RAVEN – Active Learning of Link Specifications

Axel-Cyrille Ngonga Ngomo, Jens Lehmann, Sören Auer, Konrad Höffner

Department of Computer Science, University of Leipzig
Johannisgasse 26, 04103 Leipzig, Germany
{ngonga|lehmann|auer}@informatik.uni-leipzig.de,
konrad.hoeffner@uni-leipzig.de
WWW home page: http://aksw.org

Abstract. With the growth of the Linked Data Web, time-efficient approaches for computing links between data sources have become indispensable. Yet, in many cases, determining the right specification for a link discovery problem is a tedious task that must still be carried out manually. We present RAVEN, an approach for the semi-automatic determination of link specifications. Our approach is based on the combination of stable solutions of matching problems and active learning with the time-efficient link discovery framework LIMES. RAVEN aims at requiring a small number of interactions with the user to generate classifiers of high accuracy. We focus on using RAVEN to compute and configure boolean and weighted classifiers, which we evaluate in three experiments against link specifications created manually. Our evaluation shows that we can compute linking configurations that achieve more than 90% F-score by asking the user to verify at most twelve potential links.

Keywords: Linked Data, Link Discovery, Algorithms, Constraints

1 Introduction

The core idea behind the Linked Data paradigm is to facilitate the transition from the document-oriented to the Semantic Web by extending the current Web with a commons of interlinked data sources [2]. One of the key challenges that arise when trying to discover links between two data sources lies in the *specification* of an appropriate configuration for the tool of choice [10]. Such a specification usually consists of a set of restrictions on the source and target knowledge base, a list of properties of the source and target knowledge base to use for similarity detection, a combination of suitable similarity measures (e.g., Levenshtein [9]) and similarity thresholds. Specifying link configurations is a tedious process, as the user does not necessarily know which combinations of properties lead to an accurate linking configuration. The difficulty of devising suitable link discovery specifications is amplified on the Web of Data by the *sheer size* of the knowledge bases (which often contain millions of instances) and their *heterogeneity* (i.e., by the complexity of the underlying ontologies, which can contain thousands of different types of instances and properties) [2].

In this paper, we present the RApid actiVE liNking (RAVEN) approach. RAVEN is the first approach to apply active learning techniques for the semi-automatic detection of specifications for link discovery. Our approach is based on a combination of stable matching and a novel active learning algorithm derived from perceptron learning. RAVEN allows to determine (1) a sorted matching of classes to interlink; this matching represents the set of restrictions of the source and target knowledge bases, (2) a stable matching of properties based on the selected restrictions that specifies the similarity space within which the linking is to be carried out and (3) a highly accurate link specification via active learning. Our evaluation with three series of experiments shows that we can compute linking configurations that achieve more than 90% F-score by asking the user to verify at most twelve potential links. RAVEN is generic enough to be employed with any link discovery framework that supports complex link specifications. The results presented herein rely on the LIMES framework for linking. We chose LIMES because it implements lossless approaches and is very time-efficient.

2 Related Work

Current approaches for link discovery on the Web of Data can be subdivided into two categories: *domain-specific* and *universal* approaches. Domain-specific link discovery frameworks aim at discovering links between knowledge bases from a particular domain. For example, the approach implemented in RKB knowledge base (RKB-CRS) [5] focuses on computing links between universities and conferences while GNAT [12] discovers links between music data sets. Further simple or domain-specific approaches can be found in [7, 17, 11].

Universal link discovery frameworks are designed to carry out mapping tasks independently from the domain of the source and target knowledge bases. For example, RDF-AI [15] implements a five-step approach that comprises the preprocessing, matching, fusion, interlinking and post-processing of data sets. SILK [18] is a time-optimized tool for link discovery. It implements a multi-dimensional blocking approach that projects the instances to match in a multi-dimensional metric space. Subsequently, this space is subdivided into to overlapping blocks that are used to retrieve matching instances without loosing links. Another lossless Link Discovery framework is LIMES [10], which addresses the scalability problem by utilizing the *triangle inequality* in metric spaces to compute pessimistic estimates of instance similarities. Based on these approximations, LIMES can filter out a large number of non-matches.

