Preprint Cambridge Centre for Computational Chemical Engineering ISSN 1473 – 4273

A Simple and Efficient Approach to Unsupervised Instance Matching and its Application to Linked Data of Power Plants

Andreas Eibeck [1], Shaocong Zhang [1], Mei Qi Lim [1], Markus Kraft [1,2,3,4]

released: January 26, 2023

[1] CARES
Cambridge Centre for Advanced
Research and Education in Singapore
1 Create Way
CREATE Tower, #05-05
Singapore, 138602

[2] Department of Chemical Engineering
and Biotechnology
University of Cambridge
Philippa Fawcett Drive
Cambridge, CB3 0AS
United Kingdom

[3] School of Chemical
and Biomedical Engineering
Nanyang Technological University
62 Nanyang Drive
Singapore, 637459

[4] The Alan Turing Institute
London
United Kingdom

Preprint No. 293

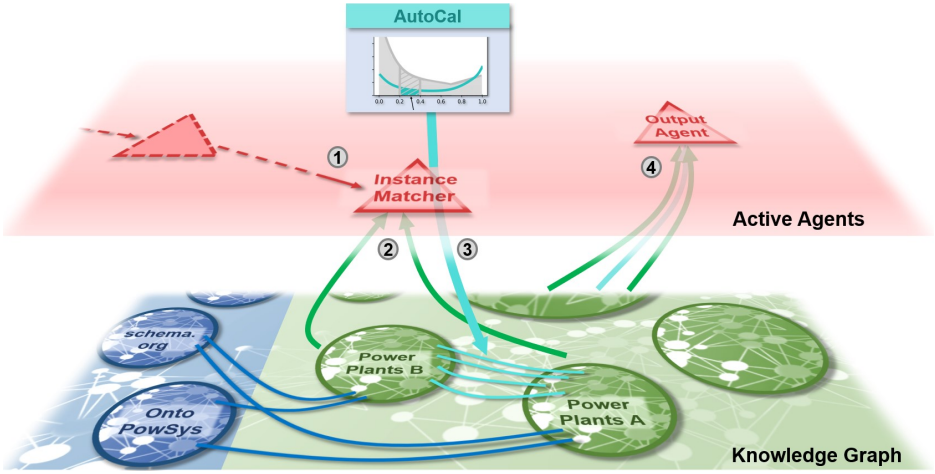
Keywords: Semantic Web, Linked Data, knowledge graph, instance matching, database integration, power plants

Edited by
Computational Modelling Group
Department of Chemical Engineering and Biotechnology
University of Cambridge
Philippa Fawcett Drive
Cambridge, CB3 0AS
United Kingdom

E-Mail: mk306@cam.ac.uk
World Wide Web: https://como.ceb.cam.ac.uk/

Abstract

Knowledge graphs store and link semantically annotated data about real-world entities from a variety of domains and on a large scale. The World Avatar is based on a dynamic decentralised knowledge graph and on semantic technologies to realise complex cross-domain scenarios. Accurate computational results for such scenarios require the availability of complete, high-quality data. This work focuses on instance matching – one of the subtasks of automatically populating the knowledge graph with data from a wide spectrum of external sources. Instance matching compares two data sets and seeks to identify instances (data, records) referring to the same real-world entity. We introduce AutoCal, a new instance matcher which does not require labelled data and runs out of the box for a wide range of domains without tuning method-specific parameters. AutoCal achieves results competitive to recently proposed unsupervised matchers from the field of Machine Learning. We also select an unsupervised state-of-the-art matcher from the field of Deep Learning for a thorough comparison. Our results show that neither AutoCal nor the state-of-the-art matcher is superior regarding matching quality while AutoCal has only moderate hardware requirements and runs 2.7 to 60 times faster. In summary, AutoCal is specifically well-suited to be used in an automated environment. We present its prototypical integration into the World Avatar and apply AutoCal to the domain of power plants which is relevant for practical environmental scenarios of the World Avatar.



Highlights

- A simple and efficient instance matcher designed for automated environments.
- Performance competitive to unsupervised Machine Learning methods for matching.
- Integration of power plant instances into the knowledge-graph based World Avatar.

Contents

1 Introduction

The Semantic Web [3] and Linked Data [4] enable data integration and semantic interoperability over the World Wide Web. Ontologies [17] define semantic terms and relationships of data in a formal way and provide the vocabularies necessary to describe entities and their properties. Recently, knowledge graphs [18] became popular as large-scale data structures storing and linking entity data.

The World Avatar builds on these concepts and related semantic technologies. While it originally focused on simulation and optimisation of eco-industrial parks to reduce waste and pollution [36], today it encompasses domains such as combustion chemistry [14], power systems [11], and urban planning [6]. Its architecture consists of a dynamic, decentralised knowledge graph. Computational agents operate on the knowledge graph to query, process and update its data. The same agents can be composed to solve complex cross-domain tasks with varying scales and initial conditions.

The following three air pollution scenarios exemplify the idea of agent composition: In the scenario presented by [13], a *power plant* agent computes emissions of a power plant located in Berlin, and a *dispersion* agent estimates the dispersion profile in the plant's neighbourhood with respect to surrounding buildings and real-time data on weather conditions. In a similar scenario [14], *ship* agents use real-time data of ship positions and types in the vicinity of Singapore to simulate their emissions while the same *dispersion* agent estimates the impact on a district of Singapore. In a future scenario, greenhouse gas emissions from various emitters could be computed and aggregated nationwide by creating more complex ensembles of agents. This scenario refers to a knowledge-graph approach for national digital twins proposed by [1].

Such scenarios assume the availability of high-quality data on entities such as power plants, ships and buildings to achieve accurate results. Additional agents which populate and update the knowledge graph with data from a multitude of providers are desirable but their automation is challenging because it involves several steps such as pre-processing, schema matching, instance matching and data fusion. If these steps are not performed correctly, the quality of the knowledge graph will degrade. In this paper, we focus on the subtask of instance matching on tabular data sets as the first step towards automation.

Instance matching considers two data sets and aims to find all instances denoting the same real-world entity. It can be regarded as a binary classification of pairs of instances which are either a match or nonmatch. Providers frequently publish data in tabular form, e.g. tables of power plants with properties (columns) for a plant's name, owner, locality, fuel type, design capacity, commissioning year, etc. In the case of tabular data, instance matching is also called record linkage. For many years, both fields are under active research, and a broad range of approaches was published, in particular from Machine Learning (ML) and Deep Learning (DL). For an overview of recent approaches and frameworks we refer to [15], [29], [20], [8], [23], [2].

DL methods for instance matching often concatenate property labels and values for each instance to generate high-dimensional embeddings from pre-trained neural models for training. By contrast, many ML methods rely on features which measure the similarity between property values of two instances. The latter requires the alignment of related

or equivalent properties of both data sets. This is commonly provided manually or by a preceding schema-matching step.

Supervised methods of instance matching require manual effort to label instance pairs. The models trained can be validated automatically. This allows for the optimisation of hyperparameters which are critical for matching performance. Unsupervised methods do not need labelled pairs but frequently rely critically on the adjustment of specific parameters for the data sets at hand [20, 29]. While specifying domain-related parameters, such as property alignments and similarity measures, is commonly not an issue, choosing the proper values for technical, rather method-specific parameters can be difficult. Challenges are thus to be expected when such methods are performed in an automated environment. This was our starting point for designing a new instance matcher.

The **purpose of this paper** is to present our results and experiences with matching real-world data such as data of power plants, with respect to matching quality, runtime behaviour and degree of automation. We introduce AutoCal, a new algorithm which can be applied stand-alone to match tabular data or be wrapped as an agent populating the knowledge graph of the World Avatar.

AutoCal does not require any labelled instance pairs. Like many ML approaches, AutoCal expects the similarity features of properties as input. Since data sets frequently consist of a mixture of expressive properties such as plant name or locality and more or less discriminative properties, AutoCal "calibrates" automatically the similarity feature values such that they become directly comparable and summable, and uses the resulting total scores for predicting matches.

We evaluate AutoCal on two test scenarios for the domain of power plants and on four test scenarios for other domains. The latter is used widely in the literature but gives more weight to textual than numerical property values. AutoCal achieves matching results comparable to results published for semi-supervised and unsupervised ML methods. At the same time, AutoCal exhibits only two technical parameters and shows stable behaviour for the parameters' default values. Hence, it can be used out of the box for new matching scenarios and is particularly suitable for integration into an automated environment such as the World Avatar.

Unsupervised DL methods usually perform better for the latter four test scenarios. For a deeper analysis, we choose the unsupervised state-of-the-art DL matcher CollaborEM [16] and compare it to AutoCal regarding the applicability, matching quality, and runtime behaviour. It turned out that AutoCal performs significantly better on one of the four widely used test scenarios and – after adding a missing "stop" criterion to CollaborEM – slightly better on both power plant scenarios. AutoCal runs at least 2.6 times faster than CollaborEM on a dedicated ML machine in the cloud and between 15 and 60 times faster on a local machine with a powerful CPU but less GPU memory. AutoCal's runtime behaviour and moderate GPU requirements make it a good choice to run on the same decentralised infrastructure as the World Avatar.

The paper is structured as follows: Section 2 outlines AutoCal and its matching pipeline. Section 3 gives an overview of related work. Section 4 summarises token-blocking and similarity functions, and presents AutoCal in detail. Section 5 evaluates AutoCal for the six test scenarios, presents published results of existing instance matchers, and compares

the performance of AutoCal and CollaborEM. Section 6 summarises the World Avatar's architecture and AutoCal's prototypical integration and demonstrates how matching is applied in the case of the power plant domain. Conclusions are outlined in section 7.

2 Outline of AutoCal

Instance matching can be regarded as a binary classification task where two data sets A and B are given and each pair of instances (a,b), $a \in A$, $b \in B$, is either a matching pair (match) or a nonmatching pair (nonmatch). A pair is a match if a and b refer to the same entity in the real world. The number of all pairs is $|A| \cdot |B|$ while the number of matches only grows linearly with the size of A and B. For instance, 905 out of around 2 million pairs are matches in one of the power plant scenarios considered in this work. This indicates that instance matching suffers from extreme imbalance and scalability issues.

