Active Learning of Expressive Linkage Rules for the Web of Data

Robert Isele, Anja Jentzsch, and Christian Bizer

Freie Universität Berlin, Web-based Systems Group
Garystr. 21, 14195 Berlin, Germany
mail@robertisele.com, mail@anjajentzsch.de, chris@bizer.de

Abstract. The amount of data that is available as Linked Data on the Web has grown rapidly over the last years. However, the linkage between data sources remains sparse as setting RDF links means effort for the data publishers. Many existing methods for generating these links rely on explicit linkage rules which specify the conditions which must hold true for two entities in order to be interlinked. As writing good linkage rules by hand is a non-trivial problem, the burden to generate links between data sources is still high. In order to reduce the effort and required expertise to write linkage rules, we present an approach which combines genetic programming and active learning for the interactive generation of expressive linkage rules. Our approach automates the generation of a linkage rule and only requires the user to confirm or decline a number of example links. The algorithm minimizes user involvement by selecting example links which yield a high information gain. The proposed approach has been implemented in the Silk Link Discovery Framework. Within our experiments, the algorithm was capable of finding linkage rules with a full F1-measure by asking the user to confirm or decline a maximum amount of 20 links.

1 Introduction

The central idea of Linked Data is to extend the Web with a global data space by making data accessible according to the Linked Data best practices [7] and by setting RDF links between data sources. While the amount of data that is accessible as Linked Data has grown significantly over the last years, most data sources are still not sufficiently interlinked[1]. In order to help data publishers to set RDF links pointing into other data sources, several link discovery tools have been developed. These tools compare entities in different Linked Data sources based on user-provided linkage rules which specify the conditions that must hold true for two entities in order to be interlinked. Writing good linkage rules by hand is a non-trivial problem as the rule author needs to have detailed knowledge about the structure of the data sets to be interlinked.

In this paper, we present an approach to learn linkage rules interactively using active learning and genetic programming. It learns a linkage rule by asking

[1] http://lod-cloud.net/state/ (09/19/2011)

M. Brambilla, T. Tokuda, and R. Tolksdorf (Eds.): ICWE 2012, LNCS 7387, pp. 411–418, 2012.

the user to confirm or reject example links which are actively selected by the algorithm. Our approach lowers the required level of expertise as the task of generating the linkage rule is automated while the user only has to verify a set of example links. User involvement is minimized by only selecting the links with the highest information gain. Within our experiments, the algorithm was capable of finding linkage rules with a full F1-measure by asking the user to confirm or decline a maximum amount of 20 links. The algorithm chooses which properties to compare together with distance measures, aggregation functions, thresholds and data transformations to normalize data prior to comparison. Although in this paper we focus on interlinking data sources in the Web of Data, our approach is not limited to that use case and can be applied to entity matching in other areas - such as in the context of relational databases - as well.

This paper makes the following contributions to the state of the art:

1. We are the first to apply an approach that combines genetic programming and active learning to the problem of learning linkage rules for generating RDF links in the context of the Web of Data.
2. The learned rules are more expressive than the linkage rules learned in previous work on entity matching as our algorithm combines different similarity measures nonlinearly and also determines the data translations that should be employed to normalize data prior to comparison.
3. We have implemented the proposed approach in the Silk Link Discovery Framework which is available under the terms of the Apache License.

This paper is organized as follows: Section 2 introduces our linkage rule representation. Section 3 describes the proposed active learning method. Section 4 presents the experimental evaluation. Finally, Section 5 discusses related work.