The task of discovering links between knowledge bases is closely related with record linkage and deduplication [3]. The database community has produced a vast amount of literature on efficient algorithms for solving these problems. Different blocking techniques such as standard blocking, sorted-neighborhood, bi-gram indexing, canopy clustering and adaptive blocking (see, e.g., [8]) have been developed to address the problem of the quadratic time complexity of brute force comparison methods. Active learning has been employed in the database community [13, 14, 1] to address the configuration problem because active learn-

ing approaches usually present only few match candidates to the user for manual verification. The technique is particularly efficient in terms of required user input [16], because the user is only confronted with those match candidates which provide a high benefit for the underlying learning algorithm.

The RAVEN approach goes beyond the state of the art in several ways: It is the first active learning algorithm and RDF-based approach to use machine learning for obtaining link specifications. Moreover, it is the first approach to detect corresponding classes and properties automatically for the purpose of Link Discovery. Note that similar approaches developed for databases assume the mapping of columns to be known [1]. Yet, this assumption cannot be made when trying to link knowledge bases from the Web of Data because of the possible size of the underlying ontology. By supporting the automatic detection of links, we are able to handle heterogeneous knowledge bases with large schemata.

3 Preliminaries

Our approach to the active learning of linkage specifications extends ideas from several research areas, especially classification and stable matching problems. In the following, we present the notation that we use throughout this paper and explain the theoretical framework underlying our work.

3.1 Problem Definition

The link discovery problem, which is similar to the record linkage problem, is an ill-defined problem and is consequently difficult to model formally [1]. In general, link discovery aims to discover pairs $(s, t) \in S \times T$ related via a relation R.

Definition 1 (Link Discovery). *Given two sets S (source) and T (target) of entities, compute the set $\mathcal{M}$ of pairs of instances $(s, t) \in S \times T$ such that $R(s, t)$.*

The sets S resp. T are usually subsets of the instances contained in two knowledge bases $\mathcal{K}_S$ resp. $\mathcal{K}_T$. In most cases, the computation of whether $R(s, t)$ holds for two elements is carried out by projecting the elements of S and T based on their properties in a similarity space $\mathfrak{S}$ and setting $R(s, t)$ iff some similarity condition is satisfied. The specification of the sets S and T and of this similarity condition is usually carried out within a *link specification*.

Definition 2 (Link Specification). *A link specification consists of three parts: (1) two sets of restrictions $\mathcal{R}_1^S$ ... $\mathcal{R}_m^S$ resp. $\mathcal{R}_1^T$ ... $\mathcal{R}_k^T$ that specify the sets S resp. T, (2) a specification of a complex similarity metric σ via the combination of several atomic similarity measures σ_1, ..., σ_n and (3) a set of thresholds τ_1, ..., τ_n such that τ_i is the threshold for σ_i.*

A restriction $\mathcal{R}$ is generally a logical predicate. Typically, restrictions in link specifications state the `rdf:type` of the elements of the set they describe, i.e., $\mathcal{R}(x) \leftrightarrow x$ `rdf:type someClass` or the features the elements of the set must

have, e.g., $\mathcal{R}(x) \leftrightarrow (\exists y : x \texttt{ someProperty } y)$. Each $s \in S$ must abide by each of the restrictions $\mathcal{R}_1^S$ ... $\mathcal{R}_m^S$, while each $t \in T$ must abide by each of the restrictions $\mathcal{R}_1^T$ ... $\mathcal{R}_k^T$. Note that the atomic similarity functions $\sigma_1, ..., \sigma_n$ can be combined to σ by different means. In this paper, we will focus on using boolean operators and real weights combined as conjunctions.

According to the formalizations of link discovery and link specifications above, finding matching pairs of entities can be defined equivalently as a classification task, where the classifier $\mathcal{C}$ is a function from $S \times T$ to $\{-1, +1\}$.

Definition 3 (Link Discovery as Classification). *Given the set $S \times T$ of possible matches, the goal of link discovery is to find a classifier $\mathcal{C} : S \times T \rightarrow \{-1, +1\}$ such that $\mathcal{C}$ maps non-matches to the class -1 and matches to $+1$.*

