Learning Conformation Rules for Linked Data Integration

Axel-Cyrille Ngonga Ngomo[1]

Department of Computer Science
University of Leipzig
Johannisgasse 26, 04103 Leipzig
ngonga@informatik.uni-leipzig.de,
WWW home page: http://bis.uni-leipzig.de/AxelNgonga

Abstract. Over the last years, manifold applications that consume, link and integrate Linked Data have been developed. Yet, the specification of integration processes for Linked Data is rendered increasingly tedious by several factors such as the great number of knowledge bases in the Linked Data Cloud as well as schema mismatches and heterogeneous conventions for property values across knowledge bases. Especially the specification of rules for transforming property values has been carried out mostly manually so far. In this paper, we present CaRLA, an algorithm that allows learning transformation rules for pairs of property values expressed as strings. We present both a batch and an active learning version of CaRLA. The batch version of CaRLA uses a three-step learning approach to retrieve probable transformation rules. The active learning version extends the batch version by requesting highly informative property value pairs from the user so as to improve the learning speed of the system. We evaluate both versions of our approach on four experiments with respect to runtime and accuracy. Our results show that we can improve the precision of the data integration process by up to 12% by discovering transformation rules with human accuracy even when provided with small training datasets. In addition, we can even discover rules that were missed by human experts.

1 Introduction

The Linked Open Data (LOD) Cloud consists of more than 30 billion triples[1]. Making use of this large amount of domain knowledge within large-scale semantic applications is currently gaining considerable momentum. For example, the DeepQA framework [4] combines knowledge from DBpedia[2], Freebase[3] and several other knowledge bases to determine the answer to questions with a speed superior to that of human champions. Complex applications that rely on several sources of knowledge usually integrate them into a unified view by the means of

[1] http://www4.wiwiss.fu-berlin.de/lodcloud/state/
[2] http://dbpedia.org
[3] http://www.freebase.com

the Extract-Transform-Load (ETL) paradigm [7]. Yet, over the last few years, the automatic provision of such unified views on Linked datasets has been rendered increasingly tedious. The difficulties behind the integration of Linked Data are not only caused by the mere growth of the datasets in the Linked Data Web but also by large disparity across these datasets.

Linked Data Integration is commonly impeded by two categories of mismatches: ontology mismatches and naming convention mismatches. The second category of common mismatches (which is the focus of this work) mostly affects the transformation step of ETL and lies in the different conventions used for equivalent property values. For example, the labels of films in DBpedia differ from the labels of films in LinkedMDB[4] in three ways: First, they contain a language tag. Second, the extension "(film)" is added to the label of movies if another entity with the same label exists. Third, if another film with the same label exists, the production year of the film is added. Consequently, the film `Liberty` from 1929 has the label "Liberty (1929 film)@en" in DBpedia, while the same film bears the label "Liberty" in LinkedMDB. A similar discrepancy in naming persons holds for film directors and actors. Finding a conform representation of the labels of movies that maps the LinkedMDB representation would require knowing the rules `replace(''@en'', ϵ)` and `replace(''(*film)'', ϵ)` where ϵ stands for the empty string.

In this paper, we address the problem of discovering transformation rules by presenting CaRLA, the Canonical Representation Learning Algorithm. Our approach learns canonical (also called conform) representation of data type property values by implementing a simple, time-efficient and accurate learning approach. We present two versions of CaRLA: a batch learning and an active learning version. The batch learning approach relies on a training dataset to derive rules that can be used to generate conform representations of property values. The active version of CaRLA (aCarLa) extends CaRLA by computing unsure rules and retrieving highly informative candidates for annotation that allow the validation or negation of these candidates. One of the main advantages of CaRLA is that it can be configured to learn transformations at character, n-gram or even word level. By these means, it can be used to improve integration and link discovery processes based on string similarity/distance measures ranging from character-based (edit distance) and n-gram-based (q-grams) to word-based (Jaccard similarity) approaches.