2 Linkage Rules

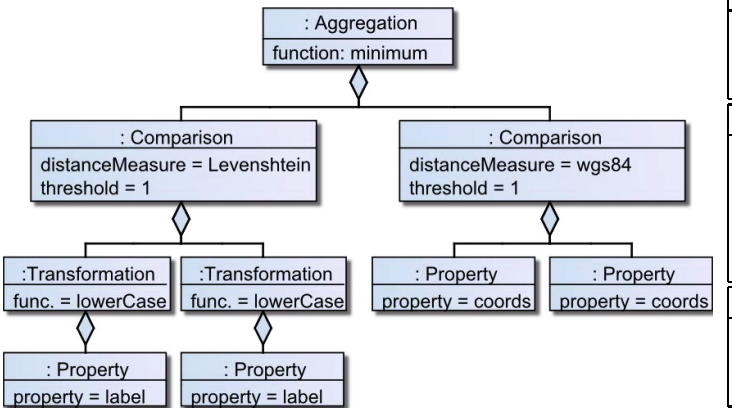
We represent a linkage rule as a tree built from 4 basic operators:

Property Operator: Creates a set of values to be used for comparison by retrieving all values of a specific property of the entity.

Transformation Operator: Transforms the input values of a single entity according to a specific data transformation function. Examples of common transformation functions include case normalization, tokenization and concatenation of the values of multiple operators. Multiple transformation operators can be nested in order to apply a sequence of transformations.

Comparison Operator: Evaluates the similarity between the values of two input operators according to a specific distance measure, such as Levenshtein, Jaccard, or geographic distance. Allowed input operators are property operators and transformation operators. A threshold specifies the maximum distance.

Aggregation Operator: Aggregates the similarity values from multiple operators into a single value according to a specific aggregation function. Aggregation functions such as the weighted average may take weights into account. Aggregation operators can be nested in order to create nonlinear hierarchies.



(1.1): Example linkage rule

Aggregations
Average
Maximum
Minimum
Dist. Measures
Levenshtein dist.
Jaccard index
Numeric dist.
Geographic dist.
Date dist.
Transformations
Lower case
Tokenize
Strip URI prefix

(1.2): Functions

Our approach is independent of any specific aggregation functions, distance measures or data transformations. Table 1.2 shows the set of functions which has been used by us in all experiments[2].

3 Learning Workflow

The main idea of our approach is to evolve a population of linkage rules iteratively while building a set of reference links. Figure 1 summarizes the three steps which are involved in each iteration:

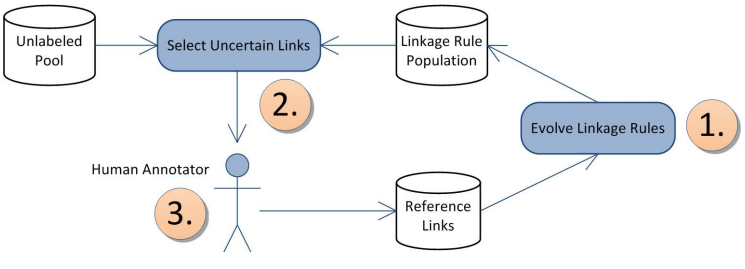


Fig. 1. Learning Workflow

(1) A genetic programming algorithm evolves a population of linkage rules. It starts with an initial population of candidate solutions which is iteratively evolved by applying genetic operators. The population is evolved until either a maximum number of iterations is reached or a linkage rule has been found which covers all reference links. As in most use cases there are either no or only a small number of reference links available, the purpose of the subsequent steps is to add additional links to the set of reference links.

[2] The details about the functions are provided in the Silk user manual on the website

(2) The active learning algorithm selects the example links to be labeled by the user. Links are selected according to a query by committee (QBC) strategy: As the linkage rules in the population are all trained on the current reference links they represent competing hypotheses. By selecting the links for which most linkage rules disagree, the contribution of the user confirming or declining the link is maximized. The links are selected from an unlabeled pool of possible links.

(3) A human expert labels the selected links as correct or incorrect. Confirmed links are added to the positive reference link set and declined links are added to the negative reference link set. After the human expert labeled the provided links, the genetic algorithm continues evolving the current population with the updated reference links.

3.1 Evolving Linkage Rules

The initial population is build by generating random linkage rules. In order to reduce the search space, we employ a simple algorithm by only selecting property pairs which hold similar values.