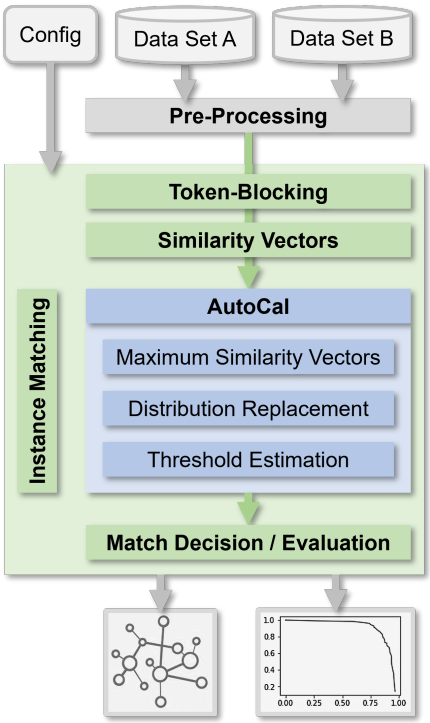


Figure 1: *Pipeline for instance matching: The green boxes denote preparatory steps commonly taken by methods of instance matching. The blue area refers to steps of our proposed instance matcher AutoCal.*

Figure 1 presents our complete pipeline for instance matching which consists of the preparatory steps such as blocking and the computation of similarity vectors and the characteristic steps of the AutoCal instance matcher. After pre-processing the data sets, blocking discards pairs that are very likely to be nonmatches and keeps a much smaller set of candidate pairs for classification. In practical applications, the overall performance of instance matching is crucial, and the chosen blocking technique and set of similarity features strongly impact it.

Before detailing token-blocking, similarity functions and AutoCal in section 4, we want to summarise the core ideas behind AutoCal. The entire pseudocode for token-blocking and AutoCal is presented in appendix A.2. The corresponding Python code was published on GitHub[1].

First of all, AutoCal splits values of string-valued properties into tokens and considers certain subsets of pairs of instances sharing tokens. Roughly speaking, for each subset, AutoCal selects the pair with the highest similarity values, i.e. the "most similar" pair, if it exists. Heuristically, the "most similar" pair has a higher chance of being a match compared to all competing pairs in the same subset and can be regarded as a potential matching candidate. AutoCal also considers the case if such a pair does not exist, but for the sake of simplicity, we ignore this case here. The complete procedure refers to step *Maximum Similarity Vectors* in Figure 1.

AutoCal's next step *Distribution Replacement* aims towards the conditional probability that the pair (a,b) is a match given $s_k(a,b) \in I$. Here, $I \subset [0,1]$ is a small interval and $s_k(a,b)$ denotes the similarity value between instances a and b regarding the k-th property. The conditional probability can be approximated by the ratio between all true matches having their similarity value in I and all pairs (both matches and nonmatches) having their similarity value in I as well. This ratio measures much better than the similarity value itself how relevant a similarity value is for matching. We illustrate this observation for instances representing power plants: If two instances have the same value for property *fuel*, the corresponding similarity value is 1, but it is unlikely that both instances match because there are many more nonmatching than matching pairs with the same fuel type. In this case, the ratio would be quite small. On the other hand, if two instances have only a similarity value of 0.6 for property *name* but corresponding values for nonmatches usually fall below 0.5, the ratio may be pushed towards 1.

Of course, the counts for true matches are not known. For this reason, AutoCal computes approximated ratios by replacing the unknown counts with the known counts for the "most similar" matching candidates from step 1. In summary, AutoCal converts the original similarity values property-wise into matching-relevant ratios. We call these ratios *calibrated scores* because – in contrast to the original scores – the ratios for different properties are directly comparable to each other in the context of instance matching. The last step, *Threshold Estimation* in Figure 1, computes a total score for each instance pair just by averaging its calibrated scores and predicts pairs with a total score above a certain threshold as matches. Since the calibration works in an unsupervised way without any labelled pairs, we call our algorithm "auto calibration" or "AutoCal" for short.

We regard AutoCal as a simple algorithm which is based on statistics and heuristics: AutoCal employs only basic statistics throughout all steps, such as histograms with equally-sized bins to represent distributions of similarity values on the range $[0,1]$. In addition, the replacement of the unknown distribution of true matches by the empirical distribution of "most similar" pairs is motivated heuristically. However, AutoCal does also make use of a pre-trained language model to derive string similarity features.

[1] https://github.com/cambridge-cares/TheWorldAvatar/tree/main/Agents/OntoMatchAgent

3 Related Work

In this section, we outline some existing methods for instance matching which are unsupervised or require only a small amount of labelled pairs, and which are thus candidates for an automated environment. The selected methods cover a variety of approaches and achieve results close to or among state-of-the-art in ML and DL. Many of them use similarity functions which score the similarity between two property values, where the score ranges from 0 (no similarity) to 1 (equality). We refer to [20] for details about semi- and unsupervised methods in instance matching. For the sake of simplicity, we use the terms *instance*, *entity* and *record* synonymously, and likewise the terms *property*, *column* and *attribute*.

Auto-FuzzyJoin [26] considers instance matching as an optimisation problem that maximises recall for a user-defined minimum precision value. The algorithm is based on a geometric argument and searches iteratively for combinations of distance functions and upper distance bounds to define proper matching rules.

Wombat [34] is a state-of-the-art approach for link discovery. Link discovery can predict relations of any type between instances in knowledge graphs and is not restricted to instance matching or tabular data. Wombat relies exclusively on positive examples, i.e., on an initial subset of labelled matches in case of instance matching. It learns rules which specify properties, similarity functions and similarity thresholds for linking (matching) two instances. Wombat starts with simple rules and applies set-valued operators to combine and refine them such that the resulting more complex rules provide a higher performance on the initial subset of labelled matches.

Christen [7] presents an iterative self-supervised approach: First of all, a given percentage of pairs with similarity vectors closest to the 1-vector (e.g. closest with respect to Manhattan or Euclidean metric) is labelled as matches while those closest to the 0-vector are labelled as nonmatches. These labelled seed pairs are used to initially train a Support Vector Machine (SVM) classifier. After training, the "strongest" predictions for matches and nonmatches, i.e. the predictions that are furthest away from the SVM decision boundary, are added to the training set, and a new SVM classifier is trained. This procedure is repeated as long as the size of the training set is below a given threshold.

Kejriwal and Miranker [21] propose an iterative semi-supervised method requiring a small seed set of labelled training pairs initially. Their method trains an ensemble of base classifiers (either Random Forests or Multilayer Percepton) using boosting. Similarly to [7], the most confident predictions for matches and nonmatches are added to the training set for training a new boosted classifier. The training loop terminates after a configured number of iterations.

Jurek et al. [19] also train an ensemble of base classifiers (SVM) iteratively but their approach is unsupervised and includes several initial steps to select a set of seed pairs for each base classifier such that the sets exhibit a high diversity. These steps include the selection of similarity functions and the computation of property weights that express the distinguishing power of the property values.

ZeroER [35] models the unknown match and nonmatch distributions as a Gaussian mixture. It is based on the expectation-maximisation algorithm to estimate the unknown

parameters for the Gaussian distributions and the mixture ratio between matches and non-matches. The method exploits the fact that matches tend to have higher similarity values than nonmatches and provides an adaptive regularisation strategy to prevent overfitting.

EmbDI [5] converts records from both data sets into a common graph and learns neural embeddings for the graph nodes. The graph defines three types of nodes to represent record IDs (entities), column IDs (properties), and tokens. Nodes representing two records are connected indirectly via a common token node if the records share a token. Next, random walks are performed on the graph nodes, and the visited nodes of each random walk are concatenated to a sentence containing record IDs, column IDs and tokens. The total of all sentences generated allows it to learn embeddings in an unsupervised fashion. Finally, the cosine distance between embeddings representing a pair of records serves as a matching criterion.

ErGAN [33] is a semi-supervised generative adversarial network which is specifically designed for instance matching. Initially, it computes the median of all similarity values for each property and divides the set of unlabelled pairs into subsets according to whether their similarity values are smaller or greater than the median values. Then, for each training iteration, its diversity module gives unlabelled pairs a higher priority if they belong to a smaller subset. In combination with the propagation module, the diversity module tackles the problem of imbalance between matches and nonmatches and avoids overfitting.

CollaborEM [16] is a self-supervised framework running two phases: The first phase computes embedding vectors for each record by means of Sentence-BERT [32]. By using the cosine distance and additional constraints, record pairs are identified that are very likely to be true matches or true nonmatches. The second phase uses this auto-generated seed set of labelled pairs for supervised training: each record is converted into a multi-relational graph, and a graph neural network is trained to learn four types of graph features (embeddings) for labelled pairs. Afterwards, both the trained graph features and the sentence representations of record pairs are used in a collaborative way to fine-tune a pre-trained transformer model and to make a final matching decision.

Regarding its paradigm, AutoCal is most comparable to Auto-FuzzyJoin. By contrast, many previously outlined methods train standard ML classifiers or use neural embeddings. Of course, all ML methods take advantage of the fact that the similarity of property values tends to be greater for matches than for nonmatches. But AutoCal strictly exploits this fact in its first step when identifying "most similar" instance pairs. On closer inspection, some of them also reveal implicit assumptions or steps similar to those of AutoCal. For example, EmbDi's graph conversion relies on common tokens, and ErGAN's propagation module uses some statistics.

While the previously outlined methods are unsupervised and partly achieve state-of-the-art results, some of them have requirements which may reduce their applicability in an automated environment. For instance, Auto-FuzzyJoin assumes a reference data set not containing any duplicates, while Wombat requires a specific coverage threshold for properties. Jurek-Loughrey and Deepak [20] point out that it can be problematic to adapt method-specific parameters for new data sets without labelled pairs. CollaborEM lacks a criterion to stop the fine-tuning. In contrast to all of those methods, AutoCal can be used out of the box.

4 Instance Matching

4.1 Token-Blocking

Many data sets contain columns with highly expressive string values, like instance names, descriptions, or addresses. Because of their expressiveness, such values are valuable for instance matching. Each of the string values can be split into a set of tokens. While the respective strings related to matching entities may differ in detail, they usually contain some equal tokens. We thus choose token-blocking, which considers two instances as a candidate pair for matching if they share at least one token. While AutoCal may be combined with other blocking techniques, token-blocking suits AutoCal in a natural way since the subsets for which AutoCal's first step identifies the "most similar" instance pairs can be easily created and managed in parallel. Token-blocking was introduced by Papadakis et al. [31]; we refer the reader to [30] for an overview of further unsupervised blocking approaches.

Figure 2 demonstrates token-blocking by means of instances from two data sets A and B. Here, A and B are two data sets of power plants in Germany which we will also use for evaluation later on in this paper. Among the instances of the data sets, we find an instance $a3 \in A$ with name "moabit" and an instance $b1 \in B$ with name "berlin moabit power station". Both instances share the token "moabit" in their name, which denotes a district of the city of Berlin. We thus add them to the set of all instances from A and B, resp., which also contain the token. Since the name of instance $b1$ contains three more tokens "berlin", "power", and "station", the instance is also added to the corresponding sets of these three tokens.

We create a mapping $Index_A$ from the set of all tokens to their related instances in data set A by iterating over all instances in A, tokenising their names and further string values of other properties, and finally adding the instances to the corresponding sets. For $Index_B$, we proceed likewise. The block for a given token t is defined as the Cartesian product $Index_A(t) \times Index_B(t)$. The set of candidate pairs is the union of all blocks for all found tokens. Blocks for two different tokens overlap if some instances $a \in A$ and $b \in B$ share both tokens. However, the candidate pair (a,b) only appears once in the union set. A token found only in instances from one of the data sets would lead to an empty block and is ignored.