The rest of this paper is structured as follows: In Section 2, we give an overview of the notation used in this paper. Thereafter, we present the two different versions of the CaRLA algorithm: the batch version in Section 3 and the active version in Section 4. Subsequently, we evaluate our approach with respect to both runtime and accuracy in four experiments (Section 5). After a brief overview of the related work (Section 6), we present some future work and conclude in Section 7.

[4] http://linkedmdb.org/

2 Preliminaries

In the following, we define terms and notation necessary to formalize the approach implemented by CaRLA. Let $s \in \Sigma^*$ be a string from an alphabet Σ. We define a tokenization function as follows:

Definition 1 (Tokenization Function). *Given an alphabet A of tokens, a tokenization function token : $\Sigma^* \rightarrow 2^A$ maps any string $s \in \Sigma^*$ to a subset of the token alphabet A.*

Note that string similarity and distances measures rely on a large number of different tokenization approaches. For example, the Levenshtein similarity [8] relies on a tokenization at character level, while the Jaccard similarity [6] relies on a tokenization at word level.

Definition 2 (Transformation Rule). *A transformation rule is a function $r : A \rightarrow A$ that maps a token from the alphabet A to another token of A.*

In the following we will denote transform rules by using an arrow notation. For example, the mapping of the token "`Alan`" to "`A.`" will be denoted by $<$ "`Alan`" $\rightarrow$ "`A.`"$>$. For any rule $r =< x \rightarrow y >$, we call x the *premise* and y the *consequence* of r. We call a transformation rule *trivial* when it is of the form $< \mathtt{x} \rightarrow \mathtt{x} >$ with $x \in A$. We call two transformation rules r and r' *inverse* to each other when $r =< \mathtt{x} \rightarrow \mathtt{y} >$ and $r' =< \mathtt{y} \rightarrow \mathtt{x} >$. Throughout this work, we will assume that the characters that make up the tokens of A belong to $\Sigma \cup \{\epsilon\}$, where ϵ stands for the empty character. Note that we will consequently denote deletions by rules of the form $< x \rightarrow \epsilon >$ where $x \in A$.

Definition 3 (Weighted Transformation Rule). *Let Γ be the set of all rules. Given a weight function $w : \Gamma \rightarrow \mathbb{R}$, a weighted transformation rule is the pair $(r, w(r))$, where $r \in \Gamma$ is a transformation rule.*

Definition 4 (Transformation Function). *Given a set R of (weighted) transformation rules and a string s, we call the function $\varphi_R : \Sigma^* \rightarrow \Sigma * \cup \{\epsilon\}$ a transformation function when it maps s to a string $\varphi_R(s)$ by applying all rules $r_i \in R$ to every token of token(s) in an arbitrary order.*

For example, the set $R = \{<$"`Alan`"$\rightarrow$"`A.`"$>\}$ of transformation rules would lead to φ_R("James Alan Hetfield") ="James A. Hetfield".

3 Batch Learning Approach

The goal of CaRLA is two-fold: First, it aims to compute rules that allow the derivation of conform representations of property values. As entities can have several values for the same property, CaRLA also aims to detect a condition under which two property values should be merged during the integration process. In the following, we will assume that two source knowledge bases are to be integrated to one. Note that our approach can be used for any number of source knowledge bases.

3.1 Overview

Formally, CaRLA addresses the problem of finding the required transformation rules by computing an equivalence relation $\mathcal{E}$ between pairs of property values (p_1, p_2) that is such that $\mathcal{E}(p_1, p_2)$ holds when p_1 and p_2 should be mapped to the same *canonical representation* p. CaRLA computes $\mathcal{E}$ by generating two sets of weighted transformation function rules R_1 and R_2 such that for a given similarity function σ, $\mathcal{E}(p_1, p_2) \rightarrow \sigma(\varphi_{R_1}(p_1), \varphi_{R_2}(p_2)) \geq \theta$, where θ is a similarity threshold. The canonical representation p is then set to $\varphi_{R_1}(p_1)$. The similarity condition $\sigma(\varphi_{R_1}(p_{R_1}), \varphi_{R_2}(p_2)) \geq \theta$ is used to distinguish between the pairs of properties values that should be merged.

