TuneR: Fine Tuning of Rule-based Entity Matchers

Matteo Paganelli
DIEF-UNIMORE
Modena, Italy
matteo.paganelli@unimore.it

Paolo Sottovia
DISI-UNITN
Trento, Italy
paolo.sottovia@unitn.it

Francesco Guerra
DIEF-UNIMORE
Modena, Italy
francesco.guerra@unimore.it

Yannis Velegrakis
Utrecht University
i.velegrakis@uu.nl

ABSTRACT

A rule-based entity matching task requires the definition of an effective set of rules, which is a time-consuming and error-prone process. The typical approach adopted for its resolution is a trial and error method, where the rules are incrementally added and modified until satisfactory results are obtained. This approach requires significant human intervention, since a typical dataset needs the definition of a large number of rules and possible interconnections that cannot be manually managed. In this paper, we propose TuneR, a software library supporting developers (i.e., coders, scientists, and domain experts) in tuning sets of matching rules. It aims to reduce human intervention by offering a tool for the optimization of rule sets based on user-defined criteria (such as effectiveness, interpretability, etc.). Our goal is to integrate the framework in the Magellan ecosystem, thus completing the functionalities required by the developers for performing Entity Matching tasks.

KEYWORDS

entity resolution, data deduplication, data integration

ACM Reference Format:
Matteo Paganelli, Paolo Sottovia, Francesco Guerra, and Yannis Velegrakis. 2019. TuneR: Fine Tuning of Rule-based Entity Matchers . In *The 28th ACM International Conference on Information and Knowledge Management (CIKM '19), November 3–7, 2019, Beijing, China.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3357384.3357854

1 INTRODUCTION

Entity matching (EM) is a fundamental task in data integration scenarios. It is the process of identifying records modeling the same real-world entity. One way of performing EM is by setting a set of rules that identify the conditions that would make two records be considered as matched. For instance, a rule could be of the form:

$$(name, edit, =, 1) \wedge (address, edit, >, 0.8) \wedge (city, edit, >, 0.8)$$

which indicates that two records would match if they fully agree on the name, the similarities between their respective address and city attributes are more than 80%. The rule also specifies that the similarity between two attributes is measured using the edit distance (word "edit"). It will, thus, determine that the records ⟨`name`:"pisces", `address`:"95 ave. a at 6th st.", `city`:"new york city"⟩ and ⟨`name`:"pisces", `address`:"95 ave. a", `city`:"new york city"⟩ match since the edit distance between their respective three attributes is 1 for the first, and more than 80% for the other two.

Generating EM rules is a complex and erroneous task. The data engineer will have to get into many intricate technical details to tune the EM, i.e., specify the rules. This specification means deciding what attributes need to be checked, what similarity functions to be used for that checking and what parameters and thresholds these similarity functions should run. In the past, this was a manual process performed by an expert. More modern approaches compute rules by employing Machine Learning techniques. Nevertheless, many challenges still remain.

One of the first challenges is *dimensionality*. Modern data sets are highly heterogeneous and for effective matching, many different factors can be considered. This means that there may be many rules that need to be defined. The functions in each rule have to be decided alongside the parameters in each one of them. Parameter tuning is an important task that needs to be performed for every rule for the best performance. This optimization cannot be done locally. The final decision on whether two structures match is the combination of the result of many rules, and optimizing each rule separately, does not necessarily mean that their combined outcome is also optimized. One has to consider the winning combination of functions and parameters collectively. The number of candidate combinations, however, is high. For n rules, applied on m attributes, using f different similarity functions, each with v possible thresholds (assuming that the threshold values are discrete), means that one has to consider $2^{n*m*f*v}$ combinations.

Another challenge is the decision on the *importance* of the rules. Not all rules are equal. Depending on the domain and the task, some rules may weight more in the final decision than others, or may not be preferred. For instance, rules capable of identifying matches with extreme precision could be very verbose. Scoring functions may work well in different scenarios, thus rules with highly effective functions for a particular domain play a more important role. The degree of overlap between the rules, the number of rules, or the

CIKM '19, November 3–7, 2019, Beijing, China

ACM ISBN 978-1-4503-6976-3/19/11…$15.00
https://doi.org/10.1145/3357384.3357854

number of predicates that each rule has, are all factors that affect the user preference towards specific rules.

Last but not least, rule configuration is not a one-time, one-way task. It is an *iterative*, trial and error process, where domain experts try different configurations to identify those that work best for a specific domain. Any tool support for this task is of paramount importance since it can offer significant savings in time and effort.