If the index considered only tokens derived from equivalent properties in A and B, then there may be instances of A that do not share any tokens with instances of B. For example, 23 % of the instances in one of the data sets of power plants in Germany miss values for the property *name*. On the other hand, most of these instances provide a non-empty string for the property *address locality*. For this reason, token-blocking takes tokens from multiple properties in order to create the index. It subsequently considers instances as potential matching partners that would be filtered out otherwise.

Frequent tokens are less distinctive and increase the number of candidate pairs disproportionately. Our implementation of token-blocking provides the "maximum token frequency" (MTF) as a configuration parameter and ignores those tokens for which the number of corresponding instances either in A or in B exceeds the configured MTF value. In

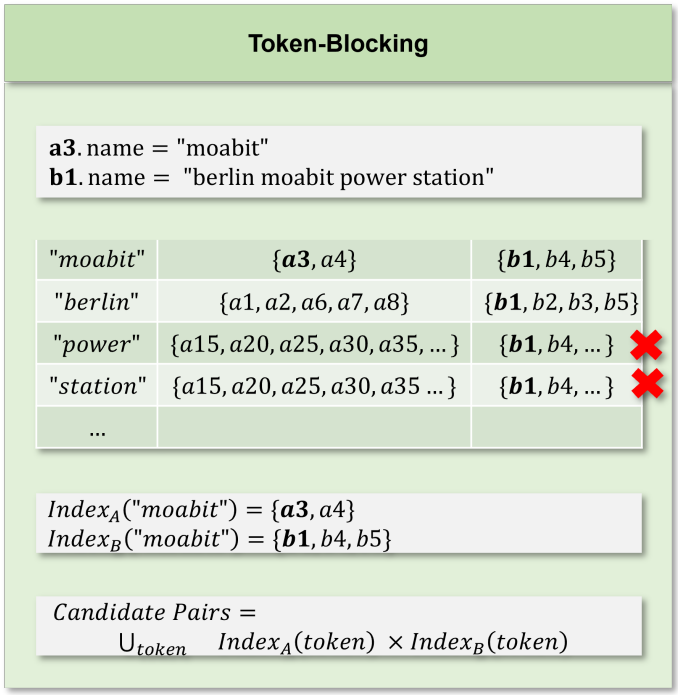


Figure 2: *Token-blocking identifies those entity pairs that share at least one token. This technique reduces the number of candidate pairs that have to be classified as match or nonmatch. It thus avoids issues of scalability and extreme imbalance.*

Figure 2, this is indicated by two red crosses for tokens "power" and "station" for the case MFT $= 5$. MFT strongly impacts the number of candidate pairs and plays an important role in the overall performance of instance matching.

4.2 Similarity Vectors

Two instances a and b can be regarded as similar if their property values are similar. The similarity of two property values $v_1, v_2 \in V$ is measured by a similarity function $f : V \times V \to [0,1]$. The similarity value $f(v_1, v_2)$ is 1 if v_1 and v_2 are equal, close to 1 if they are very similar, and 0 if they are completely different. A wide range of similarity functions was devised, each being applicable for particular domains, e.g. Cohen et al. [9] evaluates various similarity functions for matching names.

Table 1 lists the similarity functions considered for matching in this paper. Note that a distance function such as the absolute error can easily be converted into a similarity function by composing with a suitable increasing function $g : [0,\infty) \to [0,1]$ and subtracting from 1. The last two rows refer to cosine similarity which is defined for two (non-zero) vectors v and w by the normalised dot product $v \cdot w / \|v\| \|w\|$. Here, v and w are vector representations of two given strings. "Weighted" means that each vector component corresponds to a separate token and is set to its tf-idf weight, see e.g. [9], while the last row refers to sentence embeddings [32][2].

[2] see https://github.com/UKPLab/sentence-transformers

Table 1: *Similarity functions.*

Data type	Name	Description
any	equality	$1_{\{x=y\}}$
number	absolute error	$1 - g(\lvert x-y \rvert)$
number	relative error	$1 - \frac{\lvert x-y \rvert}{max\{\lvert x \rvert, \lvert y \rvert\}}$
string	cosine similarity - weighted	tf-idf weights, [9]
string	cosine similarity - sentence embeddings	[32]

AutoCal requires a mapping between related properties of data sets A and B as well as respective suitable similarity functions. This mapping may be derived manually or automatically by schema matching, for example. The selection of similarity function for a pair of mapped properties may also be automated, e.g., by preliminary analysis of type and range of property values in data set A and B. The similarity functions from Table 1 cover all data types relevant to this work and may be modified or extended, for instance for a new data type such as *date*. For the evaluation of AutoCal in section 5, we applied both cosine similarity functions to each mapped pair of string-valued properties since they complement each other: The cosine similarity between weighted vectors works well for short strings with a few expressive tokens such as names or locations while the cosine similarity between sentence embeddings expresses the semantic similarity for longer strings such as product descriptions.

We describe the mapping by K triples of the form $(p_k, q_k, f_k), k = 1, \ldots, K$. Here, p_k and q_k denote properties of instances $a \in A$ and $b \in B$ resp., and f_k a suitable similarity function that can be applied to the pair $a.p_k$ and $b.q_k$ of property values. The definition of a mapping allows the application of several distinct similarity functions for the same related properties. Each f_k may also map to the special case *null* to indicate missing property values. Finally, a given mapping of K triples defines a mapping from the set of candidate pairs into the space of similarity vectors by

$$s(a,b) = (f_1(a.p_1, b.q_1), \ldots, f_K(a.p_K, b.q_K)) \in ([0,1] \cup \{null\})^K . \tag{1}$$

Each similarity vector consists of K similarity values. In the following, we will use the term k-th component, k-th triple or k-th property interchangeably to refer to its individual similarity values.

4.3 AutoCal

AutoCal requires the set C of candidate pairs and corresponding similarity vectors $s(\cdot,\cdot)$ from the two previous subsections as input and consists of three steps as shown in Figure 1. For a given candidate pair (a,b), the three steps often perform calculations and comparisons on the subsets

$$C(a) = \{(a, \tilde{b}) \in C\} \quad \text{and} \quad C(b) = \{(\tilde{a}, b) \in C\} \tag{2}$$

and hence consider two directions: The first keeps a fixed and varies associated $\tilde{b} \in B$, while the second keeps b fixed and varies associated $\tilde{a} \in A$. Due to token-blocking, asso-

ciated instances $\tilde{b}$ are those that share at least one token with a. However, two instances $\tilde{b}_1$ and $\tilde{b}_2$ associated both with a might not have any tokens in common by themselves. The same holds for the other direction.

The **first step** of AutoCal computes the **maximum similarity vector** for each $a \in A$:

$$s^{max}(a) = (\max\{s_k(a,\tilde{b}) : (a,\tilde{b}) \in C\})_{k=1}^{K} \in ([0,1] \cup \{null\})^K \tag{3}$$

The maximum similarity vector $s^{max}(b)$ for each $b \in B$ is determined likewise.

The maximum similarity vector is larger than or equal to all associated similarity vectors in any of the K components. The left diagram in Figure 3 illustrates this concept for a power plant instance $a3$ with only $K = 2$ properties *fuel* and *name*. Of the three pairs shown, the maximum similarity vector is the one coloured turquoise. It may happen that none of the similarity vectors can be identified as best. This situation is illustrated in the right diagram of Figure 3. In this case, $s^{max}(a)$ is not associated to any pair $(a,\tilde{b}) \in C$ and we consider it as "virtual" vector. Moreover, if the value for the k-th property is missing for a or $\tilde{b}$, then $s_k(a,\tilde{b})$ is also missing, and is ignored when computing the maximum in eq. 3. If the k-th value is missing for all $(a,\tilde{b})$, $s_k^{max}(a)$ is set to *null*. This propagation of missing values through operations and comparisons is straightforward and will not be mentioned further in the following.

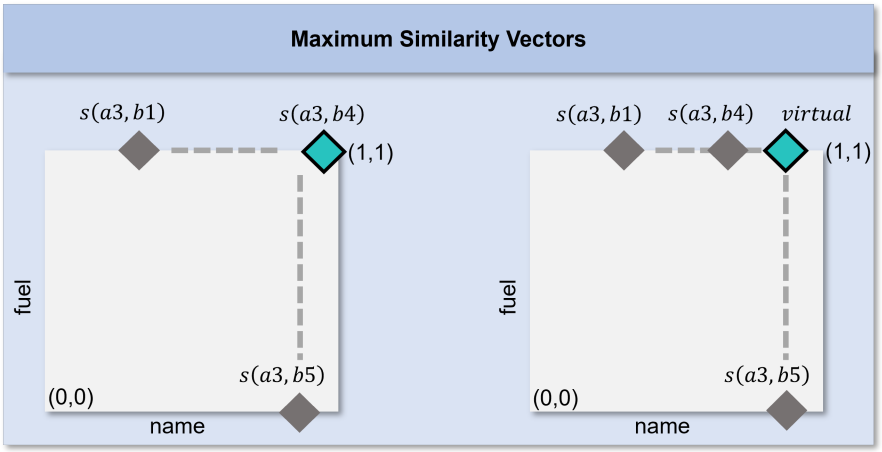


Figure 3: *Candidate pair $(a3,b4)$ is more likely to be a match compared to $(a3,b1)$ and $(a3,b5)$ since its similarity vector (turquoise diamond in the left diagram) is larger in all its components. If no such "best" pair exists, a "virtual" maximum similarity vector (turquoise diamond in the right diagram) is computed.*

The heuristic behind definition 3 is that if we choose any $a \in A$ and look at the subset $C(a)$ of candidate pairs, we would suppose that a pair with similarity vector in the vicinity of $s^{max}(a)$ has a higher chance to be a match than pairs with similarity vectors further away — as long as the similarity functions employed are reasonable and do not contradict the concept of similarity. Likewise for $b \in B$ and $C(b)$. Nevertheless, such a pair does not have to be a match.

The **second step** of AutoCal uses the maximum similarity vectors and a statistical approach to decide for each of the K components which similarity values are both typical

and relevant for matches. It is based a) on the **replacement of the unknown match distribution** by the known empirical distribution for the maximum similarity vectors, and b) on a rough estimation for the mixture of the marginal match and nonmatch distributions. Figure 4 illustrates this concept for the k-th triple referring to property *name*. The upper left diagram depicts the counts of matches and nonmatches for the k-th component. In the lower left diagram, the match counts are replaced by counts derived from the k-th component of all maximum similarity vectors. Usually, the counts from the maximum similarity vectors differ from the match counts, in particular in the vicinity of 0 and 1. They provide a very rough approximation of the actual match distribution; however, if the bias is not too strong, the counts help to rate the relevance of similarity values for instance matching in form of scores.