To detect R_1 and R_2, CaRLA assumes two training datasets P and N, of which N can be empty. The set P of positive training examples is composed of pairs of property value pairs (p_1, p_2) such that $\mathcal{E}(p_1, p_2)$ holds. The set N of negative training examples consists of pairs (p_1, p_2) such that $\mathcal{E}(p_1, p_2)$ does not hold. In addition, CaRLA assumes being given a similarity function σ and a corresponding tokenization function *token*. Given this input, CaRLA implements a three-step approach. It begins by computing the two sets R_1 and R_2 of plausible transformation rules based on the positive examples at hand (Step 1). Then it merges inverse rules across R_1 and R_2 and discards rules with a low weight during the rule merging and filtering step. From the resulting set of rules, CaRLA derives the similarity condition $\mathcal{E}(p_1, p_2) \rightarrow \sigma(\varphi_{R_1}(p_1), \varphi_{R_2}(p_2)) \geq \theta$. It then applies these rules to the negative examples in N and tests whether the similarity condition also holds for the negative examples. If this is the case, then it discards rules until it reaches a local minimum of its error function. The retrieved set of rules and the novel value of θ constitute the output of CaRLA and can be used to generate the canonical representation of the properties in the source knowledge bases.

In the following, we explain each of the three steps in more detail. Throughout the explanation we use the toy example shown in Table 1. In addition, we will assume a word-level tokenization function and the Jaccard similarity.

Type	Property value 1	Property value 2
$\oplus$	"Jean van Damne"	"Jean Van Damne (actor)"
$\oplus$	"Thomas T. van Nguyen"	"Thomas Van Nguyen (actor)"
$\oplus$	"Alain Delon"	"Alain Delon (actor)"
$\oplus$	"Alain Delon Jr."	"Alain Delon Jr. (actor)"
$\ominus$	"Claude T. Francois"	"Claude Francois (actor)"

Table 1. Toy example dataset. The positive examples are of type $\oplus$ and the negative of type $\ominus$.

3.2 Rule Generation

The goal of the rule generation set is to compute two sets of rules R_1 resp. R_2 that will underlie the transformation φ_{R_1} resp. φ_{R_2}. We begin by tokenizing all positive property values p_i and p_j such that $(p_i, p_j) \in P$. We call T_1 the set of all tokens p_i such that $(p_i, p_j) \in P$, while T_2 stands for the set of all p_j. We begin the computation of R_1 by extending the set of tokens of each $p_j \in T_2$ by adding ϵ to it. Thereafter, we compute the following rule score function *score*:

$$score(< x \to y >) = |\{(p_i, p_j) \in P : x \in token(p_i) \wedge y \in token(p_j)\}|. \quad (1)$$

score computes the number of co-occurrences of the tokens x and y across P. All tokens $x \in T_1$ always have a maximal co-occurrence with ϵ as it occurs in all tokens of T_2. To ensure that we do not only compute deletions, we decrease the score of rules $< x \to \epsilon >$ by a factor $\kappa \in [0, 1]$. Moreover, in case of a tie, we assume the rule $< x \to y >$ to be more natural than $< x \to y' >$ if $\sigma(x, y) > \sigma(x, y')$. Given that σ is bound between 0 and 1, it is sufficient to add a fraction of $\sigma(x, y)$ to each rule $< x \to y >$ to ensure that the better rule is chosen. Our final score function is thus given by

$$score_{final}(< x \to y >) = \begin{cases} score(< x \to y >) + \sigma(x, y)/2. & \text{if } y \neq \epsilon, \\ \kappa \times score(< x \to y >) & \text{else.} \end{cases} \quad (2)$$

Finally, for each token $x \in T_1$, we add the rule $r =< x \to y >$ to R_1 iff $x \neq y$ (i.e., r is not trivial) and $y = \arg\max_{y' \in T_2} score_{final}(< x \to y' >)$. To compute R_2 we simply swap T_1 and T_2, invert P (i.e., compute the set $\{(p_j, p_i) : (p_i, p_j) \in P\}$) and run through the procedure described above.

