Instance-based 'one-to-some' Assignment of Similarity Measures to Attributes

Tobias Vogel and Felix Naumann

Hasso Plattner Institute, University of Potsdam, Germany
{firstname.lastname}@hpi.uni-potsdam.de

Abstract. Data quality is a key factor for economical success. It is usually defined as a set of properties of data, such as completeness, accessibility, relevance, and conciseness. The latter includes the absence of multiple representations for same real world objects. To avoid such duplicates, there is a wide range of commercial products and customized self-coded software. These programs can be quite expensive both in acquisition and maintenance. In particular, small and medium-sized companies cannot afford these tools. Moreover, it is difficult to set up and tune all necessary parameters in these programs. Recently, web-based applications for duplicate detection have emerged. However, they are not easy to integrate into the local IT landscape and require much manual configuration effort.
With DAQS (Data Quality as a Service) we present a novel approach to support duplicate detection. The approach features (1) minimal required user interaction and (2) self-configuration for the provided input data. To this end, each data cleansing task is classified to find out which metadata is available. Next, similarity measures are automatically assigned to the provided records' attributes and a duplicate detection process is carried out. In this paper we introduce a novel matching approach, called one-to-some or 1:k assignment, to assign similarity measures to attributes. We performed an extensive evaluation on a large training corpus and ten test datasets of address data and achieved promising results.

Keywords: Database Management; Database Applications; Data mining; Matching; Intrinsic; Data Quality; matching, duplicate detection, similarity measures, data cleansing

1 The Need of Data Quality

How would you enter a new costumer who states his (Scandinavian) name as Christopher Quist into a customer database? "Christopher Qvist", "Christofer Kvist", or "Kristoffer Quist"? In fact, there are at least 70 possible ways to spell that name[1]. The affected company will face problems finding the correct record when trying to solve a postal delivery issue. Furthermore, it is impossible to calculate key performance indicators correctly, such as the correct number of customers or the average revenue per customer. Also, the expenses for advertisement mailings are unnecessarily high due

[1] `http://tinyurl.com/optimizeyourdataquality-wp-com`

to sending multiple shipments to the same customer. And last but not least, customer satisfaction suffers when personal data is managed poorly.

Data are often captured by humans. Ignoring the correct spelling, insufficient audio quality on a phone line, stress, typos, poor OCR, or encoding issues are severe problems for the overall correctness of data in customer relationship management (CRM) systems. There are validation tools that check data for soundness to some extent, e. g., syntactical correctness of telephone numbers or existence of postal addresses. However, this is not applicable to correct spellings of names or persons' date-of-birth. Therefore, databases will always contain errors to a certain degree.

Yet there are approaches to manage these deficiencies by being more tolerant in comparing records: Similarity measures are introduced that relax strict comparisons by assigning a real number (typically between 0 and 1) to a pair of records, instead. This number represents the probability that both records represent the same real-world entity.

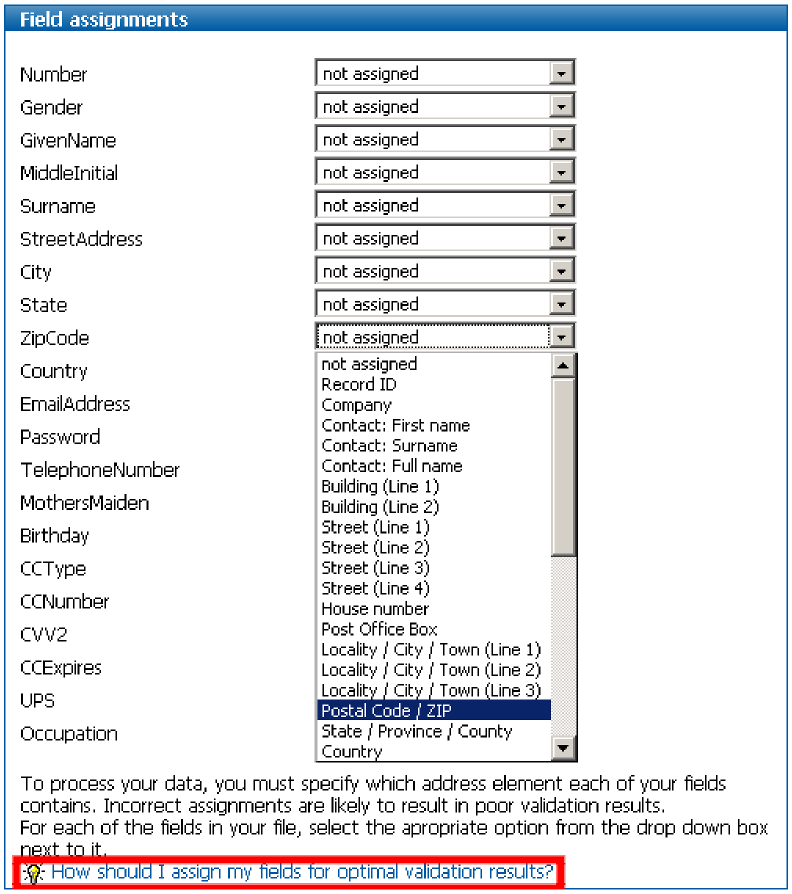
The process of searching through a database and looking for pairs of records that have a high similarity and thereby identifying duplicates is called *duplicate detection*. To perform efficient duplicate detection, the challenge is to develop good similarity measures and to apply them on the corresponding attributes in the data to be cleaned. The mapping of similarity measures to attributes is an assignment problem. For also being efficient in duplicate detection, algorithms for selecting only promising pairs of elements in the provided datasets for comparison are used. Efficiency is out of the scope of this paper.