The above challenges have been studied in the past [1, 2, 5, 7], but only in isolation. They often focus only on the accuracy of the rules [7], without considering other optimization metrics. To make rules understandable by the user, some limit the rule length [5]. It does not consider optimization but adopts a statistical approach to limit the search space. The approach, however, is not deterministic and relies on external ad-hoc tools for the synthesis of programs to generate rules.

In this work, we describe and demonstrate TuneR, a software library we have developed and experimented for the Re-search Alps project[1] [6]. It supports the developers in the fine-tuning of a rule-based EM system. The library provides two main capabilities for managing sets of rules: (1) the fine-tuning of the rule parameters, i.e., given a number of rules, and a range of possible similarity functions and parameter values, the library returns the values that optimize user-defined scoring criteria on the results. An exhaustive approach would require testing a large number of possible configurations, i.e., for each rule, the ones generated by the cartesian product of sets of the user-selected parameters. Our approach implements a strategy that discards early bad configurations; (2) the weighting of the rules according to custom criteria.

To make the library generic and broadly applicable, we designed it to be fully integrated into the Magellan pipeline. Magellan [4] is a recent EM framework created with the aim of providing a unique ecosystem for the integrated and end-to-end management of the entity matching task. Magellan is considered the de-facto reference library for the developers who have to address data integration tasks. In the demonstration, we show how TuneR perfectly complements the functionalities provided by Magellan. We examine a series of possible use scenarios within an entire entity matching pipeline and show how TuneR can meet the pressing demands for simplification required from such a fundamental task in numerous academic and industrial contexts.

2 PROBLEM DEFINITION

Consider a dataset D of records $d_1, ..., d_N$. A record is a set of attribute-value pairs over a set of attributes $A = \{a_1, ..., a_m\}$, with each attribute used only once. Given a set of similarity functions $F = \{edit, Jaccard, cosine, ...\}$ and comparison operators $O = \{>, =, <, ...\}$, a *matching predicate* is a quadruple $p : (a, f, op, thr)$, where $a \in A$, $f \in F$, $op \in O$ and $thr \in \mathbb{R}$ is a threshold. For two records d_1 and d_2, the predicate $p : (a, f, op, thr)$ is true if and only if $O(f(d_1, d_2), thr)$ is true. For example, the predicate $(name, edit, >, 0.6)$ is true for two records if the edit distance between their name attribute is more than 0.6. A *template matching predicate* is a predicate that has as similarity function or threshold (or both) a variable instead of a constant value. $(name, F, >, 0.6)$, $(name, edit, >, T)$ and $(name, F, >, T)$ are examples of template matching predicates. A *rule* is a conjunction of matching predicates. A rule is said to be a *template* if it contains at least one template predicate. A rule set is a disjunction of a set of rules. For brevity we will not mention the term disjunction when it is clear of what it means and refer to it as simply a set. A *template rule set* is a rule set that has at least one template rule and/or at least two rule weights being variables.

The quality of a *rule set* is a *scoring function* over that set. Different users may consider different quality metrics. Some may be interested in the effectiveness of the set of rules, in which case the scoring function may be the f-measure, for instance. Others may be interested in the interpretability of the rules, for instance, giving preference to rule sets with few rules or to rules with reduced overlapping.

Tuning a template rule set is the task of finding a value assignment to the variables in the template rule set, such that the value of some user-selected scoring function is maximized.

Weighting a rule set is the task of finding a value assignment to the rule weights in the rule set, such that the value of some user-selected scoring function is maximized.

3 THE TUNER APPROACH

The workflow of TuneR is depicted in Figure 1, integrated within a generic Magellan EM pipeline.

Apart from the dataset on which it will apply the EM, TuneR requires also the ground truth and a *template rule set*. As preliminary steps, Magellan is used. Magellan provides 1) an interface based on Pandas [2] for data management and analysis and 2) a model, called Boolean Rule Matcher (BRM), for the management of EM tasks based on rules.

Rule Tuning The main task of TuneR is to implement the tuning of the provided template rule set. This means deciding for each similarity function variable the actual function, and for every parameter variable or threshold variable a specific value. The process boils down to an optimization problem, i.e., the discovery of that value assignment that achieves the best scoring function value. Unfortunately, the search space is large. Thus, TuneR uses an approximate technique to select the optimized set of rules. The technique is inspired by the deterministic local search [3], and finds a local optimum solution with a reduced approximate error according to user-defined optimization criteria.

