\\
X &= \{\texttt{11.10.2013}, \texttt{15-10-2013}, \texttt{7-2-2011}\}\\
s' &= \texttt{Today␣is␣18-12-2015;␣nice!}
\end{aligned}
$$

(a) Problem instance (s, X, s').

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

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

$$
\begin{aligned}
P &= \{\texttt{11.10.2013}, \texttt{15-10-2013}, \texttt{7-2-2011}\}\\
N &= \{\texttt{The}, \texttt{file}, \texttt{has}, \ldots, \texttt{on}\}\\
U &= \{\texttt{Today}, \texttt{is}, \texttt{18-12-2015}, \texttt{nice!}\}
\end{aligned}
$$

(c) Tokenization outcome (P, N, U).

Figure 2: Tokenization example.

4.2. Features computation

Given a triplet (P, N, U) and a string similarity metric m (see next section), we want to obtain a set of numerical features which are relevant to characterize the problem instance difficulty, and hence affect the effectiveness of the extractor inference on that instance. As outlined in the introduction, the basic idea consists in computing some statistics from similarity measurements able to capture aggregate differences between strings to be extracted and strings not to be extracted. In particular, we should compute the similarity among the strings in P, N, and U; next across strings in P and N as well in P and U; finally, we could compute some statistics among all these computations. However, the size of the involved multisets (N and U in particular) would make an approach of this sort not feasible. Hence, we propose two different methods to drastically reduce the amount of similarity computations.

In the first method, which we call *Sample*, we proceed as follows. We construct a subset $N' \subset N$ by randomly sampling $|P|$ elements from N and a subset $U' \subset U$ by randomly sampling $|P|$ elements from U. Then, we compute the similarity values for all pairs of strings in $P \times P$, $P \times N'$, $P \times U'$, $N' \times N'$, and $U' \times U'$ and compute 6 statistics for each of the 5 sets of measurements: min, max, mean, median, 25th-percentile, and 75th-percentile—max is not taken into account for pairs of strings of the same set ($P \times P$, $N' \times N'$ and $U' \times U'$). We predict f' by using as features $|P|$, $|N|$, $|U|$ and all the previously computed figures—$3 + 6 \cdot 5 - 3 = 30$ features.

In the second method, which we call *Rep*, we build a set

Figure 1: An overview of our prediction method. The input is a problem instance and the output is an estimate $\hat{f}'$ of the F-measure on (s', X').

P' containing 3 representatives of the positives in P, as follows. We compute, for each $p \in P$, its average similarity to all the other positives (i.e., $\bar{m}(p) := \frac{1}{|P|}\sum_{p' \in P} m(p, p')$); then, we insert in P': (1) the positive with the lowest $\bar{m}$; (2) the positive with the greatest $\bar{m}$; and, (3) the positive with the $\bar{m}$ closest to $\frac{1}{|P|}\sum_{p \in P} \bar{m}(p)$ (i.e., the average value for the average similarity). Finally, we compute the similarity values for all pairs of strings in $P' \times P'$, $P' \times N$, $P' \times U$ and the same 6 statistics as above—max is not taken into account for $P' \times P'$. We predict f' by using as features $|P|$, $|N|$, $|U|$ and all the previously computed figures—$3 + 6 \cdot 3 - 1 = 20$ features.

4.2.1. String similarity metrics

Several different string similarity metrics exist. In order to limit the number of possible prediction method variants by considering only the more promising metrics, we referred to previous studies which compared string metrics [13, 31, 32] and we chose the most promising ones: Jaccard, Jaro, JaroWinkler, Levenshtein, NeedlemanWunsch, SmithWaterman, SoftTFIDF, UnsmoothedJS. We used the SecondString Java library [33] to actually implement the metric computation. We provide below a high-level overview of each metric and refer the reader to [14] and to the documentation of the SecondString library for full details.

The Jaccard similarity index between two strings a and b is computed by considering a string as a set of characters or bigrams (as in our case) and is the ratio between the intersection and the union of the two sets a, b. The Jaro similarity index takes into account *matching* characters, i.e., characters appearing in both a and b 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. The JaroWinkler index is a modified Jaro index in which similarity grows for strings that share a common prefix. Levenshtein takes into account the minimum number of single-character edits required to make a and b identical, i.e., it is a form of edit distance. NeedlemanWunsch is a form of edit distance which assigns different costs to edit operations (we used the standard cost configuration of the SecondString library). SmithWaterman is a variant of NeedlemanWunsch assigning lower costs to sequences of insertions or deletions. SoftTFIDF is a form of *cosine similarity* weighing substrings appearing in both a and b and substrings of either string for which a substring of the other exists that is sufficiently similar according to the JaroWinkler metric. Finally, UnsmoothedJS (Jensen-Shannon) is a similarity index taking into account the loss of information when representing either string with the other, where the loss of information is quantified by the Kullback-Leibler divergence.

4.3. Regression

We explored three different options: a linear model (LM), random forests (RF) regression, and support vector machines (SVM) regression. In particular, we used the gaussian kernel and $C = 1$ for SVM [34], and we used the algorithm proposed in [35] with $n_{\text{tree}} = 500$ for RF. In all cases we set the actual predicted value to 1 if the model output is larger than 1 and to 0 if it is lower than 0—f' values are intrinsically defined in $[0, 1]$.