Traditionally, this assignment is created manually, which is time-consuming and tedious. In addition, data cleansing tools and suites are often expensive. Yet, companies that do not regularly execute duplicate detection and other data cleansing tasks, face competitive disadvantages. The degree of automation in the duplicate detection process increases when those assignments are generated automatically.

Web services are an appropriate model for such on-demand invocation styles. They offer flexible and scalable processing power and are often provided with pay-as-you-go cost models. We define a web service as a piece of software that serves a well-defined purpose, is typically invoked over a network, and is called by other programs rather than human users. Another benefit of the service paradigm comes with the multi-tenancy: the service can learn and improve the assignment quality when being exposed to many different datasets better than it could do in a single client scenario.

While there are providers for these services, e. g., Strike Iron, these are often no true web services but web applications, which involve human interaction, such as cumbersome assigning data types/semantics to uploaded input data as shown in Fig. 1. Therein, a user must manually assign semantics from a long drop-down list to all relevant attributes.

Data that are to be de-duplicated come in various guises. For example, if the data is not in relational format, it is unclear *which* attributes to compare. If the datatype or semantics are unknown, it is hard to decide *how* to compare different attributes. We address these challenges towards a service-based duplicate detection technique without human interaction, which is integrated into the DAQS (**Da**ta **Q**uality as a **S**ervice) project. In this paper, we propose



Field assignments

Number	not assigned
Gender	not assigned
GivenName	not assigned
MiddleInitial	not assigned
Surname	not assigned
StreetAddress	not assigned
City	not assigned
State	not assigned
ZipCode	not assigned
Country	
EmailAddress	
Password	
TelephoneNumber	
MothersMaiden	
Birthday	
CCType	
CCNumber	
CVV2	
CCExpires	
UPS	
Occupation	

To process your data, you must specify which address element each of your fields contains. Incorrect assignments are likely to result in poor validation results.
For each of the fields in your file, select the apropriate option from the drop down box next to it.

How should I assign my fields for optimal validation results?

Fig. 1. Screenshot of assigning semantics to attributes of uploaded input data (by kind permission of `addressdoctor.com`)

– A classification of different duplicate detection tasks based on the amount of available meta data
– A technique to classify attributes according to their semantics in the schema
– The novel 1:k assignment problem between attributes and similarity measures and an extended version of the Hungarian Algorithm to solve it
– A comprehensive evaluation on ten datasets showing the feasibility and effectiveness of our approach

The envisioned duplicate detection service works in three phases, as illustrated in Fig. 2. Each phase is described in detail in the following sections. In the *problem classification phase* (Sec. 2) the provided records are analyzed to determine their format and how to treat them during further processing, e. g., whether information retrieval techniques have to be applied, whether schema information are present, etc. In the subsequent *attribute classification phase* for each attribute a corresponding similarity measure is found automatically using the 1:k assignment technique. This phase is explained further in Sec. 3 and is the main focus of this paper. Section 4 presents an evaluation of the assignment results. The third *duplicate detection phase* executes the duplicate detection logic, i. e., the algorithm that decides against which pairs of records to apply the similarity measures. It is not the main focus of this paper and is thus described in Sec. 5 together with related work. Finally, Sec. 6 summarizes the approach and proposes future work.