The idea behind this local search technique is to progressively improve an initial solution by including and removing elements. More specifically, a single element (between the set of elements to be optimized) that generates the maximum score is selected as the initial solution. This solution is first extended and then reduced. Initially, the technique tries to integrate new elements into the current optimal solution. When the inclusion of a new element results into a higher score, the solution is extended. This iterative process is applied until no higher scores can be achieved by an element inclusion. In the second phase, the technique tries to remove elements from the current optimal solution. When the removal of an element produces a higher score, this element is removed and a new extension-reduction cycle is applied. The procedure ends when no other element can be removed from the optimal solution.

[1]Re-search Alps is a Connect Europe Facility project, funded by the European Union, Action no. 2016-EU-IA-0067, http://researchalps.eu.

[2]https://pandas.pydata.org/

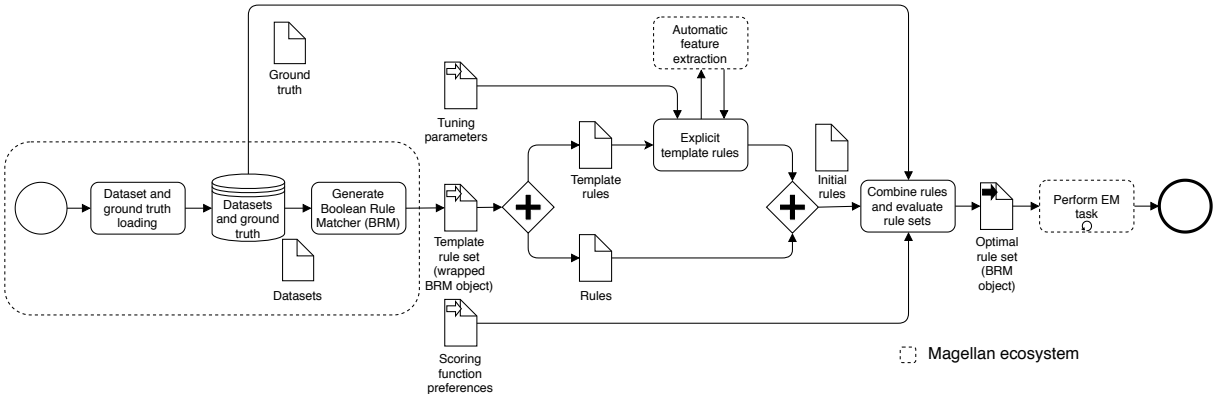

Figure 1: TuneR approach.

The application of this procedure to a set $\mathcal{X}$ of n rules evaluated with a scoring function f can be summarized as follows.

(1) Initialize the optimal set of rules $\mathcal{R}$ with the rule which generates the highest score (i.e., $\mathcal{R} := \{r\}$ where $f(\{r\})$ is the maximum over all singletons $r \in \mathcal{X}$);
(2) If there exists an element $r' \in \mathcal{X} \backslash \mathcal{R}$ such that $f(\mathcal{R} \cup \{r'\}) > (1 + \frac{\epsilon}{n^2}) f(\mathcal{R})$, then let $\mathcal{R} := \mathcal{R} \cup \{r'\}$, and go back to Step 2.
(3) If there exists an element $r' \in \mathcal{R}$ such that $f(\mathcal{R} \backslash \{r'\}) > (1 + \frac{\epsilon}{n^2}) f(\mathcal{R})$, then let $\mathcal{R} := \mathcal{R} \backslash \{r'\}$, and go back to Step 2.
(4) Return the maximum of $f(\mathcal{R})$ and $f(\mathcal{X} \backslash \mathcal{R})$

The optimization metrics (scoring function) adopted inside the implemented optimization function are precision, recall, compactness of rules (i.e., number of matching predicates in a rule), number of rules and their overlap. The user defines the contribution of each metric into the optimization function $f(R)$ as follows $f(R) = \sum_{j=0}^{k} \alpha_j * M_j$, where R is a set of rules, M_j and α_j are the optimization measures and its associated weight respectively. Another optional functionality provides the rules with a score that represents their importance in the set of rules according to the user-defined preferences. The process exploits a process similar to the one adopted for tuning the rules. Users are asked to select the contribution of each weighting metric into the optimization function and function returns the rules weighted according to the selected criteria.

EM task resolution Once rules optimization process is complete, it is possible to take advantage of the Boolean Rule Matcher containing the desired set of rules within Magellan's ecosystem to complete the entity matching task. If the result is not satisfactory, the user can review some tuning parameters and can incrementally modify the rules included in the optimal Boolean Rule Matcher.