Starting with the current population, the genetic algorithm breeds a new population by evolving selected individuals using the genetic operations crossover and mutation. A detailed description of the algorithm as well as the specific crossover and mutation operators that are used is found in [8]. The individuals are selected from the population based on two functions: The *fitness function* and the *selection method*. The purpose of the *fitness function* is to assign a value to each individual which indicates how close the given individual is to the desired solution. In order to make the algorithm more robust against unbalanced reference link sets, we use *Matthews correlation coefficient* (MCC) as fitness measure. Based on the fitness of each individual, the *selection method* selects the individuals to be evolved. As selection method we chose tournament selection as it has been shown to produce strong results in a variety of GP systems [10] and is easy to parallelize.

The algorithm iteratively evolves the population until either a linkage rule has been found which covers all reference links or a configured maximum number of iterations is reached.

3.2 Selecting Uncertain Links

This section describes how the initial unlabeled pool is generated from which the links are selected for manual evaluation according to the query strategy.

Building the Unlabeled Pool. The overall goal of the learning algorithm is to create a linkage rule which is able to label all possible entity pairs as matches or non-matches with high confidence. The number of possible entity pairs can be very high for large data sets and usually far exceeds the number of actual matches. For this reason we use an indexing approach to build a sample which does not include definitive non-matches.

Given two data sets A and B, the initial unlabeled pool $\mathcal{U} \subset A \times B$ is built according to the following sampling process: The sampling starts by querying for all entities in both data sets. Instead of retrieving all entities at once, a stream of

entities is generated for each data set. For each property in the streamed entities, all values are indexed according to the following schema:

1. All values are normalized by removing all punctuation and converting all characters to lower case.
2. The normalized values are tokenized.
3. A set of indices is assigned to each token. The indices are generated so that tokens within an edit distance of 1 share at least one index. The MultiBlock blocking algorithm is used to generate the index [9].
4. The indices of all tokens of a value are merged. If in total more than 5 indices have been assigned to a value, 5 indices are randomly selected while discarding the remaining indices.

Now all pairs of entities which have been assigned the same index are added to the unlabeled pool until a configured maximum size is reached.

Query Strategy. The purpose of the query strategy is to select links which are to be labeled by a human expert as correct or incorrect. In order to minimize the number of links to be verified by the user the algorithm selects the links from the unlabeled pool for which the linkage rules in the current population disagree the most. To measure the disagreement with respect to a link l, we use the vote entropy:

$$d(l) = H\left(\frac{v(l)}{|P|}\right) \quad \text{with } H(p) = -p \cdot log(p) - (1-p) \cdot log(1-p)$$

With $v(l)$ denoting the number of linkage rules which confirm the link and $|P|$ the size of the population. The rationale here is that if the percentage of linkage rules which confirm the link approaches 50%, half of the linkage rules are implicitly upvoted or downvoted by the user by confirming or declining the given link. The links with the highest vote entropy are returned for evaluation by the user.

4 Evaluation

4.1 Experiment Setup

All experiments have been run 10 times with 2 fold cross-validation. For each experiment, we provide the averaged results with respect to the training data set as well as the validation data set together with the standard deviation. All experiments have been run on a 3GHz Intel(R) Core i7 CPU with 4 cores while the Java heap space has been restricted to 1GB. Table 1 lists the parameters which have been used in all experiments.

Table 1. Parameters

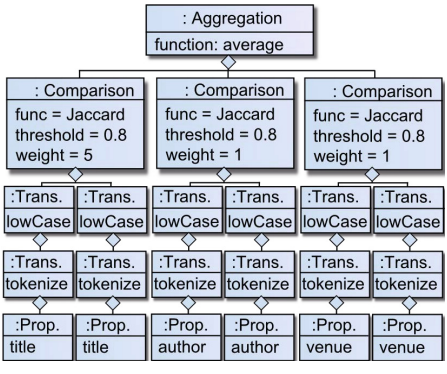
Parameter	Value	Parameter	Value
Population size	500	Probability of Mutation	25%
Maximum generations	50	Stop Condition	MCC = 1.0
Selection method	Tournament selection	Unlabeled Pool size $\|\mathcal{U}\|$	10,000
Tournament size	5	Query Size $\|\mathcal{U}_q\|$	5