In general, we assume classifiers that operate in an n-dimensional similarity space $\mathfrak{S}$. Each of the dimensions of $\mathfrak{S}$ is defined by a similarity function σ_i that operates on a certain pair of attributes of s and t. Each classifier $\mathcal{C}$ on $\mathfrak{S}$ can be modeled via a specific function $\mathcal{F}_\mathcal{C}$. $\mathcal{C}$ then returns $+1$ iff the logical statement $\mathcal{P}_\mathcal{C}(\mathcal{F}_\mathcal{C}(s,t))$ holds and -1 otherwise, where $\mathcal{P}_\mathcal{C}$ is what we call the specific predicate of $\mathcal{C}$. In this work, we consider two families of classifiers: *linear weighted* classifiers $\mathcal{L}$ and *boolean conjunctive* classifiers $\mathcal{B}$. The specific function of linear weighted classifiers is of the form

$$\mathcal{F}_\mathcal{L}(s,t) = \sum_{i=1}^{n} \omega_i \sigma_i(s,t), \tag{1}$$

where $\omega_i \in \mathbb{R}$. The predicate $\mathcal{P}_\mathcal{L}$ for a linear classifier is of the form $\mathcal{P}_\mathcal{L}(X) \leftrightarrow (X \geq \tau)$, where $\tau = \tau_1 = ... = \tau_n \in [0,1]$ is the similarity threshold. A boolean classifier $\mathcal{B}$ is a conjunction of n atomic linear classifiers $\mathcal{C}_1$, ... ,$\mathcal{C}_n$, i.e., a conjunction of classifiers that each operate on exactly one of the n dimensions of the similarity space $\mathfrak{S}$. Thus, the specific function $\mathcal{F}_\mathcal{B}$ is as follows:

$$\mathcal{F}_\mathcal{B}(s,t) = \bigwedge_{i=0}^{n} (\sigma_i(s,t) \geq \tau_i) \tag{2}$$

and the specific predicate is simply $\mathcal{P}_\mathcal{B}(X) = X$. Note that given that classifiers are usually learned by using iterative approaches, we will denote classifiers, weights and thresholds at iteration t by using superscripts, i.e., $\mathcal{C}^t$, ω_i^t and τ_i^t.

Current approaches to learning in record matching assume that the similarity space $\mathfrak{S}$ is given. While this is a sensible premise for mapping problems which rely on simple schemas, the large schemas (i.e., the ontologies) that underlie many data sets in the Web of Data do not allow such an assumption. The DBpedia ontology (version 3.6) for example contains 275 classes and 1335 properties. Thus, it would be extremely challenging for a user to specify the properties to map when carrying out a simple deduplication analysis, let alone more complex tasks using the DBpedia data set. In the following, we give a brief overview of stable matching problems, which we use to solve the problem of suggesting appropriate sets of restrictions on data and matching properties.

3.2 Stable Matching Problems

The best known stable matching problem is the stable marriage problem $\mathcal{SM}$ as formulated by [4]. Here, one assumes two sets M (males) and F (females) such that $|M| = |F|$ and two functions $\mu : M \times F \rightarrow \{1, ..., |F|\}$ resp. $\gamma : M \times F \rightarrow \{1, ..., |M|\}$, that give the degree to which a male likes a female resp. a female a male. $\mu(m, f) > \mu(m, f')$ means that m prefers f to f'. Note, that for all f and f' where $f \neq f'$ holds, $\mu(m, f) \neq \mu(m, f')$ must also hold. Analogously, $m \neq m'$ implies $\gamma(m, f) \neq \gamma(m', f)$. A bijective function $s : M \rightarrow F$ is called a stable matching iff for all m, m', f, f' the following holds:

$$(s(m) = f) \wedge (s(m') = f') \wedge (\mu(m, f') > \mu(m, f)) \rightarrow (\gamma(m', f') > \gamma(m, f')) \quad (3)$$

In [4] an algorithm for solving this problem is presented and it is shown how it can be used to solve the well-know Hospital/Residents ($\mathcal{HR}$) problem. Formally, $\mathcal{HR}$ extends $\mathcal{SM}$ by assuming a set R of residents (that maps M) and a set of hospitals (that maps F) with $|R| \neq |F|$. Each hospital $h \in H$ is assigned a capacity $c(h)$. A stable solution of the Hospital/Residents problem is a mapping of residents to hospitals such that each hospital accepts maximally $c(h)$ residents and that fulfills Equation 3. Note that we assume that there are no ties, i.e., that the functions μ and γ are injective.

4 The RAVEN Approach

Our approach, dubbed RAVEN (RApid actiVE liNking), addresses the task of linking two knowledge bases S and T by using the active learning paradigm within the pool-based sampling setting [16]. Overall, the goal of RAVEN is to find the best classifier $\mathcal{C}$ that achieves the highest possible precision, recall or F_1 score as desired by the user. The algorithm also aims to minimize the burden on the user by limiting the number of link candidates that must be labeled by the user to a minimum.