4 DEMONSTRATION SCENARIOS

We demonstrate TuneR[3] through two scenarios that aim to highlight the benefits it can offer to the data engineer, namely the significant reduction of the rule set tuning effort and the flexibility of the approach that allows users to easily generate rules satisfying specific optimization criteria. The scenarios described in the paper are based on the Restaurant dataset [4], a collection of 864 restaurant records from the Fodor's and Zagat's restaurant guides that contains 112 duplicates. For this dataset, Magellan provides a ground truth of 450 elements[5]. During the demo we will use this simple dataset for introducing the main ideas of the application to the attendants, but we will also use other and larger datasets[6] for demonstrating the scalability of the technique.

The demonstration starts by analyzing TuneR API as shown in Figure 2. As usual in the Magellan framework, three dataframes have to be provided as input: two of them represent the datasets to apply the rules, the third is a ground truth where to evaluate the rules. The rules are encoded by using a wrapper of the Boolean Rule Matcher formalism adopted in Magellan. The other two parameters allow the users to define the rule elements to be optimized and the criteria used for their optimization. Optimization criteria concern the performance on the effectiveness used in the first scenario for tuning the rules and the performance on the interpretability used in the second scenario for weighting the rules. Precision, recall, f-measure, and accuracy are examples of effectiveness optimization criteria. Examples of criteria optimizing the interpretability of the rule set are the overall number of rules, the overlap, i.e., the normalized number of dataset elements covered by more than one rule , the support, i.e., the normalized number of dataset elements where a rule is applied, the confidence, i.e., the normalized number of dataset elements where a rule is applied with a correct result, ...). For each criterion the user can express a value in the range $[-1, 1]$, where values closed to -1 (1) denote that the associated criterion has to be minimized (maximized). Values close to 0 (default value) denote that the value is not relevant for the optimization process.

4.1 Scenario 1: generating and tuning EM rules

The first scenario simulates a domain expert that introduces matching rules to tune and evaluates if the overall rule set satisfies his desiderata. During the demo, we will show the typical iterative process followed by the developers who formulate, and tune rules for

[3] See the video at https://tinyurl.com/y37o4fpr

[4] http://www.cs.utexas.edu/users/ml/riddle/data.html
[5] https://sites.google.com/site/anhaidgroup/useful-stuff/data
[6] For example http://www.cs.utexas.edu/users/ml/riddle/data.html

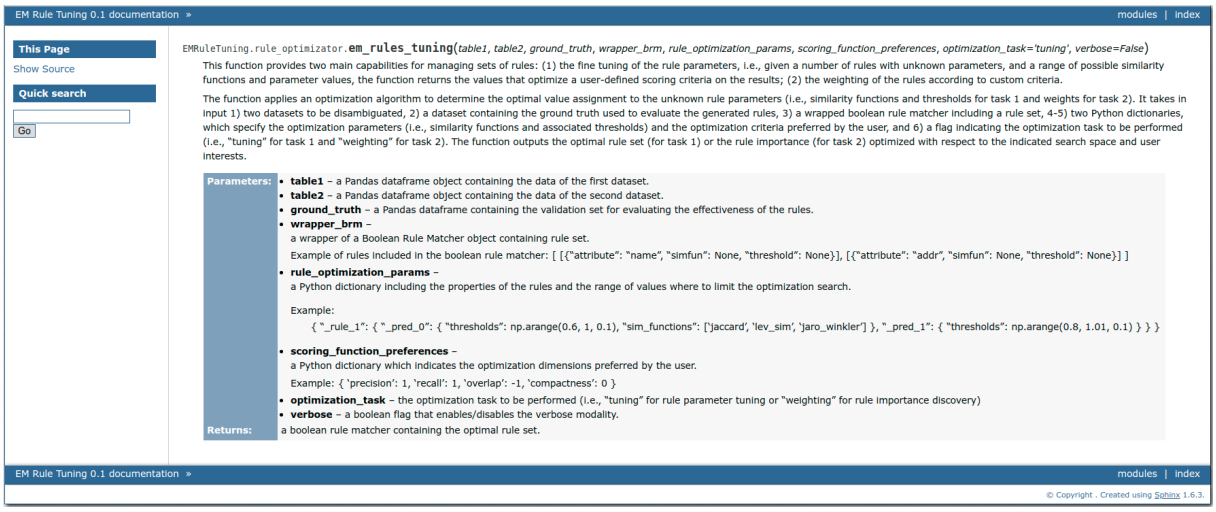

Figure 2: TuneR - rule tuning function documentation.