4.2 Experiment 1: Comparison with Related Work

In order to show the competitiveness of the genetic programming algorithm that is used to evolve the populations within our approach, we compare the algorithm to the state-of-the-art genetic programming algorithm presented by Carvalho et. al. in [4]. One data set commonly used for evaluating different record deduplication approaches is *Cora*. The Cora data set contains citations to research papers from the Cora Computer Science research paper search engine. Table 2.1 summarizes the cross validation results. The learned linkage rules compared by title, author and venue. Figure 2.2 shows an example of a learned linkage rule. On average, our approach achieved an F-measure of 96.9% against the training set and 93.6% against the validation set and needed less than 5 minutes to perform all 50 iterations on the test machine. For the same data set, Carvalho et. al. report an F-measure of 90.0% against the training set and 91.0% against the validation set [4].

Iter.	Time	Train. F1 (σ)	Val. F1 (σ)
1	4.0	0.896 (0.022)	0.896 (0.021)
10	31.1	0.956 (0.013)	0.954 (0.015)
20	71.4	0.964 (0.008)	0.960 (0.010)
30	132.5	0.965 (0.007)	0.962 (0.007)
40	217.6	0.968 (0.004)	0.945 (0.036)
50	271.1	0.969 (0.003)	0.936 (0.056)
Ref.	-	0.900 (0.010)	0.910 (0.010)

(2.1): Experiment 1. The last row contains the best results of Carvalho et. al.



(2.2): Example linkage rule

4.3 Experiment 2: Active Learning

In this experiments we evaluated that the presented approach is able to learn a linkage rule with minimal user interaction using an easy to understand example from the media domain: Interlinking movies from LinkedMDB[3] with the corresponding entry in DBpedia about the same movie. For evaluation we used a manually created set of 100 positive and 100 negative reference links.

First we evaluated the capability to learn a linkage rule from reference links. Table 2.1 shows the averaged results of all runs. In all runs, the learning algorithm needed no more than 12 iterations in order to achieve the full F-measure.

Next we evaluated if our active learning approach is able to build a reference link set interactively. For this, we started with an empty reference link set and in each iteration let a user manually confirm and decline 5 links which have been selected by the algorithm. We repeated the experiment 3 times and averaged the

[3] `http://linkedmdb.org/`

Iter.	Time in s (σ)	Train. F1 (σ)	Val. F1 (σ)
1	5.9 (0.1)	0.968 (0.005)	0.968 (0.005)
10	19.6 (4.0)	0.995 (0.005)	0.995 (0.005)
12	20.9 (5.3)	1.000 (0.000)	1.000 (0.000)

(2.1): Experiment 2: Passive learning

Iter.	Links	Train. F1	Val. F1
1	5	1.00	0.65
2	10	1.00	0.97
3	15	1.00	0.99
4	20	1.00	1.00

(2.2): Experiment 2: Active l.

results. Table 2.2 shows results for each iteration. For each iteration it shows the F-measure based on the manually confirmed links (Training F1) and on the full reference link set (Validation F1). The second column shows the links which have been evaluated by the user. The results show that after querying 10 links from the user, the learning algorithm already learned a linkage rule which achieves a F-measure of 97 % when compared with the reference links.

4.4 Experiment 3: Large Scale Active Learning

In this experiment we show that the presented approach is able to scale to large data sets. At the time of writing, DBpedia contains 323,257 settlements while LinkedGeoData contains 560,123 settlements. The execution of the learned linkage rules generates over 70,000 links. While for passive learning the learning only needs to take the provided reference links into account, active learning also needs to take the pool of unlabeled data into account which in this example amounts to over 180 billion pairs. In order to evaluate the learned linkage rules we used a manually collected set of 100 positive and 100 negative reference links.