Algorithm 1 The RApid actiVE liNking (RAVEN) algorithm

Require: Source knowledge base $\mathcal{K}_S$
Require: Target knowledge base $\mathcal{K}_T$
  Find stable class matching between classes of $\mathcal{K}_S$ and $\mathcal{K}_T$
  Find stable property matching for the selected classes
  Compute sets S and T; Create initial classifier $\mathcal{C}^0$; $t := 0$
  while termination condition not satisfied **do**
    Ask the user to classify 2α examples; Update $\mathcal{C}^t$ to $\mathcal{C}^{t+1}$; t := t+1
  end while
  Compute set $\mathcal{M}$ of links between S and T based on $\mathcal{C}^t$
  return $\mathcal{M}$

An overview of our approach is given in Algorithm 1. In a first step, RAVEN aims to detect the restrictions that will define the sets S and T. To achieve this

goal, it tries to find a stable matching of pairs of classes, whose instances are to be linked. The second step of our approach consists of finding a stable matching between the properties that describe the instances of the classes specified in the first step. The user is also allowed to alter the suggested matching at will. Based on the property mapping, we compute S and T and generate an initial classifier $\mathcal{C} = \mathcal{C}^0$ in the third step. We then refine $\mathcal{C}$ iteratively by asking the user to classify pairs of instances that are most informative for our algorithm. $\mathcal{C}$ is updated until a termination condition is reached, for example $\mathcal{C}^t = \mathcal{C}^{t+1}$. The final classifier is used to compute the links between S and T, which are returned by RAVEN. In the following, we expand upon each of these three steps.

4.1 Stable Matching of Classes

The first component of a link specification is a set of restrictions that must be fulfilled by the instances that are to be matched. We use a two-layered approach for matching classes in knowledge bases. Our *default approach* begins by selecting a user-specified number of `sameAs` links between the source and the target knowledge base randomly. Then, it computes μ and γ on the classes C_S of $\mathcal{K}_S$ and C_T of $\mathcal{K}_T$ as follows[1]:

$$\mu(C_S, C_T) = \gamma(C_S, C_T) = |\{s \texttt{ type } C_s \wedge s \texttt{ sameAs } t \wedge t \texttt{ type } C_T\}|. \quad (4)$$

In the case when no `sameAs` links are available, we run our *fallback* approach. It computes μ and γ on the classes of S and T as follows:

$$\mu(C_S, C_T) = \gamma(C_S, C_T) = |\{s \texttt{ type } C_s \wedge s \texttt{ p } x \wedge t \texttt{ type } C_T \wedge t \texttt{ q } x\}|, \quad (5)$$

where `p` and `q` can be any property. Let $c(S)$ be the number of classes C_S of S such that $\mu(C_S, C_T) > 0$ for any C_T. Furthermore, let $c(T)$ be the number of classes C_T of T such that $\gamma(C_S, C_T) > 0$ for any C_S. The capacity of each C_T is set to $\lceil c(S)/c(T) \rceil$, thus ensuring that the hospitals provide enough capacity to map all the possible residents. Once μ, γ and the capacity of each hospital has been set, we solve the equivalent $\mathcal{HR}$ problem.

4.2 Stable Matching of Properties

The detection of the best matching pairs of properties is very similar to the computation of the best matching classes. For properties `p` and `q`, we set:

$$\mu(p, q) = \gamma(p, q) = |\{s \texttt{ type } C_s \wedge s \texttt{ p } x \wedge t \texttt{ type } C_T \wedge t \texttt{ q } x\}|. \quad (6)$$

The initial mapping of properties defines the similarity space in which the link discovery task will be carried out. Note that none of the prior approaches to active learning for record linkage or link discovery automatized this step. We associate each of the basis vectors σ_i of the similarity space to exactly one of the pairs (`p`, `q`) of mapping properties. Once the restrictions and the property mapping have been specified, we can fetch the elements of the sets S and T.

[1] Note that we used `type` to denote `rdf:type` and `sameAs` to denote `owl:sameAs`.

4.3 Initial Classifier

The specific formula for the initial linear weighted classifier $\mathcal{L}^0$ results from the formal model presented in Section 3 and is given by

$$\mathcal{F_L}^0(s,t) = \sum_{i=1}^{n} \omega_i^0 \sigma_i(s,t). \tag{7}$$

Several initialization methods can be used for ω_i^0 and the initial threshold τ^0 of $\mathcal{P_L}$. In this paper, we chose to use the simplest possible approach by setting $\omega_i^0 := 1$ and $\tau^0 := \kappa n$, where $\kappa \in [0,1]$ is a user-specified *threshold factor*. Note that setting the overall threshold to κn is equivalent to stating that the arithmetic mean of the $\sigma_i(s,t)$ must be equal to κ.