5. Experimental evaluation

We constructed and assessed experimentally all the 48 prediction model variants resulting from the combination of: 2 feature set construction methods (Sample and Rep, Section 4.2); 8 string similarity metrics (Section 4.2.1); 3 regressors (LM, RF, and SVM, Section 4.3). We trained each model variant with a set of solved problem instances $\mathcal{E}_{\text{train}}$ and assessed the resulting predictor on a set of solved problem instances disjoint from $\mathcal{E}_{\text{train}}$, as detailed in the next sections.

5.1. Data

We consider 19 challenging *extraction tasks* built over a text corpus fully annotated with the entities to be extracted. We use a selection of the tasks used in [9], summarized in Table 1. The name of each extraction task is composed of the name of the corpus followed by the name of the entity type to be extracted. Entity names should be self-explanatory: Username corresponds to extracting only the username from Twitter citations (e.g., only `MaleLabTs` instead of `@MaleLabTs`); Email-ForTo corresponds to extracting email addresses appearing after the strings `for:` or `to:` (possibly capitalized). Names ending with a $*$ suffix indicate extraction tasks with *context*. These are extraction tasks in which a text snippet must or must not be extracted depending on the text surrounding the snippet—e.g., an email address might have to be extracted only when following a `Reply-To:` header name.

Each extraction task consists of a string s_0 annotated with all the desired extractions. Table 1 shows, for each extraction task, the length $\ell(s_0)$ of the string s_0 (in thousands of characters) and the number $|X_0|$ of snippets corresponding to entities to be extracted. The construction of problem instances from extraction tasks is described in the next section. The table shows also the average F-measure

obtained by the approach of [9] on those tasks see next section for more details) and the average time taken to learn the extractor.

5.2. Experimental procedure

We aimed at investigating the prediction effectiveness at varying difficulty of extraction, in particular concerning: (a) the amount of data available for learning; (b) the complexity of the pattern of the involved entity; and (c) the degree of representativeness of the learning data w.r.t. all the other data. To this end, we built a number of different solved problem instances (s, X, s', X', f') from our 19 extraction tasks, as follows.

Concerning the amount of data available for learning, we considered three values for the number n_X of snippets to be extracted, $\{25, 50, 100\}$. Then, for each extraction task and each n_X value, we built 5 solved problem instances (s, X, s', X', f') (*repetitions*): (i) we randomly chose a substring s of s_0 containing n_X snippets to be extracted; (ii) we generated an extractor from (s, X) with the method in [9]; (iii) we randomly chose a substring s' of s_0 non-overlapping s and such that $|X'| = 500$; (iv) we assessed the f-measure f' of the generated extractor on (s', X'). Concerning step ii, we used the tool available at `https://github.com/MaLeLabTs/RegexGenerator` and described in [9, 6]: the tool generates a regular expression which aims at extracting from s all and only the snippets in X while trying to generalize the learning data. The regular expression is generated by means of an evolutionary procedure which searches the space of the regular expressions driven by a multiobjective optimization strategy—we refer the reader to the cited paper for full details. Table 1 shows the average value of f'—i.e., the F-measure on (s', X') obtained by the regular expression generated by the cited tool from the examples in (s, X)—over the 5 repetitions for each task and each value of n_X.

Next, we used the 285 solved problem instances for assessing our 48 model variants. In particular, we executed a 5-fold cross validation of the behavior of each model variant for each pair extraction task and n_X value. That is, we trained each model variant on 4 of the 5 repetitions and assessed the prediction on the remaining one. The rationale for partitioning the dataset of solved problem instances based on repetitions is the need of ensuring the presence in the training data of at least one problem instance for each extraction task. Such a partitioning corresponds to the scenario in which the data available for calibrating the prediction method are representative of a wide range of different extraction tasks. That scenario could be implemented in a real setting easily, as it would suffice to retrain the predictor after each new run of the engine. We remark, however, that *none* of the problem instances used for assessing the prediction method was available in the training phase. Later, in Section 5.3, we analyze the much more challenging scenario in which a novel extraction task arises.

For assessing the predictor, we computed the following indexes:

- Mean absolute error (MAE), which measures the average value of the absolute difference between the predicted value $\hat{f}'$ and the actual value f':

$$\text{MAE} = \frac{1}{|\mathcal{E}|} \sum_{\mathcal{E}} \left| \hat{f}' - f' \right|$$

  where $\mathcal{E}$ is the set of solved problem instances on which the predictor is assessed.