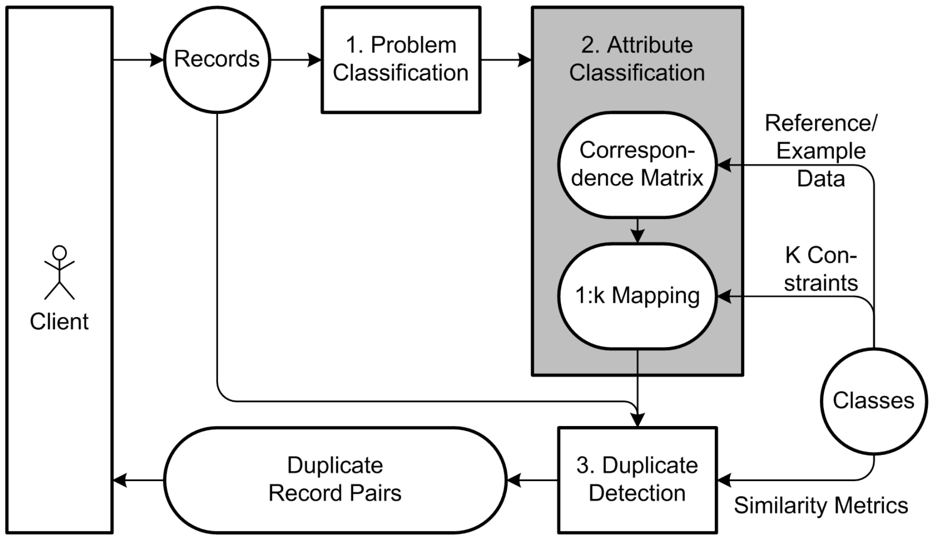


Fig. 2. Workflow and architecture of the duplicate detection service

2 Problem Classification

Usually, web services only operate on data provided during the service call, because they cannot access the customer's infrastructure, especially their data storage, ontologies, and schema definitions. Thus, web services are not invoked with the full diversity of data, but with a (perhaps simplified and reduced) copy of it, e. g., a CSV file instead of a complete SQL database. Depending on the actual case, the provided data might differ in the amount of details that are given. Input to our proposed duplicate detection service might lack a schema, lack data types, etc. Duplicate detection relies on knowledge about which attributes to compare and how to compare the attributes' values. Consequently, result quality degrades the less information is available.

Figure 3 shows a severity-classification of the problem, depending on the type of available metadata. Optimal conditions ((1) in Fig. 3) are present, if the semantics of the attributes are clear. In particular, we know not only the primitive datatype (String, Integer, etc.) but the concrete semantics (name, phone, date-of-birth, etc.). The comparison function as well as the mapping between the tuple's attributes can be derived from them. Consequently, the duplicate detection run can be performed nearly automatically.

In a more typical scenario, no semantics but only a mapping and possibly coarse-grained datatypes are given (2). Therefore, semantic classes have to be assigned by a human expert, which leads to the appropriate comparison functions. For instance, an expert might decide that first names should be compared using the Jaccard similarity measure.

In case the mapping is not explicitly given and the input data for the duplicate detection process is heterogeneous (3), a mapping between the values of different records has to be first established. Many methods are shown by Euzenat and Shvaiko [4] that use data types, attribute names, or values.

The most difficult case occurs when not even the attributes can be identified, e. g., as a result of web crawling or unknown data formats (4). The elements are structured, but the structure is unknown. To separate the elements' attributes from each other, information retrieval techniques have to be used.

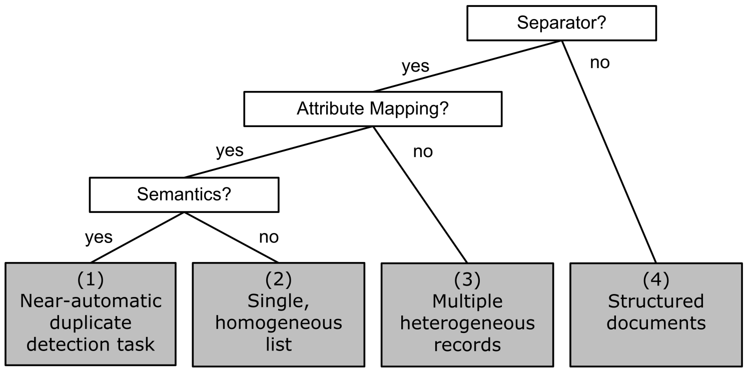

Separator?
yes
no
Attribute Mapping?
yes
no
Semantics?
yes
no
(1) Near-automatic duplicate detection task
(2) Single, homogeneous list
(3) Multiple heterogeneous records
(4) Structured documents

Fig. 3. Four different classes of duplicate detection are illustrated in a tree.

Please note that Class 1 is the most desirable one (and most easy to work with). With our approach of automatically determining suitable similarity measures we can transform Class 2 problems to Class 1 problems.

For the remainder of the paper we assume that we have no datatype information, no attribute names, but only instances. We further assume that the mapping is known, i. e., it is known which attribute from one tuple to compare with which attribute from another tuple. However, it is not clear *how* to compare these attributes. Finally, we assume that we can distinguish between several attributes and have separators (e. g., semicolons in CSV files).

3 Attribute Classification

The goal of attribute classification is to assign appropriate, highly specialized similarity measures to the attributes of the input data based on their semantics: Two instances of a *given name* should be compared differently than two *email addresses*, even though they might be of the same programming-oriented data type String. Yet a too detailed semantics detection is both not feasible and not of much help: For instance, given names and family names have the same characteristics, telephone and fax numbers are indistinguishable, so the same similarity measures can be used during duplicate detection.

Detecting attribute semantics has an additional advantage: Not only the similarity measure and appropriate parameters can be determined, but also normalization of values depends on the semantics. Telephone numbers can be sanitized from special characters, area codes might be introduced, for addresses *st* can be expanded to *street*, etc.

The process of creating such assignment between attributes and classes (i. e., semantics) is called *classification*. Similarity measures are directly derived from those classes. Note, that two different classes may imply the same similarity measure, e. g., *state* and *country*. An assignment consists of *correspondences* between attributes and classes.

Classification is performed in two phases. First, for each column, the dataset's values are compared against a set of reference data for different classes. The relative size of the overlap is used as indicator for the certainty of a correspondence (see Sec. 3.1). Second, another classification is performed with the help of feature vectors. These vectors are created for *example data* whose classes are known and the aforementioned input data whose classes have to be found. Both sets of data are then compared through machine learning techniques (see Sec. 3.2). Each of the phases returns a set of possible correspondences represented as a correspondence matrix. See Figure 4 for an overview on the whole classification workflow.