The equivalent initial boolean classifier $\mathcal{B}^0$ is given by

$$\mathcal{F_B}^0(s,t) = \bigwedge_{i=0}^{n} (\sigma_i^0(s,t) \geq \tau_i^0) \text{ where } \tau_i^0 := \kappa. \tag{8}$$

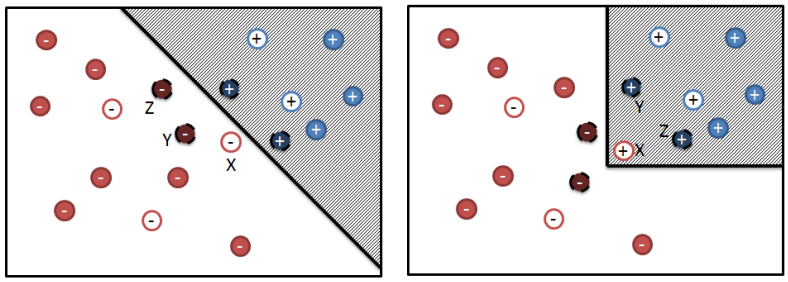
4.4 Updating Classifiers

RAVEN follows an iterative update strategy, which consists of asking the user to classify 2α elements of $S \times T$ (α is explained below) in each iteration step t and using these to update the values of ω_i^{t-1} and τ_i^{t-1} computed at step $t-1$. The main requirements to the update approach is that it computes those elements of $S \times T$ whose classification allow to maximize the convergence of $\mathcal{C}^t$ to a good classifier and therewith to minimize the burden on the user. The update strategy of RAVEN varies slightly depending on the family of classifiers. In the following, we present how RAVEN updates linear and boolean classifiers.

Updating linear classifiers. The basic intuition behind our update approach is that we aim to present the user with those elements from $S \times T$ whose classification is most unsure. We call the elements presented to the user *examples*. We call an example *positive* when it is assumed by the classifier to belong to $+1$. Else we call it *negative*. Once the user has provided us with the correct classification for all examples, the classifier can be updated effectively so as to better approximate the target classifier. In the following, we will define the notion of *most informative example* for linear classifiers before presenting our update approach.

When picturing a classifier as a boundary in the similarity space $\mathfrak{S}$ that separates the classes $+1$ and -1, the examples whose classification is most uncertain are those elements from $S \times T$ who are closest to the boundary. Note that we must exclude examples that have been classified previously, as presenting them to the user would not improve the classification accuracy of RAVEN while generating extra burden on the user, who would have to classify the same link candidate twice. Figure 1 depicts the idea behind most informative examples. In both subfigures, the circles with a dashed border represent the 2 most informative positive and negatives examples, the solid disks represent elements

from $S \times T$ and the circles are examples that have already been classified by the users. Note that while X is closer to the boundary than Y and Z, it is not a most informative example as it has already been classified by the user.



(a) Linear classifier (b) Boolean classifier

Fig. 1. Most informative examples for linear and boolean classifiers. The current elements of the classes -1 resp. $+1$ are marked with $-$ resp. $+$.

Formally, let $\mathcal{M}^t$ be the set of $(s,t) \in S \times T$ classified by $\mathcal{L}^t$ as belonging to $+1$. Furthermore, let $\mathcal{P}^{t-1}$ (resp. $\mathcal{N}^{t-1}$) be the set of examples that have already been classified by the user as being positive examples, i.e, links (resp. negative examples, i.e., wrong links). We define a set Λ as being a set of most informative examples λ for $\mathcal{L}^{t+1}$ when the following conditions hold:

$$\forall \lambda \in S \times T \ (\lambda \in \Lambda \rightarrow \lambda \notin \mathcal{P}^{t-1} \cup \mathcal{N}^{t-1}) \tag{9}$$

$$\forall \lambda' \notin \mathcal{P}^{t-1} \cup \mathcal{N}^{t-1} : \lambda' \neq \lambda \rightarrow |{\mathcal{F}_{\mathcal{L}}}^t(\lambda') - \tau^t| \geq |{\mathcal{F}_{\mathcal{L}}}^t(\lambda) - \tau^t|. \tag{10}$$