For the set P in our example, we get the following sets of rules: $R_1 = \{(<$"van"$\to$"Van"$>$, 2.08), $(<$"T."$\to \epsilon >$, 2)$\}$ and $R_2 = \{(<$"Van"$\to$"van"$>$, 2.08), $(<$"(actor)"$\to \epsilon >$, 2)$\}$.

3.3 Rule Merging and Filtering

The computation of R_1 and R_2 can lead to a large number of inverse or improbable rules. In our example, R_1 contains the rule $<$"van"$\to$"Van"$>$ while R_2 contains $<$"Van"$\to$"van"$>$. Applying these rules to the data would consequently not improve the convergence of their representations. To ensure that the transformation rules lead to similar canonical forms, the rule merging step first discards all rules $< x \to y > \in R_2$ that are such that $< y \to x > \in R_1$ (i.e., rules in R_2 that are inverse to rules in R_1). Then, low-weight rules are discarded. The idea here is that if there is not enough evidence for a rule, it might just be a random event. The initial similarity threshold θ for the similarity condition is finally set to

$$\theta = \min_{(p_1, p_2) \in P} \sigma(\varphi_{R_1}(p_1), \varphi_{R_2}(p_2)). \quad (3)$$

In our example, CaRLA would discard $<$"van"$\to$"Van"$>$ from R_2. When assuming a threshold of 10% of P's size (i.e., 0.4), no rule would be filtered out.

The output of this step would consequently be $R_1 = \{(<\text{“van”}\rightarrow\text{“Van”}>, 2.08), (<\text{“T.”}\rightarrow \epsilon >, 2)\}$ and $R_2 = \{(<\text{“(actor)”}\rightarrow \epsilon >, 2)\}$.

3.4 Rule Falsification

The aim to the rule falsification step is to detect a set of transformations that lead to a minimal number of elements of N having a similarity superior to θ via σ. To achieve this goal, we follow a greedy approach that aims to minimize the magnitude of the set

$$E = \{(p_1, p_2) \in N : \sigma(\varphi_{R_1}(p_1), \varphi_{R_2}(p_2)) \geq \theta = \min_{(p_1,p_2)\in P} \sigma(\varphi_{R_1}(p_1), \varphi_{R_2}(p_2))\}. \tag{4}$$

Our approach simply tries to discard all rules that apply to elements of E by ascending score. If E is then empty, the approach terminates. If E does not get smaller, then the change is rolled back and then the next rule is tried. Else, the rule is discarded from the set of final rules. Note that discarding a rule can alter the value of θ and thus E. Once the set E has been computed, CaRLA concludes its computation by generating a final value of the threshold θ.

In our example, two rules apply to the element of N. After discarding the rule $<\text{“T.”}\rightarrow \epsilon >$, the set E becomes empty, leading to the termination of the rule falsification step. The final set of rules are thus $R_1 = \{<\text{“van”}\rightarrow\text{“Van”}>\}$ and $R_2 = \{<\text{“(actor)”}\rightarrow \epsilon >\}$. The value of θ is computed to be 0.75. Table 2 shows the canonical property values for our toy example. Note that this threshold allows to discard the elements of N as being equivalent property values.

Property value 1	Property value 2	Canonical value
“Jean van Damne”	“Jean Van Damne (actor)”	“Jean Van Damne”
“Thomas T. van Nguyen”	“Thomas Van Nguyen (actor)”	“Thomas T. Van Nguyen”
“Alain Delon”	“Alain Delon (actor)”	“Alain Delon”
“Alain Delon Jr.”	“Alain Delon Jr. (actor)”	“Alain Delon Jr.”
“Claude T. Francois”	“Claude Francois (actor)”	“Claude T. Francois” “Claude Francois”

Table 2. Canonical property values for our example dataset

It is noteworthy that by learning transformation rules, we also found an initial threshold θ for determining the similarity of property values using σ as similarity function. In combination with the canonical forms computed by CaRLA, the configuration (σ, θ) can be used as an initial configuration for Link Discovery frameworks such as LIMES. For example, the smallest Jaccard similarity for the pair of property values for our example lies by 1/3, leading to a precision of 0.71 for a recall of 1 (F-measure: 0.83). Yet, after the computation of the transformation rules, we reach an F-measure of 1 with a threshold of 1. Consequently, the pair (σ, θ) can be used for determining an initial classifier for approaches such as the RAVEN [11] algorithm implemented in LIMES [10].