- Relative mean absolute error (RMAE), which is the average value of the ratio between the absolute error and the actual value:

$$\text{RMAE} = \frac{1}{|\mathcal{E}|} \sum_{\mathcal{E}} \left| \frac{\hat{f}' - f'}{f'} \right|$$

- Performance-class accuracy (CA), which measures accuracy on a classification task in which problem instances are partitioned in 3 difficulty classes, easy, medium, and hard, corresponding to values of f' in the intervals $]0.95, 1]$, $]0.8, 0.95]$, and $[0, 0.8]$, respectively. In our setting, this partitioning corresponds to roughly 32%, 29%, and 39% problem instances in the respective performance classes.

$$\text{CA} = \frac{\left| \left\{ C(f') = C(\hat{f}') \right\} \right|}{|\mathcal{E}|}$$

  where $C : [0, 1] \rightarrow \{]0.95, 1],]0.8, 0.95], [0, 0.8]\}$ is the function which assigns the performance-class to a given value of F-measure. In other words, CA is the ratio between the number of instances for which the predicted value $\hat{f}'$ and the actual value f' belong to the same performance-class and the number of all instances.

MAE and RMAE are indexes commonly used for assessing predictors of continuous values; we defined the latter index (CA) in order to capture the ability of the proposed predictor in providing a coarser indication of the difficulty of a problem instance.

5.3. Results and discussion

Table 2 shows the values of MAE, RMAE, and CA for all the 48 model variants, averaged over all prediction problems. The table provides the values of the three indexes computed on the problem instances which has not been used for training the model (denoted by $\mathcal{E}_{\text{validate}}$); for completeness of analysis, the table also shows indexes values on the solved problem instances used for training the model (denoted by $\mathcal{E}_{\text{train}}$).

Since we could not identify any baseline method from the literature, we chose to use the following prediction as baseline method. Concerning f', the predicted value is

Table 1: Salient information about the 19 extraction tasks. The length $\ell(s_0)$ of the string s_0 is expressed in thousands of characters. The four rightmost columns show average values for f' for different values of the number of snippets to be extracted n_X (columns 4–6) and the average time t_l (in min) taken by the inference engine to learn an extractor for $n_X = 50$.

			f'			
Extraction task	$\ell(s_0)$	$\lvert X_0\rvert$	25	50	100	t_l
BibTeX/Author*	54	589	0.80	0.85	0.85	20
BibTeX/Title*	54	200	0.69	0.70	0.73	141
Cetinkaya-HTML/href	154	214	0.98	0.98	0.99	26
Cetinkaya-HTML/href-Content*	154	214	0.74	0.73	0.80	29
Cetinkaya-Text/All-URL	39	168	0.99	0.99	0.99	8
CongressBills/Date	16 511	3085	0.42	0.63	0.64	584
Email-Headers/Email	261	1244	0.64	0.64	0.73	224
Email-Headers/Email-To-For*	261	331	0.61	0.56	0.78	398
Email-Headers/IP	261	848	0.86	0.86	0.91	89
Log/IP	4126	75 958	1.00	1.00	1.00	2
Log/MAC	4126	38 812	1.00	1.00	1.00	3
NoProfit-HTML/Email	860	1094	0.86	1.00	1.00	3
Reference/First-Author*	30	198	0.97	0.97	1.00	26
ReLIE-Email/Phone-Number	8805	5184	0.79	0.80	0.78	16
ReLIE-HTML/All-URL	4240	502	0.88	0.92	0.95	35
ReLIE-HTML/HTTP-URL	4240	499	0.88	0.91	0.92	32
Twitter/All-URL	4344	14 628	0.98	0.98	0.98	8
Twitter/Hashtag+Citation	4344	56 994	1.00	1.00	1.00	4
Twitter/Username*	4344	42 352	1.00	1.00	1.00	2

the average value across all the solved problem instances in $\mathcal{E}_{\text{train}}$; concerning the performance-class, the predicted value is the most occurring class in $\mathcal{E}_{\text{train}}$.

The table shows also the average time t_t for training the predictor on $\mathcal{E}_{\text{train}}$ and the average time t_p for computing features for a single problem instance, both expressed in ms. All the experiments have been carried out on a workstation equipped with 8 GB and a Intel Core2 Quad CPU 2.5 GHz. It can be seen that a prediction can be obtained, in many cases, in less than a second: for reference, the inference of a regular expression by means of the method of [9] for the same tasks requires much longer times—ranging from ≈ 2 min to ≈ 500 min (see rightmost column of Table 1).

By analyzing the experimental results of Table 2, several considerations can be made.

It can be seen that the best prediction is quite good: RMAE = 9.1% for the RF-Sample-NeedlemanWunsch combination. This value corresponds to an average absolute error MAE = 0.056. Moreover, RMAE $\leq 10\%$ for 6 combinations, all based on RF and evenly distributed between Sample and Rep. To place this figure in perspective, we observe that the RMAE for the baseline predictor obtained 19.8%—that is, our prediction methodology halves the error w.r.t. the baseline. In order to investigate if the effectiveness is limited to the specific inference engine here considered, we also applied it to a different engine. The detailed results are presented at the end of this section: in brief, they confirm that that our methodology outperforms the baseline.

Quality of the prediction is good also when assessed in terms of accuracy of the performance-class prediction (CA): 80.9% for the RF-Sample-NeedlemanWunsch combination. Moreover, CA $\geq 75\%$ for 12 combinations, mostly based on RF-Sample. To place this figure in perspective, we observe that the CA for baseline class predictor is 33.8%.