Note that there are several sets of most informative examples of a given magnitude. We denote a set of most informative examples of magnitude α by Λ_α. A set of most informative positive examples, Λ^+, is a set of pairs such that

$$\forall \lambda \notin \Lambda^+ \cup \mathcal{P}^{t-1} \cup \mathcal{N}^{t-1} : ({\mathcal{F}_{\mathcal{L}}}^t(\lambda) < \tau^t) \vee (\forall \lambda^+ \in \Lambda^+ : {\mathcal{F}_{\mathcal{L}}}^t(\lambda) > {\mathcal{F}_{\mathcal{L}}}^t(\lambda^+)). \tag{11}$$

In words, Λ^+ is the set of examples that belong to class $+1$ according to $\mathcal{C}$ and are closest to $\mathcal{C}$'s boundary. Similarly, the set of most informative negative examples, Λ^-, is the set of examples such that

$$\forall \lambda \notin \Lambda^- \cup \mathcal{P}^{t-1} \cup \mathcal{N}^{t-1} : ({\mathcal{F}_{\mathcal{L}}}^t(\lambda) \geq \tau^t) \vee (\forall \lambda^- \in \Lambda^- : {\mathcal{F}_{\mathcal{L}}}^t(\lambda) < {\mathcal{F}_{\mathcal{L}}}^t(\lambda^-)). \tag{12}$$

We denote a set of most informative (resp. negative) examples of magnitude α as Λ^+_α (resp. Λ^-_α). The 2α examples presented to the user consist of the union $\Lambda^+_\alpha \cup \Lambda^-_\alpha$, where Λ^+_α and Λ^-_α are chosen randomly amongst the possible sets of most informative positive resp. negative examples .

The update rule for the weights of $\mathcal{L}^t$ is derived from the well-known Perceptron algorithm, i.e.,

$$\omega_i^{t+1} = \omega_i^t + \eta^+ \sum_{\lambda \in \Lambda^+} \rho(\lambda)\sigma_i(\lambda) - \eta^- \sum_{\lambda \in \Lambda^-} \rho(\lambda)\sigma_i(\lambda), \tag{13}$$

where η^+ is the learning rate for positives examples, η^- is the learning rate for negative examples and $\rho(\lambda)$ is 0 when the classification of λ by the user and $\mathcal{L}^t$ are the same and 1 when they differ.

The threshold is updated similarly, i.e,

$$\tau_i^{t+1} = \tau_i^t + \eta^+ \sum_{\lambda \in \Lambda_\alpha^+} \rho(\lambda) {\mathcal{F}_\mathcal{L}}^t(\lambda) - \eta^- \sum_{\lambda \in \Lambda_\alpha^-} \rho(\lambda) {\mathcal{F}_\mathcal{L}}^t(\lambda). \tag{14}$$

Note that the weights are updated by using the dimension which they describe while the threshold is updated by using the whole specific function. Finally, the sets $\mathcal{P}^{t-1}$ and $\mathcal{N}^{t-1}$ are updated to

$$\mathcal{P}^t := \mathcal{P}^{t-1} \cup \Lambda_\alpha^+ \text{ and } \mathcal{N}^t := \mathcal{N}^{t-1} \cup \Lambda_\alpha^-. \tag{15}$$

Updating boolean classifiers. The notion of *most informative example* differs slightly for boolean classifiers. λ is considered a most informative example for $\mathcal{B}$ when the conditions

$$\lambda \notin \mathcal{P}^{t-1} \cup \mathcal{N}^{t-1} \tag{16}$$

and

$$\forall \lambda' \notin \mathcal{P}^{t-1} \cup \mathcal{N}^{t-1} : \lambda' \neq \lambda \rightarrow \sum_{i=1}^{n} |\sigma_i^t(\lambda') - \tau_i^t| \geq \sum_{i=1}^{n} |\sigma_i^t(\lambda) - \tau_i^t| \tag{17}$$

hold. The update rule for the thresholds τ_i^t of $\mathcal{B}$ is then given by

$$\tau_i^{t+1} = \tau_i^t + \eta^+ \sum_{\lambda \in \Lambda_\alpha^+} \rho(\lambda) \sigma_i(\lambda) - \eta^- \sum_{\lambda \in \Lambda_\alpha^-} \rho(\lambda) \sigma_i(\lambda), \tag{18}$$

where η^+ is the learning rate for positives examples, η^- is the learning rate for negative examples and $\rho(\lambda)$ is 0 when the classification of λ by the user and $\mathcal{C}_{t-1}$ are the same and 1 when they differ. The sets $\mathcal{P}^{t-1}$ and $\mathcal{N}^{t-1}$ are updated as given in Equation 15.