Table 1: Effectiveness evaluation for rule tuning.

set of rules	prec.	rec.	F1	false neg.	false pos.	over.	rules
$R_1 : (name, edit, >, 0.6)$	0.937	0.796	0.860	23	6	0.00	1
$R_2 : (name, edit, >, 0.8)$	1.000	0.759	0.863	27	0	0.00	1
$R_3 : (name, Jaccard, >, 0.4)$	0.970	0.875	0.920	14	3	0.00	1
$R_4 : (name, cosine, >, 0.5)$	0.939	0.964	0.952	4	7	0.00	1
$R_5 : (name, cosine, >, 0.5)$ $OR\ (addr, cosine, >, 0.6)$	0.926	1.000	0.961	0	9	20.67	2
$R_6 : (name, Jaccard, >, 0.6)$ $OR\ (addr, cosine, >, 0.7)$	1.000	0.973	0.986	3	0	14.00	2

finding duplicated entries in two datasets and evaluate the results achieved on the ground truth at each iteration.

The first 4 rows in Table 1 show possible tuning operation manually performed by the developer on the same rule (column 1) changing the threshold and/or the similarity measure. Note that all options reported show high f-measure levels (column 4), but differ in precision and recall (columns 2-3). Columns 5-9 describe the number of false negative, false positive, the overlapping degree and the number of rules, respectively.

The demonstration will show that the manual tuning of matching rules is a time consuming and error-prone approach. TuneR can greatly facilitate the user's work, by running the optimization process with the *template rule* $(name, \{edit, Jaccard, cosine\}, >, [0.4 : 0.8 : 0.1])$, where three choices for the similarity function are expressed by the user and the threshold can assume a value in the interval $[0.4, 0.8]$ with step 0.1. Accordingly with the user-selected optimization criteria, rules R_1 - R_4 can be generated with a single execution (e.g., R_4 is generated by maximizing precision and recall).

We will repeat the tuning process starting from R_5, a rule set which has been automatically generated through a machine learning approach. In this case, if the user wants to tune the rule for maximizing precision and recall in the optimization function he can run the application with the *template* $(name, \{edit, Jaccard, cosine\}, > , [0.4 : 0.8 : 0.1]) \vee (addr, \{edit, Jaccard, cosine\}, >, [0.4 : 0.9 : 0.1])$, thus obtaining R_6.

4.2 Scenario 2: Weighting rules

In a second scenario, we demonstrate how TuneR can support users in understanding the importance of the rules in a rule set according to user-specified criteria. Given as input a rule set, TuneR is able to provide a score for each constituent rules based on selected criteria (e.g., length, support, ...). In this way, the user can order the rules and identify which are the most relevant ones according to his perspective. We will show how changing the criteria, the scores and then the importance of the rules vary.

Furthermore, we will also show the effect generated by a new rule in a dataset. We will start from a rule set and we will compute the weights associated with its rules according to some user-specified criteria. We will perform the same computation applied to the same rule set enriched with the new rule and we will evaluate how the weights change.

In a final experiment, we will start from a number of rule sets having similar effectiveness performance. We will apply multiple times TuneR to all rule sets with different user criteria. We will analyze the overall score obtained and show, for each configuration criterium, the rule set that optimizes it.

REFERENCES

[1] Chaudhuri, S., Chen, B., Ganti, V., and Kaushik, R. Example-driven design of efficient record matching queries. In *Proceedings of VLDB* (2007), pp. 327–338.
[2] Fan, W., Jia, X., Li, J., and Ma, S. Reasoning about record matching rules. *PVLDB 2*, 1 (2009), 407–418.
[3] Feige, U., Mirrokni, V. S., and Vondrák, J. Maximizing non-monotone submodular functions. *SIAM J. Comput. 40*, 4 (2011), 1133–1153.
[4] Konda, P., Das, S., C., P. S. G., Doan, A., Ardalan, A., Ballard, J. R., Li, H., Panahi, F., Zhang, H., Naughton, J. F., Prasad, S., Krishnan, G., Deep, R., and Raghavendra, V. Magellan: Toward building entity matching management systems. *PVLDB 9*, 12 (2016), 1197–1208.
[5] Singh, R., Meduri, V. V., Elmagarmid, A. K., Madden, S., Papotti, P., Quiané-Ruiz, J., Solar-Lezama, A., and Tang, N. Synthesizing entity matching rules by examples. *PVLDB 11*, 2 (2017), 189–202.
[6] Sottovia, P., Paganelli, M., Guerra, F., and Vincini, M. Big data integration of heterogeneous data sources: the re-search alps case study. In *Proceedings IEEE International Congress on Big Data (BigData Congress)* (2019), pp. 106–110.
[7] Wang, J., Li, G., Yu, J. X., and Feng, J. Entity matching: How similar is similar. *PVLDB 4*, 10 (2011), 622–633.