Prediction based on the Sample method for feature construction tends to be more effective than prediction based on the Rep method. We speculate that choosing a few representative positives may succeed to capture diversity between positives, but fails at capturing the overall variability of the text which has to be processed, which is what actually matter in making a problem more or less hard to solve. It can also be observed that computing features takes in general longer for Rep.

With respect to the prediction model, RF appears to be the best performing choice and obtains in general better figures for all the indexes. Moreover, it appears to be more robust to the other factors involved—metric and feature computation method.

Concerning the string similarity metrics our experiments suggest that NeedlemanWunsch is the best one. Good results can be obtained with Levenshtein and SmithWaterman metrics as well, however: this finding is not surprising because these 3 metrics are closely related (see Section 4.2.1).

Depending on the specific scenario in which a prediction should be provided, efficiency could play a role almost as important as effectiveness: a faster and slightly less accu-

Table 2: Results with the 5-fold cross validation on repetitions.

		f' on $\mathcal{E}_{\text{train}}$			f' on $\mathcal{E}_{\text{validate}}$			t_t	t_p
	Metric m	MAE	RMAE	CA	MAE	RMAE	CA	[ms]	[ms]
RF-Sample	Jaccard	0.026	4.2	89.5	0.062	10.0	78.0	1053	585
	Jaro	0.026	4.3	89.0	0.065	10.5	74.9	856	49
	JaroWinkler	0.027	4.4	87.8	0.066	10.5	74.2	854	49
	Levenstein	0.024	4.0	90.1	0.059	9.6	79.8	999	492
	NeedlemanWunsch	0.023	3.8	89.8	0.056	9.1	80.9	985	487
	SmithWaterman	0.025	4.0	87.8	0.058	9.4	78.8	838	559
	SoftTFIDF	0.029	4.6	88.6	0.066	10.6	76.0	724	227
	UnsmoothedJS	0.029	4.7	87.3	0.065	10.5	72.8	729	88
RF-Rep	Jaccard	0.027	4.4	89.1	0.064	10.3	77.4	742	6072
	Jaro	0.027	4.4	87.4	0.064	10.3	73.9	650	324
	JaroWinkler	0.028	4.5	86.1	0.064	10.2	76.0	643	320
	Levenstein	0.026	4.2	88.6	0.061	9.8	74.6	699	2245
	NeedlemanWunsch	0.026	4.2	89.0	0.062	9.9	75.6	711	2247
	SmithWaterman	0.026	4.3	87.8	0.058	9.5	78.8	636	2329
	SoftTFIDF	0.032	5.1	87.5	0.070	11.1	69.3	570	1425
	UnsmoothedJS	0.034	5.4	85.6	0.073	11.4	66.4	561	862
SVM-Sample	Jaccard	0.050	8.5	85.2	0.064	10.6	75.9	59	585
	Jaro	0.089	11.0	9.2	0.144	20.0	21.6	46	49
	JaroWinkler	0.089	11.0	9.2	0.144	20.0	21.6	46	49
	Levenstein	0.053	8.9	79.1	0.071	11.6	72.8	59	492
	NeedlemanWunsch	0.050	8.4	83.0	0.066	10.7	78.1	56	487
	SmithWaterman	0.089	10.8	8.5	0.144	20.1	21.2	51	559
	SoftTFIDF	0.089	11.0	9.2	0.144	20.0	21.6	43	227
	UnsmoothedJS	0.089	11.0	9.2	0.144	20.0	21.6	45	88
SVM-Rep	Jaccard	0.052	8.9	80.7	0.065	10.7	73.9	49	6072
	Jaro	0.089	11.0	8.8	0.144	20.0	21.6	37	324
	JaroWinkler	0.089	11.0	8.8	0.144	20.0	21.6	40	320
	Levenstein	0.058	9.9	77.2	0.066	11.1	71.4	46	2245
	NeedlemanWunsch	0.058	9.9	77.2	0.066	11.1	71.4	49	2247
	SmithWaterman	0.089	10.8	9.1	0.143	20.0	21.2	44	2329
	SoftTFIDF	0.089	11.0	8.8	0.144	20.0	21.6	36	1425
	UnsmoothedJS	0.089	11.0	8.9	0.144	20.0	21.6	36	862
LM-Sample	Jaccard	0.075	11.1	66.0	0.085	12.7	62.1	17	585
	Jaro	0.081	11.9	53.0	0.089	13.1	50.2	15	49
	JaroWinkler	0.087	12.7	50.2	0.094	13.7	44.5	15	49
	Levenstein	0.082	12.0	57.3	0.096	14.2	54.4	16	492
	NeedlemanWunsch	0.073	10.5	58.0	0.091	12.9	52.7	17	487
	SmithWaterman	0.082	11.8	51.4	0.091	13.3	53.8	15	559
	SoftTFIDF	0.082	11.8	52.7	0.088	12.9	48.8	15	227
	UnsmoothedJS	0.084	12.2	51.9	0.089	13.2	49.1	14	88
LM-Rep	Jaccard	0.079	11.8	59.7	0.083	12.5	57.2	11	6072
	Jaro	0.087	12.7	51.0	0.091	13.3	50.5	12	324
	JaroWinkler	0.084	12.4	55.5	0.091	13.3	51.6	11	320
	Levenstein	0.083	12.0	56.4	0.096	13.7	53.4	10	2245
	NeedlemanWunsch	0.083	12.0	56.4	0.096	13.7	53.4	10	2247
	SmithWaterman	0.087	12.7	53.6	0.094	13.8	52.3	11	2329
	SoftTFIDF	0.093	13.6	47.5	0.096	14.2	45.9	11	1425
	UnsmoothedJS	0.090	13.2	49.6	0.093	13.7	47.0	11	862
Baseline		0.135	19.8	53.0	0.135	19.8	53.0		