5 Experiments and Results

5.1 Experimental Setup

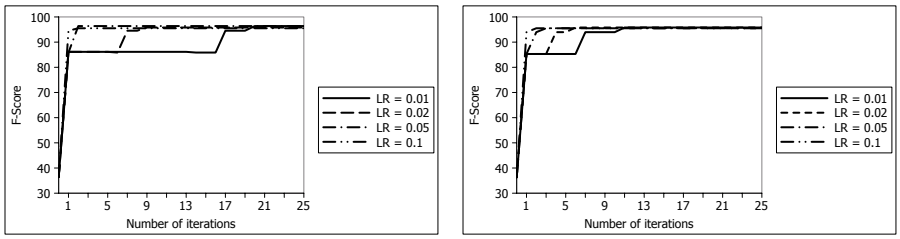
We carried out three series of experiments to evaluate our approach. In our first experiment, dubbed *Diseases*, we aimed to map diseases from DBpedia with diseases from Diseasome. In the *Drugs* experiments, we linked drugs from Sider with drugs from Drugbank. Finally, in the *Side-Effects* experiments, we aimed to link side-effects of drugs and diseases in Sider and Diseasome.

In all experiments, we used the following setup: The learning rates η^+ and η^- were set to the same value η, which we varied between 0.01 and 0.1. We set the number of inquiries per iteration to 4. The threshold factor κ was set to

0.8. In addition, the number of instances used during the automatic detection of class resp. property matches was set to 100 resp. 500. The fallback solution was called and compared the property values of 1000 instances chosen randomly from the source and target knowledge bases. We used the trigrams metric as default similarity measure for strings and the Euclidean similarity as default similarity measure for numeric values. To measure the quality of our results, we used *precision*, *recall* and *F-score*. We also measured the total number of inquiries that RAVEN needed to reach its maximal F-Score. As reference data, we used the set of instances that mapped perfectly according to a configuration created manually.

5.2 Results

The results of our experiments are shown in Figures 2 and 3. The first experiment, Diseases, proved to be the most difficult for RAVEN. Although the `sameAs` links between `Diseasome` and `DBpedia` allowed our experiment to run without making use of the fallback solution, we had to send 12 inquiries to the user when the learning rate was set to 0.1 to determine the best configuration that could be learned by linear and boolean classifiers. Smaller learning rates led to the system having to send even up to 80 inquiries ($\eta = 0.01$) to determine the best configuration. In this experiment linear classifiers outperform boolean classifiers in all setups by up to 0.8% F-score.

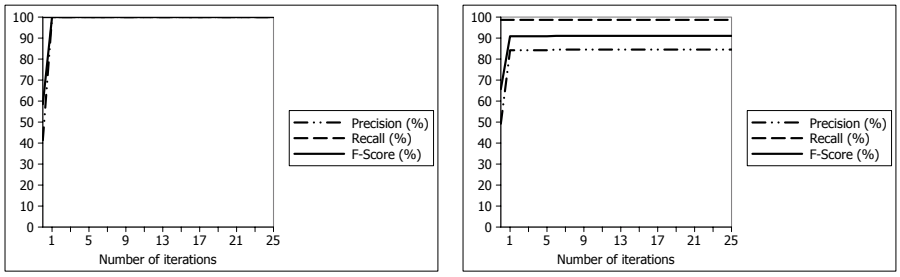


(a) Linear classifier (b) Boolean classifier

Fig. 2. Learning curves on Diseases experiments. LR stands for learning rate.

The second and the third experiment display the effectiveness of RAVEN. Although the fallback solution had to be used in both cases, our approach is able to detect the right configuration with an accuracy of even 100% in the Side-Effects experiment by asking the user no more than 4 questions. This is due to the linking configuration of the user leading to two well-separated sets of instances. In these cases, RAVEN converges rapidly and finds a good classifier rapidly. Note that in these two cases, all learning rates in combination with both linear and boolean classifiers led to the same results (see Figures 3(b) and 3(a)).

Although we cannot directly compare our results to other approaches as it is the first active learning algorithm for learning link specifications, results reported



(a) Learning curve in the Side-Effects experiment

(b) Learning curve in the Drug experiment

Fig. 3. Learning curve in the Side-Effects and Drugs experiments

in the database area suggest that RAVEN achieves state-of-the-art performance. The runtimes required for each iteration ensure that our approach can be used in real-world interactive scenarios. In the worst case, the user has to wait for 1.4 seconds between two iterations. The runtime for the computation of the initial configuration depends heavily on the connectivity to the SPARQL endpoints. In our experiments, the computation of the initial configuration demanded 60s when the default solution was used. The fallback solution required up to 90s.