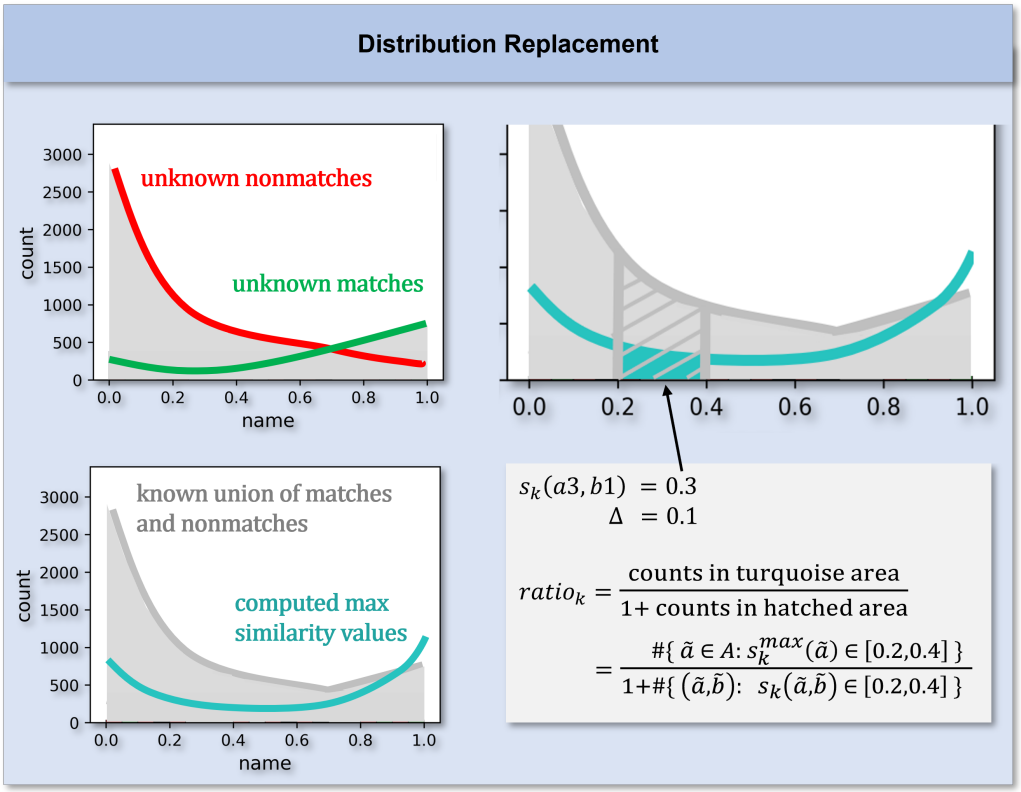


Figure 4: *The unknown marginal distributions of matches (upper left diagram) are replaced by the empirical marginal distributions of all calculated maximum similarity vectors (lower left diagram). For each property, an auto-calibrated score is calculated as the ratio between the replacement distribution and the overall distribution of matches and nonmatches (upper right diagram).*

The upper right diagram of figure 4 demonstrates how AutoCal uses the counts to calculate the score for the k-th triple in case of the specific candidate pair $(a3, b1)$ with similarity value $t = s_k(a3, b1) = 0.3$: First of all, it counts all $\tilde{a} \in A$ for which the k-th component $s_k^{max}(\tilde{a})$ of the corresponding maximum similarity vector belongs to the Δ-neighbourhood of t, i.e. it counts those in the turquoise area of width 2Δ. Next, it counts all candidate pairs $(\tilde{a}, \tilde{b})$ with similarity value $s_k(\tilde{a}, \tilde{b})$ in the same Δ-neighbourhood, i.e. it counts those in the grey hatched area. Finally, it calculates the ratio of both counts, as in the lower

left diagram for $t = 0.3$, and uses the maximum of the ratio and 1 as property score $score_k^A(a3, b1)$ to rate the relevance of the k-th component. Here, superscript A indicates that the nominator of the ratio was calculated with respect to $s_k^{max}(\tilde{a}), \tilde{a} \in A$. Likewise, the property score $score_k^B(a3, b1)$ is computed with respect to $s_k^{max}(\tilde{b}), \tilde{b} \in B$.

Appendix A.2 shows the general formulae for the counts and the ratio, and an efficient procedure for their approximate calculation. We also provide an interpretation of the property scores as rough estimates for the conditional probability that the pair (a,b) is a match given its k-th similarity value.

So far, AutoCal has transformed the k-th similarity value $s_k(a,b)$ for an arbitrary pair $(a,b) \in C$ into property scores $score_k^A(a,b)$ and $score_k^B(a,b)$ which rate the relevance for (a,b) being a match. The first step collected the best similarity values on subsets $C(a)$ and $C(b)$, resp., while the second step applied them to "calibrate" similarity values property-wise.

The **third step** aggregates the property scores from the previous steps into total scores and **estimates a threshold** to separate matches from nonmatches. To this end, it applies operations to the total scores similar to the property-wise operations in the previous two steps.

The property scores are first aggregated by taking their average, denoted by $score^A(a,b)$ and $score^B(a,b)$, resp., and the total score is computed as maximum

$$score(a,b) = \max\{score^A(a,b), score^B(a,b)\} \in [0,1] \,. \tag{4}$$

Similar to the property-wise construction of the maximum similarity vectors, we would suppose that – given $a \in A$ – the pair with the maximum total score among all pairs in $C(a)$ has the highest chance to be a match. Likewise for $b \in B$ and $C(b)$. AutoCal shortlists all these pairs with maximum total scores as potential matches C_M and considers the remaining candidate pairs as nonmatches. Next, it increases the threshold value t starting from 0 by steps of size 2Δ and calculates the counts

$$\begin{aligned} n_M(t) &= \#\{(a,b) \in C_M : t - \Delta \leq score(a,b) \leq t + \Delta\} \,, \\ n_N(t) &= \#\{(a,b) \in C \setminus C_M : t - \Delta \leq score(a,b) \leq t + \Delta\} \,. \end{aligned}$$

If the threshold value satisfies the condition $n_M(t) \geq n_N(t) + 1$ three times in sequence, it is chosen as estimated threshold t_{est}. This is a compromise between values far below t_{est} which may lead to a low precision, and values far above t_{est} which may lead to a low recall. Finally, AutoCal predicts all $(a,b) \in C_M$ satisfying $score(a,b) \geq t_{est}$ as matches.

5 Evaluation

5.1 Data Sets

Table 2 presents the scenarios for the evaluation of instance matching (IM) in this work. Each IM scenario consists of two data sets A and B and an evaluation set with labelled ground truth matches and nonmatches. The table contains the number of instances in

A and *B*, the number of properties used for matching, the size of the evaluation set, the number of matches within the evaluation set, and an ID that we will use for reference in this work.

Table 2: *Scenarios for evaluating instance matchers.*

ID	Data Set *A*	#*A*	Data Set *B*	#*B*	#Prop	#Eval	#Matches
KG	KWL	1,983	GPPD DE	982	5 (+1)	10,392	905
DG	DUKES	1,184	GPPD GB	2,536	4	13,509	950
FZ	Fodor	533	Zagat	331	5	946	110
DA	DBLP	2,616	ACM	2,294	4	12,363	2,220
DS	DBLP	2,616	Scholar	64,263	4	28,707	5,347
AG	Amazon	1,363	Google	3,226	3	11,460	1,167

The first two rows present the scenarios of the power plant domain. Scenario KG refers to matching data of power plants in Germany published by the *Federal Network Agency*[3] with data from the Global Power Plant Database (GPPD)[4], while scenario DG refers to matching data of power plants in the United Kingdom published by the *Department for Business, Energy and Industrial Strategy*[5] with data from GPPD. All data sets involved provide the properties *name*, *primary fuel*, *capacity* (in MW) and *owner*, which are considered for matching. In scenario KG, we also take into account the property *commissioning year*.

For both KG and DG, we created the ground truth manually step by step: Candidate pairs were generated iteratively by token-blocking for increasing MTF and by means of the *Python Record Linkage Toolkit* [10], and were classified as match or nonmatch manually in each iteration. In addition, we considered the property *address locality* for token-blocking in scenario KG. This property is only available for data set *A* and helps to compensate for missing power plant names in this data set. Finally, for both KG and DG, the candidate set created by token-blocking for MTF $= 50$, combined with the manually identified labels, serves as an evaluation set. Candidate pairs not identified previously are considered nonmatches.

Scenarios in all but the first two rows of Table 2 were widely used for evaluation in other papers and refer to the domains *restaurant*, *bibliography*, and *product*, respectively. The labelled data for DA, DS and AG originate from [24]. For a fair comparison, we downloaded the data from the Magellan Data Repository[6] which is frequently used for recent research in matching. Since AutoCal is unsupervised and CollaborEM is self-supervised, we use the union of the downloaded train, validation and test sets as an evaluation set.

For future use, the new data for KG and DG is publicly available[1] in the same format as

[3] https://www.bundesnetzagentur.de/DE/Sachgebiete/ElektrizitaetundGas/Unternehmen_Institutionen/Versorgungssicherheit/Erzeugungskapazitaeten/Kraftwerksliste/start.html, version 01/04/2020

[4] https://datasets.wri.org/dataset/globalpowerplantdatabase, version 1.2.0

[5] https://www.gov.uk/government/statistics/electricity-chapter-5-digest-of-united-kingdom-energy-statistics-dukes, DUKES 5.11 2019

[6] https://github.com/anhaidgroup/deepmatcher/blob/master/Datasets.md

provided by the Magellan Data Repository.

5.2 Quality Metrics and Setup

Given an evaluation set, the quality metrics precision P, recall R, and F_1-score are defined by

$$P = \frac{TM}{TM+FM}, \quad R = \frac{TM}{TM+FN}, \quad F_1 = \frac{2PR}{P+R}, \tag{5}$$

where TM, FM and FN denote the number of true predicted matches (true positives), false predicted matches (false positives) and false predicted nonmatches (false negatives) on the evaluation set. AutoCal estimates a threshold to make matching decisions for all pairs within the candidate set. Since the candidate set is no superset of the evaluation set in general, we set all pairs outside the candidate set as predicted nonmatches, i.e. true matches outside the candidate set are counted correctly as false predicted nonmatches.

AutoCal requires some parameter values to be provided: The bin width Δ is set to 0.025 for all six IM scenarios. First, we will vary the MTF value to analyse its effect on the matching quality. Then we fix it to the suitable default value of 50. We defined a fixed property (column) mapping for each scenario, comprising similarity functions from Table 1: For numerical properties, we either applied the absolute or the relative error, for properties with enumerated values (such as fuel type) we used the equality function, and for string-valued properties, we used the cosine similarity both for TF-IDF weights and sentence embeddings. In effect, we employed two similarity values per string-valued property and otherwise one.

Concerning CollaborEM, we applied the same setup including default parameter values as is described in [16]. For DS we use the same graph features as for DA, while for KG and DG we use all available four graph features.

Both AutoCal and CollaborEM ran on machines of a major cloud provider – equipped with an Intel Xeon CPU, 89 GB RAM and NVIDIA A100 GPU with 40 GB memory. In addition, we run both methods on a laptop with an Intel Core i7-11800H CPU, 32 GB RAM and NVIDIA GeForce RTX 3070 GPU with 8 GB memory. In the first case, we trained CollaborEM with batch size 64 as in [16]; in the second case, we had to reduce the batch size to 16 to avoid GPU memory problems.