rate prediction may be more useful than a longer and more accurate one. In this sense, we observe that RF-Sample-Jaro allows computing features in just a tenth of the time required by RF-Sample-NeedlemanWunsch, with a moderate decrease in CA: 74.9%, roughly −6% less than the best combination. In fact, the time for obtaining a prediction mostly consists in the time taken to compute the features, since the regressor application time is negligible—t_t is the time taken to *train* rather than to apply the regressor.

Table 3 shows the results of the best performing method (i.e., RF-Sample-NeedlemanWunsch) for each considered extraction task: besides MAE, RMAE, and CA, the table also shows the mean and standard deviation of the actual (f') and predicted ($\hat{f}'$) values of the F-measure obtained for each task. It can be seen that for the majority of the tasks, the prediction is indeed accurate, being RMAE $\leq 5\%$. On the other hand, there are 3 on 19 tasks for which RMAE is greater than 20%: two of them, including the one for which the predictor is less effective, are tasks with context (see Section 5.1). Indeed, our prediction methodology bases only on the similarities between snippets to be extracted and snippets not to be extracted—the context around snippets is not taken into account. In this respect, we manually inspected several snippets in the tasks Email-Headers/Email-To-For* and Cetinkaya-HTML/href-Content* and found that (i) snippets are in general longer than other tasks and (ii) some snippets to be extracted are similar to some snippets not to be extracted (e.g., the same email address can be present in both categories in Email-Headers/Email-To-For*): these factors can indeed make the string similarity-based prediction harder.

Concerning the performance-class accuracy CA, it can be observed that, in addition to the two tasks mentioned above, the figure is unsatisfactory also for ReLIE-Email/Phone-Number (46.7%), for which, instead, RMAE is lower than the average. We believe that this result is an artifact deriving from the choice of the performance-class intervals and the specific f' values observed for that task—they lie right around 0.8, which is the bound between two performance classes (see Table 1).

In order to investigate on which features are most relevant for the prediction, we performed a feature ablation procedure. We focused on the RF-Sample-NeedlemanWunsch variant (i.e., the best one) and repeated the experiment several times by removing one by one each feature. Table 4 shows the results (MAE, RMAE, and CA) for each removed feature: the rows are sorted according to decreasing RMAE—i.e., the feature whose removal causes the greatest drop in RMAE comes first. Despite the figures do not allow to draw sharp conclusions, it can be seen that $|U|$ and $|P|$ appear to play an important role in the prediction, followed by the median and mean of similarities among negatives and among positives, respectively. From another point of view, all the statistics about similarities appear to capture the overall difficulty of the extraction task. Moreover, we interpret the importance of $|U|$ and $|P|$ as a measure of how "big" is the unknown data w.r.t. the data which was available for learning—a respect which impacts the *representativeness* of learning data for the chosen task, which is a crucial factor in all machine learning applications.

Table 4: Results of RF-Sample-NeedlemanWunsch on $\mathcal{E}_{\text{validate}}$ according to the feature ablation procedure.

Removed feature	MAE	RMAE	CA
$\lvert U\rvert$	0.058	9.3	81.6
$\lvert P\rvert$	0.057	9.3	80.9
$\text{median}_{N'\times N'}$	0.057	9.2	79.8
$\text{mean}_{P\times P}$	0.057	9.2	81.3
$\text{min}_{P\times P}$	0.057	9.2	82.0
$\text{max}_{P\times N'}$	0.057	9.2	80.9
$\text{max}_{P\times U'}$	0.057	9.2	80.2
$\text{75th-percentile}_{U'\times U'}$	0.057	9.2	81.2
$\text{median}_{P\times U'}$	0.056	9.2	80.2
$\text{75th-percentile}_{P\times N'}$	0.057	9.2	80.9
$\text{median}_{U'\times U'}$	0.056	9.2	80.9
$\text{75th-percentile}_{P\times P}$	0.056	9.2	81.2
$\lvert N\rvert$	0.057	9.2	80.2
$\text{25th-percentile}_{P\times U'}$	0.056	9.2	80.9
$\text{25th-percentile}_{P\times P}$	0.056	9.2	80.2
$\text{25th-percentile}_{N'\times N'}$	0.056	9.2	81.2
$\text{25th-percentile}_{P\times N'}$	0.056	9.1	81.3
$\text{75th-percentile}_{P\times U'}$	0.056	9.1	80.5
$\text{75th-percentile}_{N'\times N'}$	0.056	9.1	80.2
$\text{min}_{P\times U'}$	0.056	9.1	80.9
$\text{median}_{P\times N'}$	0.056	9.1	80.9
$\text{min}_{U'\times U'}$	0.056	9.1	81.2
$\text{mean}_{N'\times N'}$	0.056	9.1	80.2
$\text{min}_{N'\times N'}$	0.056	9.1	81.3
$\text{min}_{P\times N'}$	0.056	9.1	79.8
$\text{mean}_{U'\times U'}$	0.056	9.1	81.6
$\text{mean}_{P\times N'}$	0.056	9.1	81.2
$\text{mean}_{P\times U'}$	0.056	9.0	81.2
$\text{25th-percentile}_{U'\times U'}$	0.056	9.0	82.0
$\text{median}_{P\times P}$	0.055	9.0	80.9
No removals	0.056	9.1	80.9