4 Extension to Active Learning

One of the drawbacks of batch learning approaches is that they often require a large number of examples to generate good models. As our evaluation shows (see Section 5), this drawback also holds for the batch version of CaRLA, as it can easily detect very common rules but sometimes fails to detect rules that apply to less pairs of property values. In the following, we present how this problem can be addressed by extending CaRLA to aCARLA using active learning [16].

Algorithm 1 Overview of aCaRLA

Require: Positive examples P_0
Require: Negative examples N_0
Require: Similarity function σ
Require: Damping factor κ
Require: Score threshold s_{min}
Require: Tokenization function *token*
Require: Maximal number of annotation requests q_{total}
Require: Number of questions/iteration q
  Rule sets $R_1 \leftarrow \emptyset$, $R_2 \leftarrow \emptyset$
  $q_{current} := 0$ //Current number of questions
  $Ex := 0$ //Set of examples to annotate
  $t := 0$ //Iteration counter
  $r := null$ //unsure rule
  $B := \emptyset$ //set of banned rules
  while $q_{current} \leq q_{total}$ **do**
    (R_1, R_2, θ) = runCarla(P_t, N_t, σ, κ, S_{min}, *token*) // Run batch learning
    r =getMostUnsureRule($R_1 \cup R_2$, B, $s_m in$)
    if $r \neq null$ **then**
      Ex = computeExamples(r, q)
    else
      Ex = getMostDifferent(q)
    end if
    (P, N) = requestAnnotations(Ex)
    $P_{t+1} \leftarrow P_t \cup P$
    $N_{t+1} \leftarrow N_t \cup N$
    $B \leftarrow$ updateBannedRules(B, r)
    $t \leftarrow t + 1$
    $q_{current} \leftarrow q_{current} + |Ex|$
  end while
  (R_1, R_2, θ) := runCarla(P_t, N_t, σ, κ, S_{min}, *token*)

  return (R_1, R_2, θ)

An overview of aCaRLA is given in Algorithm 1. The basic idea here is to begin with small training sets P_0 and N_0. In each iteration, all the available training data is used by the batch version of CaRLA to update the set of rules.

The algorithm then tries to refute or validate rules with a score below the score threshold s_{min} (i.e., unsure rules). For this purpose it picks the most unsure rule r that has not been shown to be erroneous in a previous iteration (i.e., that is not an element of the set of banned rules B). It then fetches a set Ex of property values that map the left side (i.e., the premise) of r. Should there be no unsure rule, then Ex is set to the q property values that are most dissimilar to the already known property values. Annotations consisting of the corresponding values for the elements of Ex in the other source knowledge bases are requested by the user and written in the set P. Property values with no corresponding values are written in N. Finally the sets of positive and negative examples are updated and the triple (R_1, R_2, θ) is learned anew until a stopping condition such as a maximal number of questions is reached. As our evaluation shows, this simple extension of the CaRLA algorithm allows it to detect efficiently the pairs of annotations that might lead to a larger set of high-quality rules.

5 Evaluation

5.1 Experimental Setup

In the experiments reported in this section, we evaluated CaRLA by two means: First we aimed to measure how well CaRLA could compute transformations created by experts. To achieve this goal, we retrieved transformation rules from four link specifications defined manually by experts within the LATC project[5]. An overview of these specifications is given in Table 3. Each link specification aimed to compute `owl:sameAs` links between entities across two knowledge bases by first transforming their property values and by then computing the similarity of the entities based on the similarity of their property values. For example, the computation of links between films in DBpedia and LinkedMDB was carried out by first applying the set of $R_1 = \{< (film) \rightarrow \epsilon >\}$ to the labels of films in DBpedia and $R_2 = \{< (director) \rightarrow \epsilon >\}$ to the labels of their directors. We ran both CaRLA and aCaRLA on the property values of the interlinked entities and measured how fast CaRLA was able to reconstruct the set of rules that were used during the Link Discovery process.