5.3 Results Regarding Matching Quality

Figure 5 summarises the results of our ablation study for AutoCal and answers the following questions: How does the choice of MTF impact AutoCal's matching quality, and how much do AutoCal's steps contribute to it respectively? Each diagram refers to one of the IM scenarios and contains three groups that we will discuss one after another.

The first group "est" shows F_1-scores on the evaluation sets for the estimated threshold t_{est} with MTF values 20, 50, 100 and 200. The second group "max" presents maximum F_1-scores. They are calculated by increasing the threshold t from 0 to 1 in small steps, computing the F_1-score for each threshold value and taking the maximum $\max_t F_1(t)$.

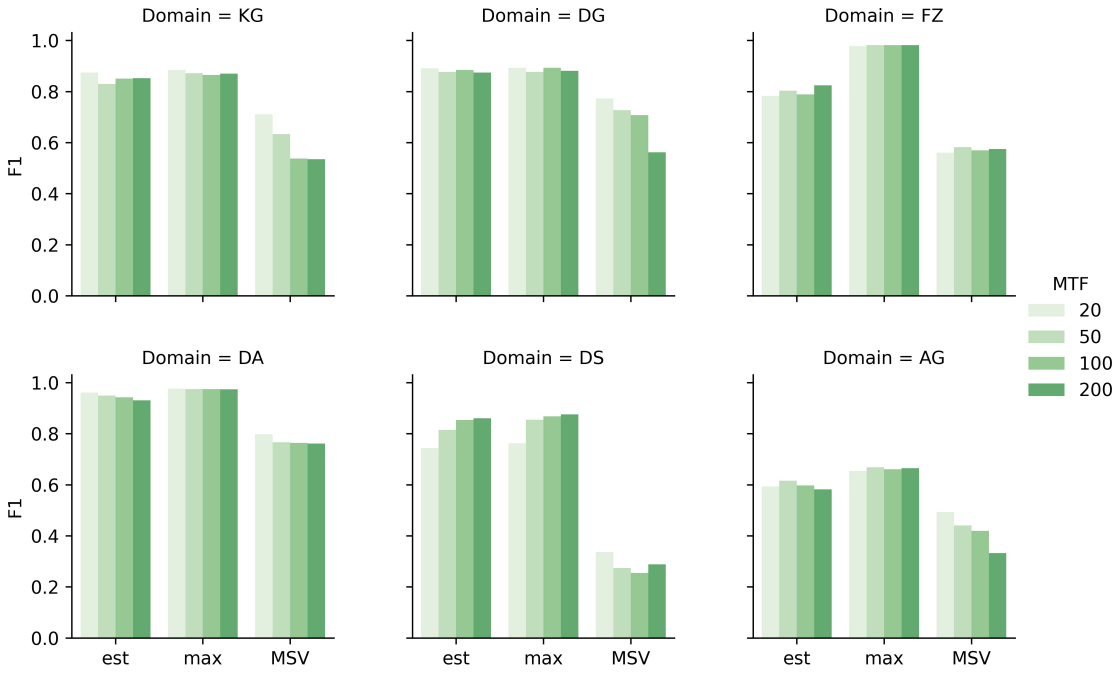


Figure 5: *Dependency of F_1-scores on maximum token frequency (MTF) and on AutoCal's steps for both threshold estimation ("est") and maximum similarity vectors ("MSV")*

The maximum F_1-scores are slightly larger than their counterparts in the first group — except for small-sized data sets of FZ. The large difference for the "easiest" scenario FZ is caused by a relatively small number of matches compared to the bin width Δ which we fixed to 0.025 for all six IM scenarios. This leads to sparsely populated bins and hence, to an inaccurate estimation of the threshold. In contrast, the threshold estimation step achieves sufficiently accurate results for mid-sized data sets.

The F_1-scores in both groups are surprisingly stable for a wide range of MTF-values — except for DS in combination with MTF $= 20$. This indicates that MTF $= 50$ is a reasonable choice when performing instance matching with AutoCal on other mid-sized data sets. Therefore, we use MTF $= 50$ as the default value in the following. Table 3 shows the resulting number of tokens and candidate pairs and AutoCal's matching quality in terms of precision, recall, and F_1-score for the estimated threshold t_{est}.

The results in the last group "MSV" of Figure 5 are obtained only when relying on AutoCal's first step. In this case, a candidate pair is predicted as a match if and only if its similarity vector is a maximum similarity vector. The resulting F_1-scores do not depend on any threshold. They indicate that maximum similarity vectors – taken in isolation – do not provide acceptable F_1-scores. Nevertheless, computing maximum similarity vectors is necessary to obtain an empirical distribution that produces useful auto-calibrated scores for all candidate pairs.

Table 3: *The number of tokens and candidate pairs due to token-blocking for MTF = 50 and resulting precision, recall and F_1-scores in % for AutoCal.*

ID	# Tokens	# Candidates	P	R	F_1
KG	962	10,392	95.1	73.5	82.9
DG	1,043	13,509	90.6	84.9	87.7
FZ	727	8,979	67.1	100.0	80.3
DA	6,647	202,565	97.8	92.2	94.9
DS	4,238	124,545	92.5	72.7	81.4
AG	1,781	54,831	61.7	61.5	61.6

5.4 Comparison

In this section, we compare AutoCal's matching quality with some unsupervised, semi-supervised and supervised methods of instance matching. Table 4 summarises published F_1-scores for FZ, DA, DS and AG. When a result was achieved by using at most 2 % of labelled matches from the complete ground truth, we assigned it to the upper category, otherwise to the lower category. The highest score for each IM scenario in each category is underlined. The "Setup" column provides additional information that might be helpful for a fair comparison: "strat. 60 % + 20 %" means that stratified sampling was applied to create training and validation sets with 60 % and 20 %, resp., of the labelled candidate pairs while the remaining 20 % were used for testing, "CV" stands for "cross validation", and if no percentage is specified, it means no manual labelling is required. There are further relevant differences that are not shown in the table such as the use of various blocking methods and sets of similarity functions.

AutoCal belongs to the upper category; the first row shows its F_1-scores copied over from Table 3 for convenient comparison. All other methods in the upper category were already introduced in section 3.

We also added supervised ML and DL methods as the second category to Table 4 since they define an upper limit for the matching quality and give an idea of what we can presently expect at most in an automated environment. The second category starts with Wombat [34] as a state-of-the-art approach for link discovery; this time, the results shown were achieved using 30 % of the labelled matches. Magellan [22] is an ML framework for instance matching which integrates various methods for blocking, feature generation and matching (such as Random Forests and Support Vector Machine). The last three rows refer to state-of-the-art DL models DeepMatcher [28], DITTO [27] and DeepER [12]. Supervised methods usually achieve higher F_1-scores than those from the upper category but require a large number of labelled matching pairs. It is out of scope here to describe these models in detail; instead, we refer to [2] for a survey of DL instance matchers.

The upper category of Table 4 shows that unsupervised methods employing embeddings and deep learning, i.e. EmbDI [5], ErGAN [33] and CollaborEM [16], frequently achieve higher F_1-scores. In particular, self-supervised CollaborEM obtains the best scores for FZ, DA and AG. DL approaches benefit fully from high-dimensional embedding vectors instead of only using the cosine similarity between them. AutoCal achieves F_1-scores

Table 4: *F_1-scores in % for unsupervised, semi-supervised and supervised approaches. The best F_1-score for each scenario in each category is underlined.*

	Method	Setup	FZ	DA	DS	AG
zero and low resources	AutoCal	t_{est}	80.3	94.9	81.4	61.6
	Auto-FuzzyJoin [26]		88.9	97.7	-	-
	SBC [21]	2 % seed matches	94.7	94.7	-	43.0
	Wombat [34]	2 % seed matches	98	94	-	53
	Ensemble Learning [19]		94	92	55	-
	ZeroER [35]		<u>100</u>	96	86	48
	EmbDI [5]		99	95	<u>92</u>	59
	ErGAN [33]	1 % seed matches	-	96.9	85.9	-
	CollaborEM [16]		<u>100</u>	<u>98.6</u>	-	<u>68.6</u>
mid to high	Wombat [34]	30 % matches	97	95	91	54
	Magellan [22] [28]	strat. 60 % + 20 %	<u>100</u>	98.4	92.3	49.1
	DeepMatcher [28]	strat. 60 % + 20 %	<u>100</u>	98.4	94.7	69.3
	DITTO [27]	strat. 60 % + 20 %	<u>100</u>	<u>99.0</u>	95.6	75.6
	DeepER [12]	10 % + 5-fold CV	<u>100</u>	98.6	<u>97.7</u>	<u>96.0</u>

which are competitive to results for ML methods, but at first sight, DL approaches seem to achieve superior matching results.

On the other hand, DL approaches often interpret numerical data such as product prices as strings and missing data as empty strings. Since the power plant scenarios KG and DG contain numerical data and short strings, it is not clear whether unsupervised DL approaches can use this information effectively and achieve better performance for KG and DG – out of the box, to be specific – than AutoCal. Moreover, we are also interested in further aspects that are relevant when applying instance matching in real-world projects. For this reason, we will have a deeper look at these aspects in the next subsection and choose the state-of-the-art DL matcher CollaborEM for intensive study.

5.5 Results Regarding Applicability and Runtime Behaviour

This section compares AutoCal and CollaborEM in more detail and focuses on applicability in real-world projects as well as runtime behaviour. For this purpose, the number of epochs is crucial for CollaborEM's training process, i.e., for fine-tuning a pre-trained language model. According to [16], the number of epochs depends on the size of the data sets, but it is not specified how to choose the number in case of new datasets such as KG and DG. We thus run CollaborEM in two ways.

Firstly, we run CollaborEM as described in [16] but always for 30 epochs. After each epoch, we apply the fined-tuned model to the evaluation set and calculate the F_1-score. Finally, we take the maximum F_1-score over all 30 epochs. Since the maximum slightly depends on the initial random seed, we run CollaborEM for initial values 1 to 5. The second row of Table 5 shows the resulting maximum F_1-scores averaged over these five runs.

The average scores are close to the published results for CollaborEM as shown in Table 4. However, the evaluation set consists of ground truth matches and nonmatches which are not available in an automated environment or in the context of unsupervised methods. Thus, the evaluation set cannot be consulted for selecting the best epoch. Consequently, the values in the second row should rather be interpreted as upper bounds for F_1-scores achievable in real-world projects. The first line shows AutoCal's maximum F_1-scores for MTF $= 50$ which are also upper bounds and can only be obtained by an optimal procedure for threshold estimation. Please note that averaging is not necessary in the case of AutoCal because it is a completely deterministic algorithm which does not depend on any random seed.