In order to gain further insights on our proposal, we performed two additional experimental campaigns aimed at (i) assessing the method in the challenging scenario where a novel extraction task arises; and, (ii) validating the method when applied to a different extractor inference technique.

Concerning the former aim, we considered the best variant (RF-Sample-NeedlemanWunsch) and modified the experimental procedure (Section 5.2) as follows: instead of executing a 5-fold cross validation on the repetition number, we executed a 19-fold cross validation on the extraction task. That is, for each extraction task, we trained our prediction model on the 270 solved problem instances of the other tasks and assessed the resulting predictor on

Table 3: Results of RF-Sample-NeedlemanWunsch for f' on $\mathcal{E}_{\text{validate}}$ for different tasks.

Extraction task	MAE	RMAE	CA	f' avg	f' sd	$\hat{f}'$ avg	$\hat{f}'$ sd
BibTeX/Author*	0.060	7.4	73.3	0.837	0.026	0.829	0.021
BibTeX/Title*	0.079	12.2	86.7	0.705	0.095	0.716	0.014
Cetinkaya-HTML/href	0.022	2.3	86.7	0.984	0.007	0.964	0.010
Cetinkaya-HTML/href-Cont.*	0.230	37.9	40.0	0.757	0.189	0.759	0.035
Cetinkaya-Text/All-URL	0.012	1.2	100.0	0.991	0.006	0.981	0.003
CongressBills/Date	0.082	18.6	100.0	0.563	0.053	0.594	0.014
Email-Headers/Email	0.091	17.6	93.3	0.671	0.077	0.738	0.019
Email-Headers/Email-To-For*	0.113	23.9	86.7	0.649	0.068	0.677	0.041
Email-Headers/IP	0.066	8.1	86.7	0.878	0.050	0.880	0.016
Log/IP	0.014	1.4	100.0	1.000	0.000	0.987	0.005
Log/MAC	0.005	0.5	100.0	0.999	0.001	0.996	0.002
NoProfit-HTML/Email	0.104	20.9	33.3	0.952	0.107	0.924	0.038
References/First-Author*	0.016	1.6	100.0	0.978	0.009	0.979	0.003
ReLIE-Email/Phone-Number	0.055	6.9	46.7	0.788	0.051	0.786	0.005
ReLIE-HTML/All-URL	0.029	3.1	60.0	0.917	0.031	0.898	0.027
ReLIE-HTML/HTTP-URL	0.043	4.9	80.0	0.901	0.042	0.903	0.020
Twitter/All-URL	0.007	0.7	93.3	0.981	0.003	0.975	0.004
Twitter/Hashtag+Citation	0.014	1.4	93.3	0.999	0.001	0.984	0.018
Twitter/Username*	0.013	1.3	100.0	1.000	0.000	0.985	0.017
Average	0.056	9.1	80.9	0.871	0.043	0.871	0.016

the 15 instances of that task. Table 5 presents the results with the same structure of Table 3. It can be seen that, for the majority of the tasks, the RMAE still remains lower than 10%; moreover, for 6 tasks, the performance-class accuracy is larger than 66%. On average, as expected, the prediction is less reliable than in the scenario where the predictor is trained on problem instances from all the extraction tasks: MAE and RMAE roughly double, whereas CA roughly halves.

Finally, we applied our proposal (in the RF-Sample-NeedlemanWunsch variant) also to another inference engine which generates extractors in the form of Deterministic Finite Automata (DFA) [15]. We built the engine by implementing the method described in the cited paper and applied it to a selection of 6 extraction tasks: Cetinkaya-HTML/href, Cetinkaya-Text/All-URL, CongressBills/Date, ReLIE-Email/Phone-Number, ReLIE-HTML/All-URL, and Twitter/Hashtag+Citation—we hence obtained 90 solved problem instances. In this case the inference engine produces DFAs which tend to be much less effective than the extractors considered in the previous experiments, i.e., those generated by the engine in [9]. We thus redefined the performance classes as $]0.25, 1]$, $]0.15, 0.25]$, and $[0, 0.15]$, which roughly correspond to 20%, 45%, and 35% problem instances, respectively. We executed the same 5-fold cross validation described in Section 5.2—i.e., the predictor is trained on 4 repetitions and assessed on the remaining one—and obtained 0.065, 57.2, and 55.9 for MAE, RMAE, and CA, respectively. We verified that our predictor scored better than the baseline (+10% for CA). On the other hand, the figures are in general worse than those obtained for the inference engine of [9]. We speculate that this difference is mostly motivated by the fact that the effectiveness of [15] is poor in general and appears to be only slightly affected by the specific problem instance.