Section 3.3 describes how to distill the final assignment from those matrices with the help of a maximum bipartite graph matching and how to apply the 1:k constraint to the assignment: For this matching we take into account that some classes might be more frequent (e. g., given names) than others (e. g., birthday) and modify the matching algorithm into a 1:k assignment or "one-to-some"-assignment (Sec. 3.4).

Note that this approach is mostly language-independent. We do not know the language or country used in the dataset. However, we provide a large variety of classes with different abstraction, e. g., German street names vs. general street names as well as multi-lingual reference and training sets. We trust the algorithm to find the most appropriate class and thereby identify the language. Furthermore, for some classes, language is irrelevant, such as telephone numbers or email addresses.

3.1 Phase 1: Dictionary-based Classification

Some classes have a rather fixed domain or value range, e. g., *month* or *gender*. The overlap of the values in one of the dataset's attributes with those classes' ranges is expected to be large, even if there are mistakes in some of the values. We treat those sets of fixed values as "reliable dictionaries" and call them *reference datasets*.

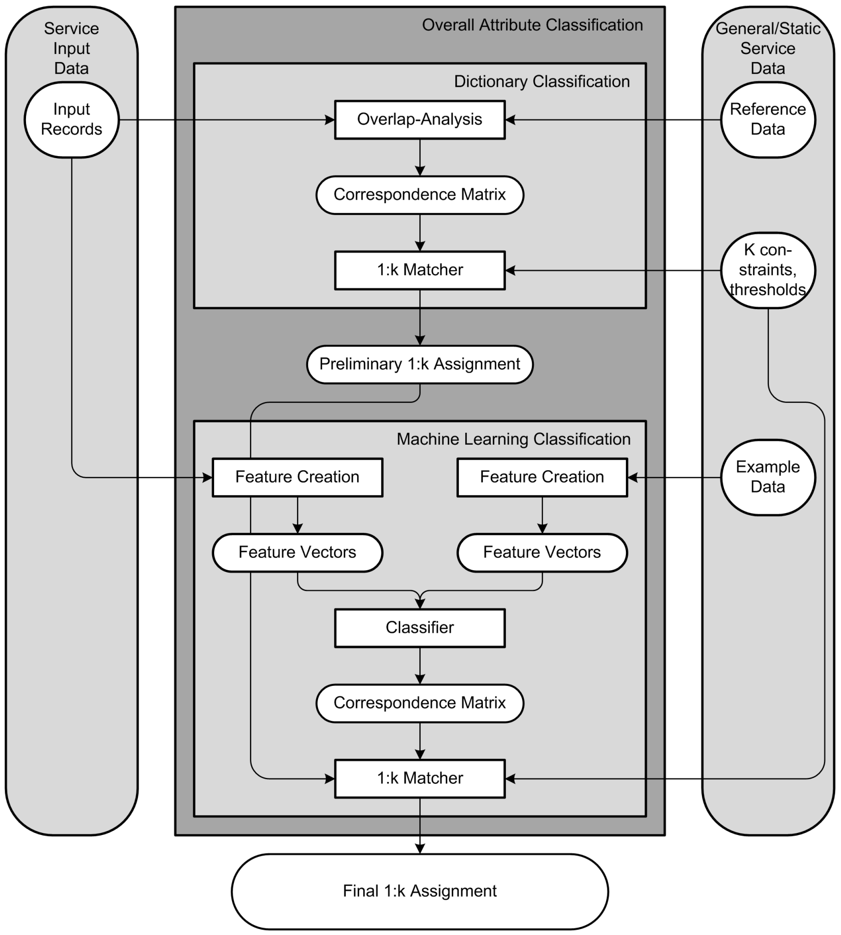


Fig. 4. Attribute classification phase at a glance (including the actual classification (Sec. 3.1, 3.2), correspondence matrices (Sec. 3.3) and the 1:k assignment (Sec. 3.4).)

To calculate the similarity between an attribute of the input data and a class, we determine the ratio of the attribute's values T that also appear in the class' reference dataset R divided by the total number of (non-NULL) attribute values T:

$$\frac{|T \rtimes R|}{|T|}$$

Ratios below a certain class-specific threshold (0.8 was a good threshold for most of the classes) are set to zero. We derive a preliminary assignment from this matrix with a maximum bipartite graph matcher, described in detail in Sec. 3.4. The All matches in this assignment are considered to be correct and are not questioned in the second phase. See the upper part of Fig. 4 for an overview on this phase.

3.2 Phase 2: Machine Learning Classification

Not all classes can be identified using a (finite) reference dataset, e. g., *telephone number* or *family name*. Therefore, we use example values available for those classes to learn the particulars of those classes for classification. The lower part of Figure 4 describes this classification process.