Table 5: *Maximum F_1-scores and F_1-scores in practice (in %) for AutoCal and CollaborEM.*

Method	**Setup**	**FZ**	**DA**	**DS**	**AG**	**KG**	**DG**
AutoCal	maximum over $t \in [0,1]$	98.2	97.4	85.4	66.8	87.2	87.7
CollaborEM	maximum over 30 epochs	100.0	98.3	66.9	67.1	86.5	88.3
AutoCal	estimated threshold t_{est}	80.3	94.9	81.4	61.6	82.9	87.7
CollaborEM	validation 30 epochs	94.1	97.1	63.5	66.2	81.4	86.6
CollaborEM	validation 10 epochs	94.1	96.6	62.5	64.1	80.1	85.9

Secondly, we consider the following approach: At the beginning, CollaborEM automatically generates a seed set of matches and nonmatches. Instead of using the entire seed set for fine-tuning, we use only 90 % for fine-tuning and the rest as the validation set. Again, we run CollaborEM for 30 epochs, but this time we apply the fine-tuned model to the validation set instead of the evaluation set after each epoch. Then, we choose the epoch ≤ 30 with the maximum F_1-score on the validation set. If the maximum value results from more than one epoch, then we choose the smallest of those epochs. We finally apply the fine-tuned model corresponding to the selected epoch for instance matching and calculate the F_1-score on the evaluation set. The fourth row of Table 5 shows the resulting F_1-scores - again averaged over five runs. The last row shows the analogous results for running CollaborEM for only 10 epochs. Not surprisingly, the averaged scores tend to decrease.

For the sake of comparison, the third row contains AutoCal's F_1-scores for the estimated threshold t_{est}, copied over from Table 3. The best score for each IM scenario within the lower block is underlined. AutoCal's F_1-scores for KG and DG are slightly better than CollaborEM's F_1-scores. However, AutoCal's F_1-score for DS is even significantly larger than CollaborEM's maximum F_1-score. We did not vary the ratio for the train-validation split or apply any further criteria for selecting the "best" epoch, possibly improving CollaborEM's F_1-scores. Nevertheless, Table 3 illustrates AutoCal's matching quality in real-world projects and indicates that none of both methods is clearly superior when applied in practice.

All matching results presented so far were achieved on a machine in the cloud with 40 GB GPU memory. Another important aspect in practice is the runtime behaviour which we checked additionally on a complementary environment: a laptop with 8 GB GPU memory. Table 6 summarises the total times (in seconds).

Table 6: *Runtime in seconds of AutoCal and CollaborEM in two different hardware environments.*

	Method	Setup	FZ	DA	DS	AG	KG	DG
Cloud	AutoCal		32	464	548	113	31	46
	CollaborEM	validation 10 epochs	99	1,231	2,982	310	364	275
Laptop	AutoCal		18	260	369	71	19	24
	CollaborEM	validation 10 epochs	393	4,588	5,687	1,103	1,145	1,143

AutoCal makes use of the GPU only when computing sentence embeddings [32][2] for string-valued properties (using the CPU for this task is possible but slower). Due to an efficient implementation of this task, AutoCal benefits from a powerful GPU with large memory only for a small fraction of its total time. On the other hand, AutoCal runs faster on the laptop due to a CPU being more powerful than the CPU on the machine in the cloud.

In contrast, CollaborEM's runtime behaviour relies heavily on a powerful GPU, allowing large batches for fine-tuning the language model. According to Table 5, fine-tuning over 10 epochs comes with an acceptable loss in matching quality. Therefore, we also consider only 10 epochs when computing CollaborEM's total time. The total times in Table 6 contain all of CollaborEM's processing steps, except the computation of F_1-scores on the evaluation set. Contrary to AutoCal, the total times increase when running CollaborEM on the laptop since fine-tuning contributes to the largest portion of the total time and slows down with a smaller batch size (16 instead of 64 samples) on a less powerful GPU.

A comparison of the total times in Table 6 shows that AutoCal runs at least 2.6 times faster than CollaborEM in the cloud. On the laptop, AutoCal performs even 15 to 20 times faster for FZ, DA, DS and AG, and 60 and 47 times faster for KG and DG, respectively.

6 Integration into the World Avatar

AutoCal was designed as a first step towards the automated merging and linking of data from various providers, and towards continuously populating the knowledge graph of the World Avatar (TWA). This section summarises TWA's core principles and presents how AutoCal is integrated into TWA. Finally, we demonstrate how the prototype is applied in TWA in the domain of power plants.

6.1 Core Principles and Prototype

TWA is built on the Semantic Web [3] and the principles of Linked Data [4]. These approaches and related semantic technologies are designed to integrate distributed data from various domains. TWA consists of two layers: the knowledge graph and the active agents.

We first describe the knowledge graph which represents the semantically annotated data of TWA. We refer to [1], [13] and [37] for details of TWA and its architecture.

An ontology is a collection of classes and properties in a domain of interest. It can be considered a vocabulary used to express statements about the instances of this domain and about the relations between instances of this domain or other domains. Each statement is expressed as a triple consisting of a subject, a predicate and an object. For instance, power plant *a*3 which was already introduced in Figure 2 may be described by triples such as *(a*3*, type, FossilFuelPlant)* and *(a*3*, name, "moabit")*. Here, *type* and *name* denote properties, *a*3 is an instance of type (class) *FossilFuelPlant* and *"moabit"* is a plain string. Since each triple can be regarded as two nodes (the subject and the object) connected by an edge (the predicate), all triples in total form a large graph: the knowledge graph.

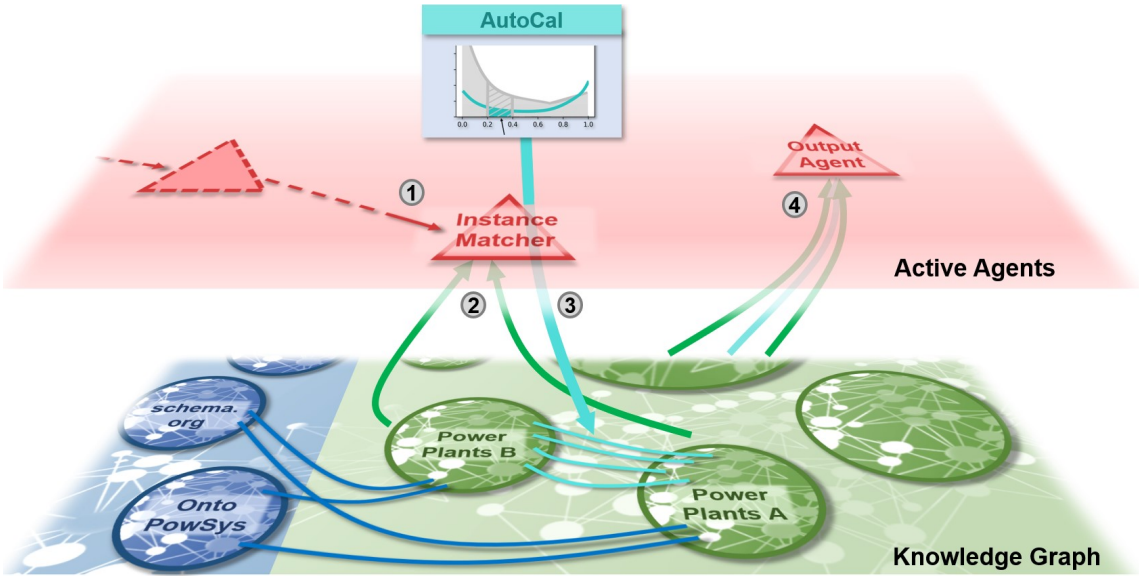


Figure 6: *Architecture of the World Avatar using the example of the instance matching agent and triples for the power plant domain*

Figure 6 illustrates TWA's core architecture and the prototypical integration of instance matching. The lower layer of Figure 6 shows some part of the knowledge graph that is relevant for our prototype. Triples are usually organised in topical sets which can be stored on different servers. The triples in the green circles use different vocabularies visualised as blue circles, e.g. *a*3 is an instance of class *FossilFuelPlant* which in turn is part of the class hierarchy defined by OntoPowSys [11], an ontology for describing power systems. Another example is the address of *a*3 which is described by means of properties *streetAddress*, *postalCode* and *addressLocality* from the *schema.org* vocabulary[7]. These references to classes and properties are indicated by blue lines between green and blue circles.

TWA uses the Resource Description Framework (RDF)[8] and the Web Ontology Language (OWL)[9] to define properties and classes, and to describe instances formally. We refer to

[7] https://schema.org/
[8] https://www.w3.org/TR/rdf11-primer/
[9] https://www.w3.org/TR/owl2-overview/

appendix A.1 for the formal and complete description of *a*3 as an example. While we use short terms in this work for better readability, TWA actually employs Uniform Resource Identifiers (URIs) according to RDF and OWL. Each URI identifies either an instance, a property or a class; for example, the complete identifier for class *FossilFuelPlant* is stated as:

```
http://www.theworldavatar.com/ontology/ontopowsys/
                PowSysRealization.owl#FossilFuelPlant
```

Due to the principles of Linked Data, this URI can be used to retrieve further information about *FossilFuelPlant* by means of an HTTP request. In this way, the knowledge graph becomes distributed over the World Wide Web, can link to other published data sources and knowledge graphs, and can be accessed and understood by other applications.

The upper layer of Figure 6 illustrates how agents operate on the knowledge graph. They call each other by HTTP requests, query and update data in the knowledge graph by means of SPARQL[10], a semantic query language.

For the prototype, we implemented a new *Instance Matcher* agent which wraps AutoCal and the matching pipeline from Figure 1. The agent requires a configuration set that specifies data sets A and B, and parameters for token-blocking, property mapping, and optional evaluation. In section 5, the data sets A and B were provided in Magellan's CSV format and described by the same column names which simplifies the evaluation of AutoCal and CollaborEM. Here, the matching context is different: Both data sets are triple sets. Data set A contains new data that we want to integrate into the knowledge graph while data set B is a suitable existing subset of the knowledge graph. As soon as the *Instance Matcher* agent is called (step 1 in Figure 6), it reads the triple sets (step 2), performs the pipeline and adds a triple of the form *(a, sameAs, b)* to the knowledge graph for each predicted match (a,b) (step 3). The property *sameAs* is a standard property defined as part of OWL. It declares instances a and b as the same. In Figure 6, the turquoise lines between the triple sets for A and B indicate resulting *sameAs*-triples.

The upper layer also shows an *Output Agent* which in this case, allows a user to query or visualise data from the knowledge graph (step 4). This agent could be a standard SPARQL query browser to query matched power plants for example or a tool to visualise the geographical distribution of power plants as demonstrated in the next subsection.

TWA aims for a high degree of automation. What are the implications for instance matching? First of all, neither AutoCal nor CollaborEM requires manually labelled instance pairs. Moreover, we showed that AutoCal's default parameter values ensure an acceptable matching quality for a wide range of IM scenarios. This suggests that AutoCal can be applied to new incoming data (from other domains) out-of-the-box, i.e., without the need to fine-tune parameters on an extra set of labelled pairs. As pointed out in subsection 5.5, CollaborEM must be equipped with a criterion for selecting the "best" epoch. Once this is done, its default parameter values can be used likewise for instance matching in real-world projects.