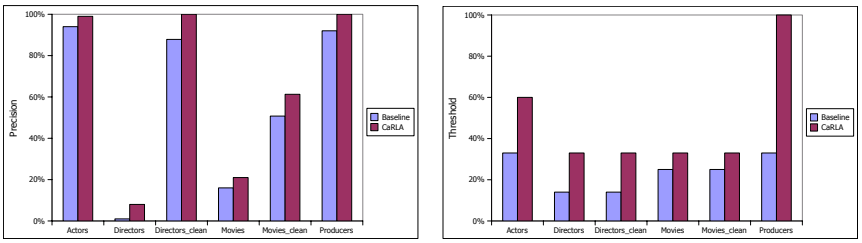
Experiment	Source	Target	Source property	Target property	Size
Actors	DBpedia	LinkedMDB	`rdfs:label`	`rdfs:label`	1172
Directors	DBpedia	LinkedMDB	`rdfs:label`	`rdfs:label`	7353
Movies	DBpedia	LinkedMDB	`rdfs:label`	`rdfs:label`	9859
Producers	DBpedia	LinkedMDB	`rdfs:label`	`rdfs:label`	1540

Table 3. Overview of the datasets

In addition, we quantified the quality of the rules learned by CaRLA. In each experiments, we computed the boost in the precision of the mapping of property pairs with and without the rules derived by CaRLA. The initial precision was

[5] http://latc-project.eu

computed as $\frac{|P|}{|M|}$, where $M = \{(p_i, p_j) : \sigma(p_i, p_j) \geq \min_{(p_1,p_2)\in P} \sigma(p_1, p_2)\}$. The precision after applying CaRLA's results was computed as $\frac{|P|}{|M'|}$ where $M' = \{(p_i, p_j) : \sigma(\varphi_{R_1}(p_i), \varphi_{R_2}(p_j)) \geq \min_{(p_1,p_2)\in P} \sigma(\varphi_{R_1}(p_1), \varphi_{R_2}(p_2))\}$. Note that in both cases, the recall was 1 given that $\forall (p_i, p_j) \in P : \sigma(p_i, p_j) \geq \min_{(p_1,p_2)\in P} \sigma(p_1, p_2)$. In all experiments, we used the Jaccard similarity metric and a word tokenizer with $\kappa = 0.8$. All runs were carried on a notebook running Windows 7 Enterprise with 3GB RAM and an Intel Dual Core 2.2GHz processor. Each of the algorithms was ran five times. We report the rules that were discovered by the algorithms and the number of experiments within which they were found.



(a) Precision (b) Thresholds

Fig. 1. Comparison of the precision and thresholds with and without CaRLA.

5.2 Results and Discussion

Table 4 shows the union of the rules learned by the batch version of CaRLA in all five runs. Note that the computation of a rule set lasted under 0.5s even for the largest dataset, Movies. The columns P_n give the probability of finding a rule for a training set of size n in our experiments. R_2 is not reported because it was empty in all setups. Our results show that in all cases, CaRLA converges quickly and learns rules that are equivalent to those utilized by the LATC experts with a sample set of 5 pairs. Note that for each rule of the form $<$"@en" $\rightarrow y>$ with $y \neq \epsilon$ that we learned, the experts used the rule $< y \rightarrow \epsilon >$ while the linking platform automatically removed the language tag. We experimented with the same datasets without language tags and computed exactly the same rules as those devised by the experts. In some experiments (such as Directors), CaRLA was even able detect rules that where not included in the set of rules generated by human experts. For example, the rule $<$"(filmmaker)" $\rightarrow$ "(director)"$>$ is not very frequent and was thus overlooked by the experts. In Table 4, we marked such rules with an asterisk. The director and the movies datasets contained a large number of typographic errors of different sort (incl. misplaced hyphens, character repetitions such as in the token "Neilll", etc.) which led to poor precision scores in our experiments. We cleaned the first 250 entries of these datasets

from these errors and obtained the results in the rows labels Directors_clean and Movies_clean. The results of CaRLA on these datasets are also shown in Table 4. We also measured the improvement in precision that resulted from applying CaRLA to the datasets at hand (see Figure 1). For that the precision remained constant across the different dataset sizes. In the best case (cleaned Directors dataset), we are able to improve the precision of the property mapping by 12.16%. Note that we can improve the precision of the mapping of property values even on the noisy datasets.