We define a set of boolean features, which are applied to each single attribute value, thus creating *feature vectors*. We use the common heuristic that there is a high probability that feature vectors for values from the same attribute/domain are similar. We define four different types of features:

1. 73 *single character features* check for occurrence of letters, digits, or special characters, e. g., “a”, “A”, “4”, or “#”.
2. 19 *multiple character features* check for more advanced patterns, e. g., whether a string begins with a lowercase letter (`[\p{L}&&[^\p{Lu}]].*` as a regular expression[2]), contains a separated 4-digit number (`(^|.*\D)\d{4}(\D.*|$)`), or has a length between 20 and 29.
3. $26^2 = 676$ *2-gram* features with letters (ignoring case)
4. 8 *lookup features* that use five services, namely DBpedia[3], behindthename.com[4], NameWiki[5], verwandt.de[6], and gofeminin.de[7], to look for given and family names. Not all services can distinguish between given and family names.

Once input and example data are represented by feature vectors, the input data can be classified based on the example data. We use the Naïve Bayes classifier of Weka [7] and classify each attribute separately.

3.3 Correspondence Matrix

Both the dictionary-based and the machine learning classification result in a *correspondence matrix*, each. A correspondence matrix contains n attributes and m classes, with $n \geq m$ without loss of generality. The matrix contains all possible correspondences between attributes and classes and describes the probability of each correspondence to appear in the final assignment.

Based on such a matrix, it is not trivial to determine which attribute to assign to which class. Conceptually, the problem corresponds to the weighted bipartite matching problem: Given the correspondence matrix with assignment weights, assign to each attribute a class such that no class participates in more than one assignment and the sum of weights is maximized. Yet, we introduce several twists to this problem:

[2] see “Classes for Unicode blocks and categories” section in the Java `Pattern` class API documentation

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

[4] `http://www.behindthename.com/`

[5] `http://wiki.name.com/`

[6] `http://verwandt.de/karten`

[7] `http://gofeminin.de`

– Thresholded assignment: To avoid matching "left-over" attributes with low similarity we introduce a threshold.
– One-to-many assignment: Several attributes might correspond to the same class, which should be reflected in our solution.
– One-to-some assignment: Domain knowledge allows us to restrict the number of matches to certain classes. For instance, we might want to encode that a person has no more than two given names. Thus, we redefine the matching problem in Sec. 3.4.

A simple 1:1 assignment would assign each attribute to a single class as long as there are unused classes left. A downside of this approach is that each attribute is forcefully matched, even if there is no correct matching partner. To solve this, we introduce a threshold: Correspondences below this threshold are not taken in the final assignment.

In addition, the 1:1 matching is too restrictive. Typical database schemas contain attributes with same or very similar semantics: For instance, "telephone", "mobile phone", and "fax" can all be compared using the same similarity measure. Thus, we want to allow the assignment of different attributes to the same class. To this end, we allow a 1:n assignment.

In further addition, knowledge about classes comprises also information about how often such a specific class may appear in the source schema: While it is possible that a schema contains several attributes similar to telephone numbers (i. e., fax numbers) it is very improbable that the gender is represented in a tuple several times. The gender class should only be able to take part once in the final assignment. To incorporate such restrictions in the matching problem, we propose the *1:k assignment* which is basically a 1:n assignment, but with varying *n* for each class.

3.4 1:k Assignment

Assume an acyclic, directed, bipartite graph $G = (S, T, E)$ with a set S of source nodes s_i, $i \in [1, n]$, a set T of target nodes t_j, $j \in [1, m]$, and a set E of edges e_{ij}, $i \in [1, n]$, $j \in [1, m]$ with $|E| = n \cdot m$. Such graph is depicted in Fig. 5. In our application, source nodes are the attributes, target nodes are the classes, and edges are the correspondences between them. Further, assume a correspondence matrix C with entries c_{ij} quantifying the similarity between source node s_i and target node t_j (c. f. Tab. 1).

Table 1. Correspondence matrix for attributes (first column) and classes (first row) with illustrative yet sound values

	Target				
Source	Firstname	Lastname	Phone	Address	City
Fullname	0.8	0.6	0.1	0.2	0.3
Telephone	0.0	0.0	0.9	0.2	0.1
Street	0.2	0.4	0.1	0.9	0.7
House Number	0.0	0.0	0.7	0.7	0.2

Assume also a set K of k-constraints $k_j, k = 1, \ldots, m$ where each k_j represents the number of assignments that are allowed for target node t_j.

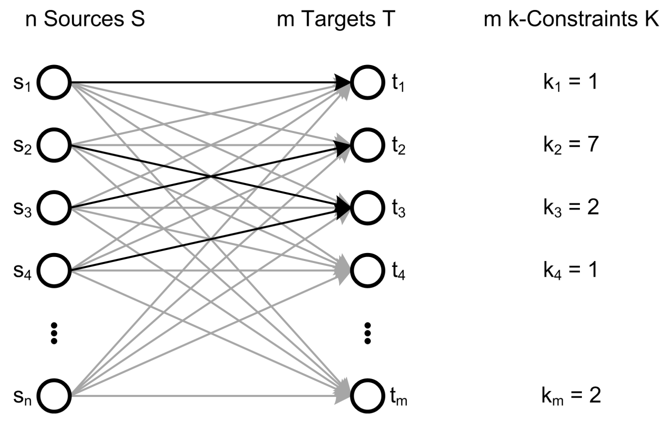


Fig. 5. Sample 1:k assignment with given K (only black edges)

Given the input stated above, the 1:k assignment problem is to find a mapping M with nodes

$$m_{ij} = \begin{cases} 0 & \text{if } e_{ij} \text{ takes not part in the mapping} \\ 1 & \text{if } e_{ij} \text{ takes part in the mapping} \end{cases}$$

where

$$\sum_{i=1}^{n} m_{ij} \leq k_j \quad (\forall j = 1, \ldots, m)$$

that maximizes the overall similarity of the selected participating nodes in the mapping:

$$\max \left(\sum_{\forall ij} c_{ij} \cdot m_{ij} \right)$$

A possible assignment is illustrated in Fig. 5, where n attributes are matched to m classes. Each attribute s_i is matched at most once, and each class t_j is assigned at most k_j times.