6. Concluding remarks

We have considered a scenario in which an extraction inference engine generates an extractor automatically from user-provided examples of the entities to be extracted from a dataset. We have addressed the problem of predicting the accuracy of the extractor that may be inferred from the available examples, by requiring that the prediction be obtained very quickly w.r.t. the time required for actually inferring the extractor. This problem is highly challenging and we are not aware of any earlier proposal in this respect. With reference to extractors consisting of regular expressions, we have proposed several techniques and analyzed them experimentally in depth. The results suggest that reliable predictions for tasks of practical complexity may indeed be obtained quickly and without actually generating the extractor.

Acknowledgements

The authors are grateful to the anonymous reviewers for their constructive comments.

Table 5: Results, in the novel extraction task scenario, of RF-Sample-NeedlemanWunsch for f' on $\mathcal{E}_{\text{validate}}$ for different tasks.

Extraction task	MAE	RMAE	CA	f' avg	f' sd	$\hat{f}'$ avg	$\hat{f}'$ sd
BibTex-Author*	0.066	8.0	53.3	0.837	0.067	0.811	0.043
BibTex-Title*	0.189	29.8	6.7	0.705	0.115	0.871	0.012
Cetinkaya-HTML/href	0.134	13.6	6.7	0.984	0.020	0.861	0.021
Cetinkaya-HTML/href-Cont.*	0.218	36.6	53.3	0.757	0.253	0.760	0.014
Cetinkaya-Text/All-URL	0.133	13.4	0.0	0.991	0.006	0.866	0.039
CongressBills-Date	0.239	51.1	40.0	0.563	0.122	0.804	0.013
Email-Headers/Email	0.134	25.6	69.2	0.672	0.136	0.795	0.061
Email-Headers/Email-To-For*	0.136	28.7	80.0	0.649	0.151	0.746	0.037
Email-Headers/IP	0.076	9.3	66.7	0.878	0.075	0.902	0.054
Log/IP	0.067	6.7	20.0	1.000	0.000	0.926	0.013
Log/MAC	0.214	21.4	0.0	0.999	0.003	0.800	0.010
NoProfit-HTML/Email	0.365	41.9	6.7	0.952	0.186	0.629	0.034
References/First-Author*	0.040	4.1	60.0	0.978	0.021	0.963	0.033
ReLIE-Email/Phone-Number	0.063	8.4	60.0	0.788	0.063	0.827	0.069
ReLIE-HTML/All-URL	0.039	4.2	60.0	0.917	0.050	0.886	0.032
ReLIE-HTML/HTTP-URL	0.044	5.0	73.3	0.901	0.051	0.909	0.033
Twitter/All-URL	0.093	9.5	0.0	0.981	0.004	0.889	0.006
Twitter/Hashtag+Citation	0.025	2.5	73.3	0.999	0.003	0.972	0.027
Twitter/Username*	0.025	2.5	100.0	1.000	0.000	0.976	0.012
Average	0.121	17.0	43.6				

References

[1] S. N. Group, TokensRegex, http://nlp.stanford.edu/software/tokensregex.shtml (2011).

[2] A. Project, UIMA-Ruta rule-based text annotation, https://uima.apache.org/ruta.html.

[3] P. Kluegl, M. Toepfer, P.-D. Beck, G. Fette, F. Puppe, Uima ruta: Rapid development of rule-based information extraction applications, Natural Language Engineering FirstView (2015) 1–40. doi:10.1017/S1351324914000114.
URL http://journals.cambridge.org/article_S1351324914000114

[4] A. Bartoli, G. Davanzo, A. De Lorenzo, E. Medvet, E. Sorio, Data quality challenge: Toward a tool for string processing by examples, Journal of Data and Information Quality (JDIQ).

[5] A. Bartoli, A. De Lorenzo, E. Medvet, F. Tarlao, Learning text patterns using separate-and-conquer genetic programming, in: 18-th European Conference on Genetic Programming, Springer, 2015, pp. 16–27.

[6] A. Bartoli, A. De Lorenzo, E. Medvet, F. Tarlao, Inference of regular expressions for text extraction from examples, Knowledge and Data Engineering, IEEE Transactions on PP (99) (2016) 1–14. doi:10.1109/TKDE.2016.2515587.

[7] R. A. Cochran, L. D'Antoni, B. Livshits, D. Molnar, M. Veanes, Program boosting: Program synthesis via crowd-sourcing, in: Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '15, ACM, New York, NY, USA, 2015, pp. 677–688. doi:10.1145/2676726.2676973.
URL http://doi.acm.org/10.1145/2676726.2676973

[8] V. Le, S. Gulwani, Flashextract: A framework for data extraction by examples, in: ACM SIGPLAN Notices, Vol. 49, ACM, 2014, pp. 542–553.