Experiment	R_1	P_5	P_{10}	P_{20}	P_{50}	P_{100}
Actors	$<$"@en" $\rightarrow$ "(actor)" $>$	1	1	1	1	1
Directors	$<$"@en" $\rightarrow$ "(director)" $>$	1	1	1	1	1
Directors	$<$"(filmmaker)" $\rightarrow$ "(director)"$>^*$	0	0	0	0	0.2
Directors_clean	$<$"@en" $\rightarrow$ "(director)" $>$	1	1	1	1	1
Movies	$<$"@en" $\rightarrow \epsilon >$	1	1	1	1	1
Movies	$<$"(film)" $\rightarrow \epsilon >$	1	1	1	1	1
Movies	$<$"film)" $\rightarrow \epsilon >^*$	0	0	0	0	0.6
Movies_clean	$<$"@en" $\rightarrow \epsilon >$	1	1	1	1	1
Movies_clean	$<$"(film)" $\rightarrow \epsilon >$	0	0.8	1	1	1
Movies_clean	$<$"film)" $\rightarrow \epsilon >^*$	0	0	0	0	1
Producers	$<$"@en" $\rightarrow$ (producer)$>$	1	1	1	1	1

Table 4. Overview of batch learning results

Interestingly, when used on the Movies dataset with a training dataset size of 100, our framework learned low-confidence rules such as $<$"(1999" $\rightarrow \epsilon >$, which were yet discarded due to a too low score. These are the cases where aCaRLA displayed its superiority. Thanks to its ability to ask for annotation when faced with unsure rules, aCaRLA is able to validate or negate unsure rules. As the results on the Movies example show, aCaRLA is able to detect several supplementary rules that were overlooked by human experts. Especially, it clearly shows that deleting the year of creation of a movie can improve the conformation process. aCaRLA is also able to generate a significantly larger number of candidates rules for the user's convenience.

6 Related Work

Linked Data Integration is an important topic for all applications that rely on a large number of knowledge bases and necessitate a unified view on this data, e.g., Question Answering frameworks [4] and semantic mashups [13]. In recent work, several challenges and requirements to Linked Data consumption and integration have been pointed out [9]. Recently, several approaches and frameworks have been developed with the aim of addressing many of these challenges. For example, the R2R framework [2] allows the specification and publication of mappings

Experiment	R_1	P_5	P_{10}	P_{20}	P_{50}	P_{100}
Actors	<"@en" → "(actor)" >	1	1	1	1	1
Directors	<"@en" → "(director)" >	1	1	1	1	1
Directors	<"(actor)" → "(director)">*	0	0	0	0	1
Directors_clean	<"@en" → "(director)" >	1	1	1	1	1
Movies	<"@en" → ϵ >	1	1	1	1	1
Movies	<"(film)" → ϵ >	1	1	1	1	1
Movies	<"film)" → ϵ >*	0	0	0	0	1
Movies	<"(2006" → ϵ >*	0	0	0	0	1
Movies	<"(199" → ϵ >*	0	0	0	0	1
Movies_clean	<"@en" → ϵ >	1	1	1	1	1
Movies_clean	<"(film)" → ϵ >	0	1	1	1	1
Movies_clean	<"film)" → ϵ >*	0	0	0	0	1
Producers	<"@en" → (producer)>	1	1	1	1	1

Table 5. Overview of active learning results

that allow mapping resources and literals across knowledge bases. The Linked Data Integration Framework LDIF [15], whose goal is to support the integration of RDF data, builds upon R2R mappings and technologies such as SILK [5] and LDSpider[6]. The concept behind the framework is to enable users to create periodic integration jobs via simple XML configurations. KnoFuss [12] addresses data integration from the point of view of link discovery by monitoring the interaction between instance and dataset matching (which is similar to ontology matching [3]). Also worth mentioning id Semantic Web Pipes[7] [13], which follows the idea of Yahoo Pipes[8] to enable the integration of data in formats such as RDF and XML. In all of these systems, the configuration of the data transformations has to be carried out manually.