[10] https://www.w3.org/TR/sparql11-query/

In general, the names of the respective columns or properties describing new data and data in the knowledge graph differ. This requires preliminary property matching. In future, an agent could automate this task by applying existing methods from the field of schema matching. Figure 6 illustrates such an agent by dashed lines. Likewise, the choice of similarity functions in the case of AutoCal and the choice of graph features in the case of CollaborEM could be automated by analysing the type and range of data for each column or property.

In summary, both AutoCal and CollaborEM are suitable building blocks for automated instance matching within TWA. While none of them is superior with respect to matching quality they significantly differ regarding runtime behaviour and hardware requirements. Contrary to other published knowledge graphs such as DBpedia [25], TWA knowledge graph is not centralised but its triples are distributed over several Web nodes. Moreover, its agents may run on the same nodes which often are server machines with strong CPUs but less powerful GPUs. Since AutoCal allows faster instance matching on the same hardware, it is particularly suited for deployment on TWA.

6.2 Use Case

This section presents how the instance-matching prototype was applied to the power plant data instantiated in TWA knowledge graph. Prior to applying the instance matching, the power plant instances in TWA knowledge graph are mainly fossil fuel types based on data obtained from the Global Energy Observatory[11], Global Power Plant Database[12] and Enipedia[13]. The instance matching prototype creates an *owl:sameAs* semantic link between the existing TWA power plant instances and its equivalent within the KWL and DUKES datasets. 111 pairs of German power plants and 30 pairs of UK power plants have been identified by the instance matching prototype to be matches. Consequently, TWA can now simultaneously query data present in the KWL and DUKES datasets for power plants in Germany and the UK, respectively, in addition to its original power plant data. Figure 7 presents a map of the geographical distribution of the fossil fuel power plants in the UK. The map was created by a visualisation agent which queries the fossil fuel power plants and their geographic coordinates directly from TWA knowledge graph.

Tables 7 and 8 show examples of additional data queried from TWA before and after instance matching for a power plant, each from Germany and the UK, respectively. As shown in Tables 7 and 8, instance matching may increase the amount of data that can be queried in TWA. This is important as it allows relevant knowledge of the same entity in the real world that is represented in numerous forms to be linked, thereby creating a more comprehensive world model. For example, as shown in Table 7, for the Gersteinwerk Block K power plant, the KWL dataset augmented the original data in TWA knowledge graph with more precise location information, i.e. Werne, Nordrhein-Westfalen, 59368. Moreover, additional information was also retrieved for the fuel type, year of construction and design capacity for the Gersteinwerk Block K power plant. Upon further investiga-

[11] http://globalenergyobservatory.org
[12] https://datasets.wri.org/dataset/globalpowerplantdatabase
[13] https://archive.ph/20140610231532/http://enipedia.tudelft.nl/

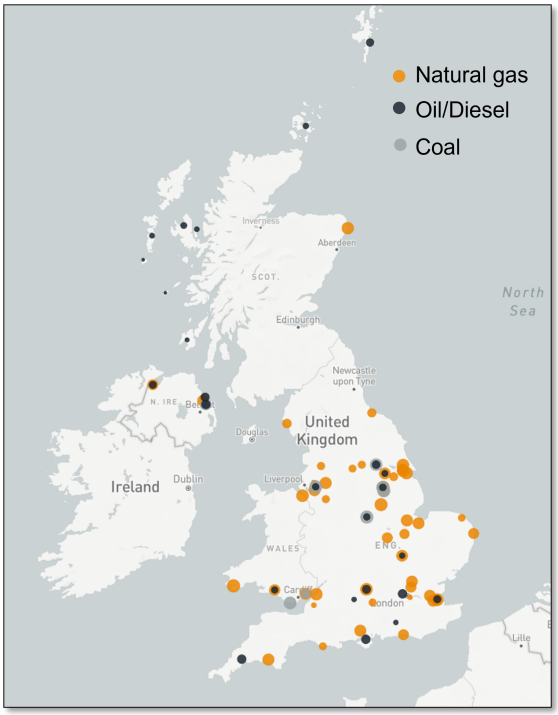


Figure 7: *Geographical distribution of fossil fuel power plants in the UK*

Table 7: *Examples of additional data queried from TWA before and after the instance matching for the Gersteinwerk Block K Power Plant in Germany.*

	Before matching	**After matching**
Coordinates	51.6739, 7.7195	51.6739, 7.7195; 51.66318, 7.635119
Address	Germany	Germany; Werne, Nordrhein-Westfalen, 59368
Fuel	Coal	Coal; Gas
Year of Construction	1981	1981; 1984
Design Capacity (MW)	770	770; 112
Owner	RWE	RWE; RWE Generation SE

Table 8: *Examples of additional data queried from TWA before and after the instance matching for the Damhead Creek Power Plant in the UK.*

	Before matching	**After matching**
Coordinates	51.424915, 0.601426	51.424915, 0.601426; 51.42, 0.601
Year of Construction	1996	1996; 2000
Design Capacity (MW)	140	140; 805
Owner	RWE_npower_plc	RWE_npower_plc; Drax Power

tion, it is revealed that the Gersteinwerk Block K power plant consists of: 1) a K1 block that uses natural gas as the fuel and 2) a K2 block that uses coal as the fuel[14]. The Gersteinwerk Block K power plant has been predominately generating electricity using coal from the nearby collieries since its operation till March 2019, when the K2 block, the coal-fired part of the plant, was decommissioned[15]. The K1 block, the part that uses natural gas, continues to operate. The original data in TWA knowledge graph reflects the status of the Gersteinwerk Block K power plant before the decommissioning of the K2 block, while the KWL dataset reflects the status of the Gersteinwerk Block K power plant after the decommissioning of the K2 block.

7 Conclusions

In the first part of this paper, we presented AutoCal, a new algorithm for unsupervised instance matching of tabular data sets. AutoCal consists of three steps based on heuristics and statistics: It computes maximum similarity vectors on subsets of paired instances sharing tokens. It uses their empirical marginal distributions to derive calibrated property scores. Finally, it estimates a matching threshold for the aggregated property scores. Token-blocking suits AutoCal's first step in a natural way and is applied – in combination with the maximum token frequency – as a preparatory step to reduce the number of candidate pairs to a feasible size.

We evaluated AutoCal with six different scenarios for instance matching: Two new scenarios with power plant entities exhibiting a mixture of properties with short string, numerical and missing values, as well as four scenarios from other domains which are frequently used in the literature and which contain a large portion of text. We collected a wide variety of state-of-the-art methods for instance matching for a comprehensive comparison with AutoCal and summarised the results published for the four scenarios. Our results for AutoCal show that its matching quality is competitive to recent unsupervised and semi-supervised instance matchers from the field of Machine Learning.

Unsupervised Deep Learning methods frequently achieve better matching results on said four common scenarios. Within this class of methods, we selected CollaborEM as a state-of-the-art matcher for a detailed comparison with AutoCal. Both CollaborEM equipped with a stop criterion and AutoCal can be applied out-of-the-box for new matching tasks without manually adapting default values for method-specific parameters. Unexpectedly, it turned out that neither AutoCal nor CollaborEM are superior regarding matching quality. Moreover, AutoCal runs 2.7 up to 60 times faster than CollaborEM for the six matching scenarios.

While AutoCal can run stand-alone, we illustrated in the second part of the paper how the new method is integrated prototypically into the World Avatar (TWA) as an agent. TWA is a dynamic knowledge graph approach for digital twins and aims for a high degree of automation. Since AutoCal achieves competitive matching results and can run out of

[14] https://de.wikipedia.org/wiki/Gersteinwerk

[15] https://www.rwe.com/en/the-group/countries-and-locations/gersteinwerk-power-plant

the box, it is an important step towards automatically and continuously populating the knowledge graph with new or updated data from external sources. TWA's knowledge graph and agents are distributed over several Web nodes. Due to its runtime behaviour and moderate hardware requirements, AutoCal is particularly suitable to run on the same decentralised infrastructure.

Finally, we demonstrated how AutoCal matches new data from the power plant domain with existing instances in the knowledge graph. In the presence of additional information and/or discrepancies in data, the current instance matching prototype is not yet able to determine whether they are explainable or pure data errors. Despite this, the prototype has increased the amount of power plant data that can be queried in TWA. It can highlight data that need more investigation. As the next step, we will extend existing ontologies and develop new ontologies to integrate publicly available data, which can act as supplementary information, into TWA to discern such situations. We will also examine state-of-the-art approaches to automatically discover data relevant to TWA on the World Wide Web.

Research data

Research data supporting this publication is available in the University of Cambridge data repository (doi:10.17863/CAM.82548).

Acknowledgements

This project is funded by the National Research Foundation (NRF), Prime Minister's Office, Singapore under its Campus for Research Excellence and Technological Enterprise (CREATE) programme. Part of this work was supported by Towards Turing 2.0 under the EPSRC Grant EP/W037211/1 & The Alan Turing Institute. M.K. gratefully acknowledges the support of the Alexander von Humboldt Foundation. The authors thank Dr Feroz Farazi for discussions, Jörg Preisendörfer for commenting extensively on the paper, Andrew Breeson for proofreading, and Wanni Xie for providing semantically annotated data.