The final assignment M is calculated using a global matching algorithm on the correspondence matrix. This mapping task can be solved with an extended version of the Hungarian Algorithm [10]. The Hungarian Algorithm solves the assignment problem [13], which does not allow the multi-mappings of multiple attributes to the same class. It also requires a square correspondence matrix, so $n = m$. In general, this is not the case ($m \geq n$), so C has to be padded to construct C'. To this end ($m - n$) additional rows have to be added representing non-existing source nodes: $c'_{i'j} = 0 \quad \forall n < i' \leq m$.

The k-constraints are incorporated by duplicating columns of C'. The value of k_j determines the total number of additional copies of the column. Note that this column duplication requires $\sum_{k \in K} k_j = a$ additional rows, since the matrix becomes broader, but still has to comply to the squareness condition. In total, $(m - n) + a$ rows have to be added to generate a squared correspondence matrix with duplicated columns. See

Tab. 2 for an example C' with $K = \{k_1 = 3, k_3 = 2, ...\}$. It does not matter, where the additional columns or rows are inserted. k_i is also allowed to be infinite for classes that may appear arbitrarily often (e. g., boolean flags may be assigned unlimitedly). Since k_i cannot actually be set to infinity, is is sufficient to set k_i to the number of source attributes n. This gives every attribute the chance to become matched to this respective class.

Table 2. Extended correspondence matrix, now squared

Attributes (Source)/ Classes (Target)	Firstname	Firstname	Firstname	Lastname	Phone	Phone	Address	City
Fullname	0.8	0.8	0.8	0.6	0.1	0.1	0.2	0.3
Telephone	0.0	0.0	0.0	0.0	0.9	0.9	0.2	0.1
Street	0.2	0.2	0.2	0.4	0.1	0.1	0.9	0.7
House Number	0.0	0.0	0.0	0.0	0.7	0.7	0.7	0.2
dummy	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
dummy	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
dummy	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
dummy	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

With this squared correspondence matrix C', the Hungarian Algorithm can be used to create a 1:k assignment: M'. However, the result has to be modified. Unfortunately, the Hungarian Algorithm involves all source attributes into the mapping, even those $s_{i'}$ for which $i' > n$. Since the final mapping M shall have the same dimensions $n{\times}m$, M' has to be transformed into M. This is achieved by removing all matches with the dummy $s_{i'}$.

$$(m_{ij}) = (m'_{i'j}) \quad i = i' = 1, \ldots, n; \quad j = 1, \ldots, m$$

After the mapping M is finally created, each attribute got a class assigned. We can now directly derive concrete similarity measures for the attributes of the input data. Subsequently, pairs of input records can be examined for duplicates using these similarity measures.

4 Evaluation

This paper describes the assignment of semantics to attributes in the context of duplicate detection. Therefore, we evaluate the classification results, rather than the effectiveness of the duplicate detection.

We established ten test datasets with address data from various sources, available for download from `http://www.hpi.uni-potsdam.de/naumann/data`. This page contains further information concerning the sources and properties of the data. They are listed in Tab. 3 in the first column. The second column shows the number of attributes in each dataset.

Fakenames and *Corporate* are generated datasets derived by obfuscating existing addresses. *Crawl1* to *Crawl4* are crawled datasets from Wikipedia and IMDB. *ListB* and *ListC* are datasets from an assignment of the University of Arkansas, which are artificially obfuscated. *Voters* comprises a list of registered voters of Clermont county, Ohio. Finally, *Mines* contains the addresses of coal and ore mines in the USA.

For classification we need instance data. The datasets contain 14 attributes, on average, each dataset providing at least 50,000 tuples. Unfortunately, most of the used datasets contain attributes that are only sparsely filled. For training, we randomly selected 500 among the best-filled tuples in each dataset leaving up to three attributes with null values, but on different attributes over multiple records. Attributes that were empty in all records were ignored a priori. The training dataset bases on the Fakename dataset. Additional attributes are taken from the Crawl1, Crawl3, and Voters dataset or – in case of numbers – were randomly generated. We ensured that the contents of the training dataset do not overlap with the test datasets.

The results of the complete classification process are shown in Tab. 3. As proposed by Euzenat and Shvaiko [4] we use F-Measure to describe the overall matching compliance to the manually defined gold standard. Columns 3 and 4 display the number of correctly classified attributes and the corresponding F-Measure, respectively.