6 Conclusion and Future Work

In this paper, we presented RAVEN, the first active learning approach tailored towards semi-automatic Link Discovery on the Web of Data. We showed how RAVEN uses stable matching algorithms to detect initial link configurations. We opted to use the solution of the hospital residence problem ($\mathcal{HR}$) without ties because of the higher time complexity of the solution of $\mathcal{HR}$ with ties, i.e., L^4, where L is the size of the longest preference list, i.e., $max(|R|, |H|)$. Still, our work could be extended by measuring the effect of considering ties on the matching computed by RAVEN. Our experimental results showed that RAVEN can compute accurate link specifications (F-score between 90% and 100%) by asking the user to classify a very small number of positive and negative examples (between 4 and 12 for a learning rate of 0.1). Our results also showed that our approach can be used in an interactive scenario because of LIMES' time efficiency, which allowed to compute new links in less than 1.5 seconds in the evaluation tasks. The advantages of this interactive approach can increase the quality of generated links while reducing the effort to create them.

In future work, we will explore how to detect optimal values for the threshold factor κ automatically, for example, by using clustering approaches. In addition, we will investigate the automatic detection of domain-specific metrics that can model the idiosyncrasies of the dataset at hand. Another promising extension to RAVEN is the automatic detection of the target knowledge base to even further simplify the linking process, since users often might not even be aware

of appropriate linking targets (see [6] for research in this area). By these means, we aim to provide the first zero-configuration approach to Link Discovery.

Acknowledgement

This work was supported by the Eurostars grant SCMS E!4604, the EU FP7 grant LOD2 (GA no. 257943) and a fellowship grant of the University of Mainz.

References

1. A. Arasu, M. Götz, and R. Kaushik. On active learning of record matching packages. In *SIGMOD*, pages 783–794, 2010.
2. C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *International Journal on Semantic Web and Information Systems*, 2009.
3. J. Bleiholder and F. Naumann. Data fusion. *ACM Comput. Surv.*, 41(1):1–41, 2008.
4. D. Gale and L. S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.
5. H. Glaser, I. C. Millard, W.-K. Sung, S. Lee, P. Kim, and B.-J. You. Research on linked data and co-reference resolution. Technical report, University of Southampton, 2009.
6. C. Guéret, P. Groth, F. van Harmelen, and S. Schlobach. Finding the achilles heel of the web of data: Using network analysis for link-recommendation. In *ISWC*, pages 289–304, 2010.
7. A. Hogan, A. Polleres, J. Umbrich, and A. Zimmermann. Some entities are more equal than others: statistical methods to consolidate linked data. In *NeFoRS*, 2010.
8. H. Köpcke, A. Thor, and E. Rahm. Comparative evaluation of entity resolution approaches with fever. *Proc. VLDB Endow.*, 2(2):1574–1577, 2009.
9. V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. Technical Report 8, 1966.
10. A.-C. Ngonga Ngomo and S. Auer. Limes - a time-efficient approach for large-scale link discovery on the web of data. In *Proceedings of IJCAI*, 2011.
11. G. Papadakis, E. Ioannou, C. Niedere, T. Palpanasz, and W. Nejdl. Eliminating the redundancy in blocking-based entity resolution methods. In *JCDL*, 2011.
12. Y. Raimond, C. Sutton, and M. Sandler. Automatic interlinking of music datasets on the semantic web. In *1st Workshop about Linked Data on the Web*, 2008.
13. S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *KDD*, pages 269–278, 2002.
14. S. Sarawagi, A. Bhamidipaty, A. Kirpal, and C. Mouli. Alias: An active learning led interactive deduplication system. In *VLDB*, pages 1103–1106, 2002.
15. F. Scharffe, Y. Liu, and C. Zhou. Rdf-ai: an architecture for rdf datasets matching, fusion and interlink. In *Proc. IJCAI 2009 IR-KR Workshop*, 2009.
16. B. Settles. Active learning literature survey. Technical Report 1648, University of Wisconsin-Madison, 2009.
17. J. Sleeman and T. Finin. Computing foaf co-reference relations with rules and machine learning. In *Proceedings of SDoW*, 2010.
18. J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov. Discovering and maintaining links on the web of data. In *ISWC 2009*, pages 650–665. Springer, 2009.