Some approaches to transformation rule learning approaches have previously been proposed in the record linkage area. For example, [14] presents an interactive approach to data cleaning while [1] developed an approach to learn string transformation from examples. Yet, none of these approaches uses active learning to improve the number of rules it detects. To the best of our knowledge, CaRLA is the first approach tailored towards Linked Data that allows the active learning of data conformation rules for property values expressed as strings.

7 Conclusion and Future Work

In this work, we presented two algorithms for learning data transformations. We present a batch learning approach that allows to detect rules by performing a co-occurrence analysis. This version is particularly useful when the knowledge bases to be integrated are already linked. We also present an active learning approach

[6] http://code.google.com/p/ldspider/
[7] http://pipes.deri.org/
[8] http://pipes.yahoo.com/pipes/

that allows to discover a large number of rules efficiently. We evaluated our approach with respect to the quality of the rules it discovers and to the effect of the rules on matching property values. We showed that the data transformation achieved by our approach allows to increase the precision of property mapping by up to 12% when the recall is set to 1. In future work, we aim to extend our approach to learning transformation rules with several tokenizers concurrently.

References

1. Arvind Arasu, Surajit Chaudhuri, and Raghav Kaushik. Learning string transformations from examples. *Proc. VLDB Endow.*, 2(1):514–525, August 2009.
2. Christian Bizer and Andreas Schultz. The R2R Framework: Publishing and Discovering Mappings on the Web. In *Proceedings of COLD*, 2010.
3. Jérôme Euzenat and Pavel Shvaiko. *Ontology matching*. Springer-Verlag, Heidelberg (DE), 2007.
4. David A. Ferrucci, Eric W. Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya Kalyanpur, Adam Lally, J. William Murdock, Eric Nyberg, John M. Prager, Nico Schlaefer, and Christopher A. Welty. Building watson: An overview of the deepqa project. *AI Magazine*, 31(3):59–79, 2010.
5. Robert Isele and Christian Bizer. Learning Linkage Rules using Genetic Programming. In *Sixth International Ontology Matching Workshop*, 2011.
6. Paul Jaccard. Étude comparative de la distribution florale dans une portion des Alpes et des Jura. *Bulletin del la Société Vaudoise des Sciences Naturelles*, 37:547–579, 1901.
7. Ralph Kimball and Joe Caserta. *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data*. Wiley, 2004.
8. Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. Technical Report 8, 1966.
9. Ian Millard, Hugh Glaser, Manuel Salvadores, and Nigel Shadbolt. Consuming multiple linked data sources: Challenges and experiences. In *First International Workshop on Consuming Linked Data*, 2010.
10. Axel-Cyrille Ngonga Ngomo. A Time-Efficient Hybrid Approach to Link Discovery. In *Sixth International Ontology Matching Workshop*, 2011.
11. Axel-Cyrille Ngonga Ngomo, Jens Lehmann, Sören Auer, and Konrad Höffner. RAVEN – Active Learning of Link Specifications. In *Proceedings of OM@ISWC*, 2011.
12. Andriy Nikolov, Victoria Uren, Enrico Motta, and Anne Roeck. Overcoming schema heterogeneity between linked semantic repositories to improve coreference resolution. In *Proceedings of the 4th Asian Conference on The Semantic Web*, pages 332–346, 2009.
13. Danh Le Phuoc, Axel Polleres, Manfred Hauswirth, Giovanni Tummarello, and Christian Morbidoni. Rapid prototyping of semantic mash-ups through semantic web pipes. In *WWW*, pages 581–590, 2009.
14. Vijayshankar Raman and Joseph M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB*, pages 381–390, 2001.
15. Andreas Schultz, Andrea Matteini, Robert Isele, Christian Bizer, and Christian Becker. Ldif - linked data integration framework. In *COLD*, 2011.
16. Burr Settles. Active learning literature survey. Technical Report 1648, University of Wisconsin-Madison, 2009.