Table 3. Combined classification results (dictionary and machine learning with Naïve Bayes)

Dataset	Number of attributes	Correct Matches	F-Measure	Close Matches	Close Match F-Measure
Fakenames	21	15	0.71	+3	0.86
Corporate	11	9	0.81	+1	0.91
Crawl1 (KT)	10	8	0.80	+0	0.80
Crawl2 (LN)	13	6	0.46	+2	0.62
Crawl3 (Po)	8	8	1.00	+0	1.00
Crawl4 (RW)	16	8	0.50	+3	0.69
ListB	9	5	0.56	+1	0.67
ListC	7	4	0.57	+1	0.71
Voters	23	12	0.52	+3	0.65
Mines	18	10	0.56	+2	0.67

In most cases, the majority of attributes has been successfully classified. The misses occur on more unusual attributes, such as credit card verification codes, UPS tracking numbers, religion, or occupation. They are not believed to be of utmost importance for the duplicate detection process. The example in Tab. 4 shows that mismatches can also be non-severe as, e. g., phone is misclassified as number and the ID is misclassified as ZIP code.

The derived similarity measure might not accurately make use of all the special characteristics of phone numbers, but might still achieve sound results. Therefore, we additionally counted "close matches", such as weight/housenumber, place-of-birth/city-state-country-combination, or number of children/month (numeric). They are added to the strict matches and presented in Col. 5 in Tab. 3 with a corresponding F-Measure column, increasing the average F-Measure from 0.61 by 20 % to 0.73.

Table 4. Result with 11 attributes for the corporate dataset (only strict matches)

Dataset's Attribute	Classified as class	Similarity	correct?
housenr	HOUSENUMBER	1.00	correct
honorific	HONORIFIC_DE	1.00	correct
title	TITLE	1.00	correct
id	ZIP	0.98	incorrect (should have been NUMBER)
ZIP	ZIP_DE	0.96	correct
Street	STREET_DE	0.88	correct
City	CITY_DE	0.83	correct
Familiy Name	FAMILYNAME	0.72	correct
Date	BIRTHDAY	0.64	correct
First Name	GIVENNAME	0.62	correct
Phone Number	NUMBER_INTEGER	0.53	incorrect (should have been PHONE_DE/PHONE))

Each classification ran on a 3 GHz, 8 GB RAM Ubuntu Maverick 64 Bit non-dedicated machine in Java (default heap space size, single thread) within about 6/50/400 seconds with Naïve Bayes/Bagging/Ensemble of Nested Dichotomies (END) classifiers. The creation of features took additional time, which depends on the number of datasets, records, and attributes. For example data, this only has to be done once. The duration for processing the attribute value was yielding a total duration of 5 hours for all datasets. This only reflects computational effort and disk I/O, as we cached the HTTP requests for lookup features from earlier runs. Without caching, the feature creation time rises to 1.25 seconds per value or 25 hours in total. Thus, caching is recommendable so that only one HTTP lookup is necessary for each attribute value.

We also compared 1:k matching results against (unbounded) 1:n matching. With the chosen set of k-constraints, 1:k assignments are never worse than 1:n assignments. There are cases in which 1:k yields better results, especially when the classifier itself is more simplistic. This is positive, because in general, it cannot be assumed that the classes are always known and that example data is available. Such lack results in reduced classification quality and 1:k raises the F-Measure of the final assignment.

Finally, Fig. 6 shows a comparison of different machine learning algorithms and the achieved F-Measure on all datasets. Naïve Bayes provides the best cost-benefit tradeoff and was used for all the experiments. The figure's legend shows the average F-Measure and the average classification time for each dataset and each classifier.

5 Related Work

Duplicate Detection In general, duplicate detection has $O(n^2)$ complexity in terms of number of record comparisons where n is the number of records. Thus, efficient duplicate detection is a process that consists of two merely independent parts. First, an

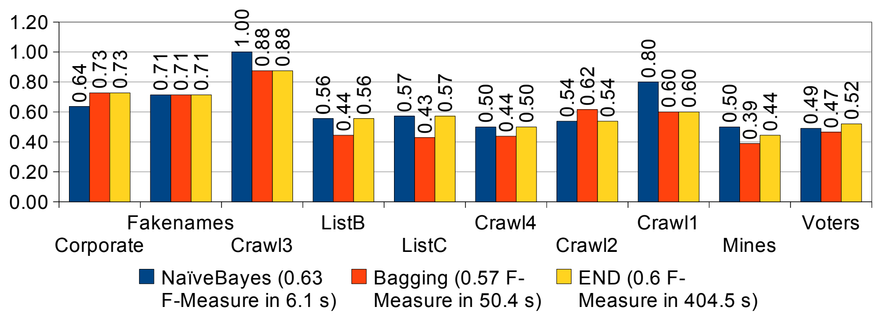


Fig. 6. The influence of different classifiers on the result is small, however the Naïve Bayes classifier is one order of magnitude faster than the others. The used classifiers are Naïve Bayes, Bagging, and Ensemble of Nested Dichotomies (END).

algorithm picks promising pairs of the set of records. Well-known heuristics are the sorted neighborhood method by Hernandez and Stolfo [8] and its extensions, such as Monge and Elkan's [9], as well as blocking or indexing approaches [6].

Second, a pair of duplication candidates has to be examined using a similarity measure. In case of relational data, it is common that (a subset of) the attributes are compared. With our approach, highly specialized measurements can be used.