Iter.	Time (σ)	Train. F1 (σ)	Val. F1 (σ)
1	2.6s (1.0)	0.984 (0.025)	0.932 (0.059)
10	3.8s (2.1)	0.996 (0.007)	0.932 (0.059)
20	3.9s (2.3)	0.998 (0.004)	0.964 (0.032)
25	4.0s (2.4)	1.000 (0.000)	1.000 (0.000)

(2.1): Experiment 3: Passive learning

Iter.	Links	Time	Train. F1	Val. F1
1	5	7.3s	1.00	0.98
2	10	15.6s	1.00	1.00

(2.2): Experiment 3: Active learning

Table 2.1 summarizes the cross validation results for passive learning. Table 2.2 shows results for each iteration in an active learning setting. The runtimes only include the time needed by the algorithm itself and not the time needed by the human to label the examples.

In all three runs, the algorithm managed to learn a linkage rule with full F-measure after the second iteration. In the first iteration it missed the case that two entities with the same name may in fact relate to different cities. In the second iteration it managed to include this rare case in the proposed example links.

5 Related Work

Genetic Programming. To the best of our knowledge, genetic programming for learning linkage rules has only been applied by Carvalho et. al. so far [3,2,4].

Their approach uses genetic programming to learn how to combine a set of pre-supplied pairs of the form `<attribute, similarity function>` (e.g. `<name, Jaro>`) into a linkage rule. Their approach is very expressive although it cannot express data transformations. On the downside, using mathematical functions to combine the similarity measures does not fit any commonly used linkage rule model [6] and leads to complex and difficult to understand linkage rules. We are not aware of any previous application of genetic programming to learn linkage rules in the context of Linked Data other than our own work [8].

Active Learning. While the majority of the approaches targeted at learning linkage rules use supervised learning, some approaches based on active learning have been proposed: Arasu et. al. propose a scalable active learning approach for entity matching by introducing the assumption of *monotonicity of precision* [1]. While they show that their approach can scale to large data sets, it is only able to learn simple linear or boolean classifiers, while our approach is capable of learning expressive linkage rules which include nonlinear aggregation hierarchies and data transformations. The only approach which combines genetic programming and active learning to learn rules for record deduplication known to us has been proposed by Freitas et. al. [5]. It is based on the genetic programming approach by Carvalho et. al. mentioned earlier and thus shares its limitations as described in the previous Section.

References

1. Arasu, A., Götz, M., Kaushik, R.: On active learning of record matching packages. In: Proceedings of the 2010 International Conference on Management of Data, SIGMOD 2010, pp. 783–794. ACM, New York (2010)
2. Carvalho, M., Laender, A., Gonçalves, M., da Silva, A.: Replica identification using genetic programming. In: Proceedings of the 2008 ACM Symposium on Applied Computing, pp. 1801–1806. ACM (2008)
3. de Carvalho, M.G., Gonçalves, M.A., Laender, A.H.F., da Silva, A.S.: Learning to deduplicate. In: Proceedings of the 6th ACM/IEEE-CS Joint Conference on Digital Libraries, JCDL 2006, pp. 41–50. ACM, New York (2006)
4. de Carvalho, M.G., Laender, A.H.F., Goncalves, M.A., da Silva, A.S.: A genetic programming approach to record deduplication. IEEE Transactions on Knowledge and Data Engineering 99(preprints) (2010)
5. de Freitas, J., Pappa, G., da Silva, A., Gonçalves, M., Moura, E., Veloso, A., Laender, A., de Carvalho, M.: Active learning genetic programming for record deduplication. In: Evolutionary Computation (CEC), pp. 1–8. IEEE (2010)
6. Euzenat, J., Shvaiko, P.: Ontology matching. Springer (2007)
7. Heath, T., Bizer, C.: Linked data: Evolving the web into a global data space. Synthesis Lectures on the Semantic Web: Theory and Technology 1(1), 1–136 (2011)
8. Isele, R., Bizer, C.: Learning linkage rules using genetic programming. In: 6th International Workshop on Ontology Matching, Bonn, Germany (2011)
9. Isele, R., Jentzsch, A., Bizer, C.: Efficient multidimensional blocking for link discovery without losing recall. In: 14th International Workshop on the Web and Databases (WebDB 2011), Athens (2011)
10. Koza, J., Keane, M., Streeter, M., Mydlowec, W., Yu, J., Lanza, G.: Genetic programming IV: Routine human-competitive machine intelligence. Springer (2005)