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

[10] K. Davydov, A. Rostamizadeh, Smart autofill - harnessing the predictive power of machine learning in google sheets, http://googleresearch.blogspot.it/2014/10/smart-autofill-harnessing-predictive.html (Oct. 2014).

[11] F. Brauer, R. Rieger, A. Mocan, W. M. Barczynski, Enabling information extraction by inference of regular expressions from sample entities, in: Proceedings of the 20th ACM International Conference on Information and Knowledge Management, CIKM '11, ACM, New York, NY, USA, 2011, pp. 1285–1294. doi:10.1145/2063576.2063763.
URL http://doi.acm.org/10.1145/2063576.2063763

[12] Y. Li, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, H. V. Jagadish, Regular expression learning for information extraction, in: Proceedings of the Conference on Empirical Methods in Natural Language Processing, EMNLP '08, Association for Computational Linguistics, Stroudsburg, PA, USA, 2008, pp. 21–30.
URL http://dl.acm.org/citation.cfm?id=1613715.1613719

[13] M. Cheatham, P. Hitzler, String similarity metrics for ontology alignment, in: The Semantic Web–ISWC 2013, Springer, 2013, pp. 294–309.

[14] W. R. Cohen, P. Fienberg, A comparison of string distance metrics for name-matching tasks, in: Proceedings of the IJCAI-03 Workshop on Information Integration on the Web.

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

[16] K. Smith-Miles, L. Lopes, Measuring instance difficulty for combinatorial optimization problems, Computers & Operations Research 39 (5) (2012) 875–889.

[17] J. Pihera, N. Musliu, Application of machine learning to algorithm selection for tsp, in: Tools with Artificial Intelligence (ICTAI), 2014 IEEE 26th International Conference on, IEEE, 2014, pp. 47–54.

[18] A. Liefooghe, S. Verel, F. Daolio, H. Aguirre, K. Tanaka, A feature-based performance analysis in evolutionary multiobjective optimization, in: Evolutionary Multi-Criterion Optimization, Springer, 2015, pp. 95–109.

[19] J. He, C. Reeves, C. Witt, X. Yao, A note on problem difficulty measures in black-box optimization: Classification, realizations and predictability, Evolutionary Computation 15 (4) (2007) 435–443.
[20] M. Graff, R. Poli, J. J. Flores, Models of performance of evolutionary program induction algorithms based on indicators of problem difficulty, Evolutionary computation 21 (4) (2013) 533–560.
[21] H. Lee, R. T. Ng, K. Shim, Extending q-grams to estimate selectivity of string matching with low edit distance, in: Proceedings of the 33rd International Conference on Very Large Data Bases, VLDB '07, VLDB Endowment, 2007, pp. 195–206.
URL http://dl.acm.org/citation.cfm?id=1325851.1325877
[22] X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, E. Keogh, Experimental comparison of representation methods and distance measures for time series data, Data Mining and Knowledge Discovery 26 (2) (2013) 275–309.
[23] T. K. Ho, M. Basu, Complexity measures of supervised classification problems, Pattern Analysis and Machine Intelligence, IEEE Transactions on 24 (3) (2002) 289–300.
[24] J. A. Rodriguez Perez, J. M. Jose, Predicting query performance in microblog retrieval, in: Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval, ACM, 2014, pp. 1183–1186.
[25] H. Raviv, O. Kurland, D. Carmel, Query performance prediction for entity retrieval, in: Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval, ACM, 2014, pp. 1099–1102.
[26] L. Goeuriot, L. Kelly, J. Leveling, An analysis of query difficulty for information retrieval in the medical domain, in: Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval, ACM, 2014, pp. 1007–1010.
[27] C. Liu, J. Liu, N. J. Belkin, Predicting search task difficulty at different search stages, in: Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, ACM, 2014, pp. 569–578.
[28] C. Hauff, D. Hiemstra, F. de Jong, A survey of pre-retrieval query performance predictors, in: Proceedings of the 17th ACM conference on Information and knowledge management, ACM, 2008, pp. 1419–1420.
[29] G. Katz, A. Shtock, O. Kurland, B. Shapira, L. Rokach, Wikipedia-based query performance prediction, in: Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval, ACM, 2014, pp. 1235–1238.
[30] A. Heise, G. Kasneci, F. Naumann, Estimating the number and sizes of fuzzy-duplicate clusters, in: Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, ACM, 2014, pp. 959–968.
[31] W. Cohen, P. Ravikumar, S. Fienberg, A comparison of string metrics for matching names and records, in: Kdd workshop on data cleaning and object consolidation, Vol. 3, 2003, pp. 73–78.
[32] M. Bilenko, R. Mooney, W. Cohen, P. Ravikumar, S. Fienberg, Adaptive name matching in information integration, IEEE Intelligent Systems (5) (2003) 16–23.
[33] W. Cohen, P. Ravikumar, S. Fienberg, Secondstring: An open source java toolkit of approximate string-matching techniques, http://secondstring.sourceforge.net (2003).
[34] C.-C. Chang, C.-J. Lin, Libsvm: a library for support vector machines, ACM Transactions on Intelligent Systems and Technology (TIST) 2 (3) (2011) 27.
[35] L. Breiman, Random forests, Machine learning 45 (1) (2001) 5–32.