Due to space restrictions and since there is much work regarding the actual implementation of duplicate detection, we do not elaborate this phase here. For example, DuDe [2] is a framework that can be easily integrated into the workflow shown in Fig. 2. An overview on both areas can be found in [11] or [3].

Schema Matching Schema matching is the technique of creating and selecting correspondences between two sets of elements, typically attributes of relations. Rahm and Bernstein give a survey on different methods for schema matching [14]; a more elaborate survey was written by Euzenat and Shvaiko [4]. A matching approach that inspired this paper, was the classification algorithm in [12]. It uses a rich feature set to create an instance-based mapping between two schemas. Instance mappings are also used in iFuice [16], where knowledge about explicit connections between different schemas is exploited. However, in the use case of customer data, those hyperlink connections are not available. Finally, Bilke and Naumann [1] combine both fields of research and utilize known duplicates for schema matching. This paper does the opposite and uses schema matching to eventually improve duplicate detection.

Services In terms of services, Faruquie et al. present a data cleansing service with an optional duplicate detection component [5]. However, proper thresholds for the pairwise attribute comparison part of the duplicate detection process have to be selected manually. Furthermore, they concentrate on arguing for data cleansing in general and omit details about how to actually perform the duplicate detection. They present different proposals for how to transfer the data to the service provider.

There are also some existing web applications that offer data cleansing. Mostly, this comprises only data verification and enrichment. Nevertheless, AddressDoctor[8], AdressExpert[9], and Uniserv[10] are commercial offers that perform duplicate detection, but only with considerable manual configuration effort.

6 Conclusion and Outlook

We presented a technique for the automatic assignment of semantic classes to attributes of a given dataset. The only prerequisite is the availability of training data for the desired domain in form of examples and/or reference data. The matching can be further improved by providing k-constraints for the 1:k matching. With this matching approach, the most relevant attributes can be identified and appropriate similarity measures can be derived.

For future work, we plan to extend the knowledge connected with the semantic classes. For example, the weighting of attributes should be different (family name vs. country) and some attributes might even be ignored, because they do not carry beneficial information for the similarity measure, e. g., IDs. This can be achieved by another learning phase with the use of known duplicate records. To further ease the usefulness for schema matching, the semantic classes will be augmented with a well-known vocabulary, e. g. with WordNET or YAGO. This helps in postprocessing steps and fosters interoperability.

Relations between different attributes should be taken into consideration. I. e., the presence of English street, city, and state names implies the existence of English ZIP codes. Another way of connecting them is the matching to composite attributes (e. g., given and family name together as full name). Finally, privacy aspects can be incorporated by not using the original values but transforming them into a metric space on client side [15].

Acknowledgements

We thank Dustin Lange for his valuable feedback and help on the machine learning aspects of this work.

References

1. Alexander Bilke and Felix Naumann. Schema matching using duplicates. In *Proceedings of the International Conference on Data Engineering (ICDE)*, 2005.
2. Uwe Draisbach and Felix Naumann. DuDe: The duplicate detection toolkit. In *Proceedings of the International Workshop on Quality in Databases (QDB)*, 2010.
3. Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge & Data Engineering (TKDE)*, 19, 2007.

[8] `http://www.addressdoctor.com/`
[9] `http://www.adressexpert.de/`
[10] `http://www.uniserv.com`

4. Jérôme Euzenat and Pavel Shvaiko. *Ontology matching*. Springer, Heidelberg (DE), 2007.
5. Tanveer Faruquie, Hima Prasad, Venkata Subramaniam, Mukesh Mohania, Girish Venkatachaliah, Shrinivas Kulkarni, and Pramit Basu. Data cleansing as a transient service. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, 2010.
6. Alfredo Ferro, Rosalba Giugno, Piera Puglisi, and Alfredo Pulvirenti. An efficient duplicate record detection using q-grams array inverted index. In *Data Warehousing and Knowledge Discovery (DaWaK)*, 2010.
7. Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: An update. *SIGKDD Explororation Newsletter*, 11(1), 2009.
8. Mauricio A. Hernández and Salvatore J. Stolfo. The merge/purge problem for large databases. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 127–138, 1995.
9. Alvaro Monge and Charles Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *Proceedings of the SIGMOD Workshop on Data Mining and Knowledge Discovery (DMKD)*, 1997.
10. James Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics (SIAM)*, 5(1), 1957.
11. Felix Naumann and Melanie Herschel. *An Introduction to Duplicate Detection*. Morgan & Claypool Publishers, March 2010.
12. Felix Naumann, Ching-Tien Ho, Xuqing Tian, Laura M. Haas, and Nimrod Megiddo. Attribute classification using feature analysis. In *Proceedings of the International Conference on Data Engineering (ICDE)*, 2002.
13. Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Prentice-Hall, 1982.
14. Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4), 2001.
15. Monica Scannapieco, Ilya Figotin, Elisa Bertino, and Ahmed K. Elmagarmid. Privacy preserving schema and data matching. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, 2007.
16. Andreas Thor. *Automatische Mapping-Verarbeitung von Web-Daten*. Dissertation, Institut für Informatik, Universität Leipzig, 2007.