Nomenclature

AG	IM scenario for products - Amazon vs. Google Products
DA	IM scenario for bibliography - DBLP vs. ACM
DG	IM scenario for power plants in the United Kingdom
DL	Deep Learning
DS	IM scenario for bibliography - DBLP vs. Google Scholar
FZ	IM scenario for restaurants - Fodor vs. Zagat
GPPD	Global Power Plant Database
HTTP	Hypertext Transfer Protocol
IM	Instance Matching
KG	IM scenario for power plants in Germany
ML	Machine Learning
MTF	Maximum Token Frequency
OWL	Web Ontology Language
RDF	Resource Description Framework
SPARQL	SPARQL Protocol and RDF Query Language
SVM	Support Vector Machine
TWA	The World Avatar
URI	Uniform Resource Identifier

A Appendix

A.1 RDF Graph Example

The code shown below demonstrates how an entry (record) of the data source KWL (see Table 2) is represented in the knowledge graph of the World Avatar. Although the World Avatar can import data in a variety of syntaxes, we use the *Turtle*[16] syntax here for demonstration, since it is particularly human readable. The data description uses terms from different vocabularies (such as *OntoPowSys, OntoCAPE, schema.org*). Each column is mapped to a suitable property, nested in part by means of the brackets notation "[...]" for blank nodes[17].

```
@prefix base: <http://www.theworldavatar.com/kb/powsys/kwl/> .
@prefix psreal: <http://www.theworldavatar.com/ontology/
  ontopowsys/PowSysRealization.owl#> .
@prefix cpsys: <http://www.theworldavatar.com/ontology/
  ontocape/upper_level/system.owl#> .
@prefix sdo: <https://schema.org/> .
...

base:BNA0085a_Moabit_DE a psreal:FossilFuelPlant;
  rdfs:label "Moabit"@en ;
  dbo:country dbr:Germany ;
  cptecsys:realizes [ pow:consumesPrimaryFuel "Coal" ] ;
  pow:hasYearOfBuilt [ cpsys:hasValue [
    cpsys:numericalValue 1990 ] ] ;
  eipreal:designCapacity [ cpsys:hasValue [
    cpsys:hasUnitOfMeasure cpunit:MW ;
    cpsys:numericalValue 8.9e+01 ] ] ;
  cpsysv1:isOwnedBy [
    cpsysv1:hasName "Vattenfall Wärme Berlin AG" ] ;
  sdo:address [
    sdo:addressCountry "DE" ;
    sdo:addressLocality "Berlin" ;
    sdo:addressRegion "Berlin" ;
    sdo:postalCode 13353 ;
    sdo:streetAddress "Friedrich-Krause-Ufer 10- 15" ] .
```

[16] https://en.wikipedia.org/wiki/Turtle_(syntax)
[17] https://en.wikipedia.org/wiki/Blank_node

A.2 Pseudocode for AutoCal with Token-Blocking

Algorithm 1 shows the pseudocode for AutoCal. Its main part AUTOCAL-WITH-TOKEN--BLOCKING corresponds to the steps in Figure 1 while subsequent procedures shown in algorithm 1 describe its steps in detail. For completeness, we included the steps for token-blocking and for computing the similarity vectors in lines 1–4. The pseudocode for creating the token-based index (lines 13–22), for creating the set of candidate pairs (lines 23–31) and for computing the similarity vectors (lines 32–38) is straight-forward and follows the explanation given in sections 4.1 and 4.2. Thus, we focus on the three sub steps of AutoCal here.

Line 5 calls COMPUTE-MAX-SIM-VECTORS with the set *CandPairs* (the result of CREATE--CANDIDATE-PAIRS) and with the similarity vectors s (the result of COMPUTE-SIM--VECTORS). Lines 39–49 only apply definition 3 and create the maximum similarity vectors s^{max} in both directions, i.e. for all $a \in A$ and for all $b \in B$. By contrast, lines 6 and 7 call REPLACE-DISTRIBUTION for each direction separately, once for computing $score^A$ and once for computing $score^B$.

The pseudocode for procedure REPLACE-DISTRIBUTION in lines 50–78 expects as input the set E of instances (which is either A or B), the candidate pairs, the similarity vectors s, the maximum similarity vectors s^{max}, as well as the bin width Δ and the mapping triples. In accordance with figure 4, this procedure computes property scores as estimated ratio between match and nonmatch counts, given a candidate pair (a,b) and its similarity values $s_k(a,b), k = 1,\ldots,K$.

Since calculating the counts for each candidate pair was very inefficient, AutoCal calculates them only once for a small subset of equidistant points of the form $t = 2\Delta j$ as in lines 53 and 54. The counts themselves are computed in lines 56 and 57 for the k-th property: Line 56 counts the instances for which the k-th component of their maximum similarity vectors is in the Δ-neighbourhood of t. This count refers to the turquoise area in the right upper diagram of Figure 4 and is used as rough estimate for the counts with respect to the unknown matches. Line 57 counts all candidate pairs (matches and nonmatches) for which the k-th component of their similarity vectors is in the Δ-neighbourhood of t. This count refers to the grey hatched area in the right upper diagram. Finally, the property ratio is computed in line 58. Since the count in the nominator may become larger than the count in the denominator, line 58 takes the maximum to restrict the ratio to 1. However, the effect of this clamping to 1 is negligible.

The second part of procedure REPLACE-DISTRIBUTION starts with the computation of the numbers of different and same property pairs (lines 61–65) which are needed for averaging (line 75 and 72, resp.). Afterwards, it iterates over all candidate pairs (line 67). For each similarity value $s_k(a,b)$, line 71 computes the integer j for the closest $t = 2\Delta j$. The corresponding property ratios computed in the first part are aggregated by summing them up (lines 69 and 72) and by taking the average over all different pairs (lines 75). These lines produce the auto-calibrated property scores. The averaging in line 72 has the effect that the same scores are computed even if a triple was added again to the mapping triples. Finally, line 77 returns the resulting scores for all candidate pairs.

Before presenting the last procedure, we would like to note that the ratio in line 58 is re-

lated to conditional probabilities: Suppose random variables $B_1, \ldots, B_J$ take values 0 (nonmatch) and 1 (match) with probability q and $1-q$, resp., and $X_1^0, \ldots, X_J^0$ and $X_1^1, \ldots, X_J^1$ are distributed on $[0,1]$ with probability $p^0(\cdot)$ and $p^1(\cdot)$, resp. All random variables are supposed to be independently distributed. Moreover, set $Y_j = X_j^0$ if $B_j = 0$ and $Y_j = X_j^1$ if $B_j = 1$. For a given interval $I \subset [0,1]$, sufficiently large J and a fixed l with $1 \leq l \leq J$, the conditional probability for $B_l = 1$ given $Y_l \in I$ can be approximated by

$$\text{Prob}(B_l = 1 | Y_l \in I) = \frac{\text{Prob}(B_l = 1, Y_l \in I)}{\text{Prob}(Y_l \in I)} \approx \frac{\#\{j : 1 \leq j \leq J, B_j = 1, Y_j \in I\}}{\#\{j : 1 \leq j \leq J, Y_j \in I\}} \,. \quad \text{(A.1)}$$

AutoCal chooses a small interval I (of size 2Δ) containing $s_k(a,b)$ and replaces the unknown count of matches in the nominator of eq. A.1 by the count in line 56 and the count in the denominator by the count in line 57. Consequently, the ratio in line 58 can be interpreted as rough approximation of the probability that a candidate pair – given its similarity value $s_k(a,b)$ – is a match.

Line 8 in the main algorithm calls the last procedure `ESTIMATE-THRESHOLD` with scores $score^A(a,b)$ and $score^B(a,b)$, previously computed for all candidate pairs (a,b). Lines 79–100 show the implementation of the procedure. It requires the same parameter Δ as before. The code in lines 80–82 determines potential matches as pairs with maximum scores among those sharing a token. The remaining candidate pairs are regarded as nonmatches (line 83). Lines 85–87 define the total score for each candidate pair as the maximum of $score^A(a,b)$ and $score^B(a,b)$. In lines 89 and 90 the threshold t is iterated over equidistant points of the form $2\Delta j$ (as in lines 53 and 54). Line 91 counts the potential matches with total scores in the Δ-neighbourhood of t and line 92 does the same for the remaining nonmatches. In line 93, the ratio of both counts is computed and in line 95, the threshold t_{est} is determined as the minimum t for which the ratio is above 1 three times in sequence. After setting the score for nonmatches to 0 (lines 96–98), the updated scores and the estimated threshold are returned (line 99).

Finally, the returned scores and the threshold are used in line 12 of the main algorithm to predict the matches. If the optional ground truth in form of a set of matches was given as part of the input, line 10 calls an additional evaluation procedure (which is not specified here) to compute the matching performance in terms of precision, recall and F_1-score.

Algorithm 1 AUTOCAL-WITH-TOKEN-BLOCKING

Input:
data set A, data set B
subsets P_A, P_B of properties of A, B for tokenization
mapping triples $((p_k, q_k, f_k))_{k=1}^{K}$
maximum token frequency MTF
bin width Δ
ground truth for evaluation (optional)

$Index_A \leftarrow$ CREATE-INDEX(A, P_A)
$Index_B \leftarrow$ CREATE-INDEX(B, P_B)
$CandPairs \leftarrow$ CREATE-CANDIDATE-PAIRS$(Index_A, Index_B, MTF)$
$s \leftarrow$ COMPUTE-SIM-VECTORS$(CandPairs, MappingTriples)$
$s^{max} \leftarrow$ COMPUTE-MAX-SIM-VECTORS$(A, B, CandPairs, s)$
$score^A \leftarrow$ REPLACE-DISTRIBUTION$(A, CandPairs, s, s^{max}, \Delta, MappingTriples)$
$score^B \leftarrow$ REPLACE-DISTRIBUTION$(B, CandPairs, s, s^{max}, \Delta, MappingTriples)$
$score, t_{est} \leftarrow$ ESTIMATE-THRESHOLD$(CandPairs, score^A, score^B, \Delta)$
if $GroundTruth$ is given **then**
    EVALUATE$(score, t_{est}, GroundTruth)$
end if
return $\{(a,b) \in CandPairs : score(a,b) \geq t_{est}\}$

procedure CREATE-INDEX(E, P)
    $Index \leftarrow []$
    for $instance \in E$ **do**
        $TokenSet \leftarrow$ TOKENIZE$(instance, P)$
        for $token \in TokenSet$ **do**
            $Index[token] \leftarrow Index[token] \cup \{instance\}$
        end for
    end for
    return $Index$
end procedure

procedure CREATE-CANDIDATE-PAIRS$(Index_A, Index_B, MTF)$
    $CandPairs \leftarrow \emptyset$
    for token $t \in Index_A \cap Index_B$ **do**
        if $\#Index_A[t] \leq MFT$ AND $\#Index_B[t] \leq MFT$ **then**
            $CandPairs \leftarrow CandPairs \cup \{(a,b) : a \in Index_A[t], b \in Index_B[t]\}$
        end if
    end for
    return $CandPairs$
end procedure

procedure COMPUTE-SIM-VECTORS($CandPairs, MappingTriples$)
    $s \leftarrow []$
    for $(a,b) \in CandPairs$ **do**
        $s(a,b) \leftarrow ((f_k(a.p_k, b.q_k))_{k=1}^{K}$
    end for
    return s
end procedure

procedure COMPUTE-MAX-SIM-VECTORS($A, B, CandPairs, s$)
    $K \leftarrow$ Dimension of similarity vectors s
    $s^{max} \leftarrow []$
    for $a \in A$ **do**
        $s^{max}(a) \leftarrow (\max\{s_k(a,\tilde{b}) : (a,\tilde{b}) \in CandPairs\})_{k=1}^{K}$
    end for
    for $b \in B$ **do**
        $s^{max}(b) \leftarrow (\max\{s_k(\tilde{a},b) : (\tilde{a},b) \in CandPairs\})_{k=1}^{K}$
    end for
    return s^{max}
end procedure

procedure REPLACE-DISTRIBUTION($E, CandPairs, s, s^{max}, \Delta, MappingTriples$)
  $K \leftarrow$ dimension of similarity vectors s
  $ratio \leftarrow []$
  for $j \in \{0, \ldots, \lceil 1/2\Delta \rceil\}$ **do**
    $t \leftarrow 2\Delta j$
    for $k \in \{1, \ldots, K\}$ **do**
      $n^{max} \leftarrow \#\{e \in E : t - \Delta \leq s_k^{max}(e) \leq t + \Delta\}$
      $n \leftarrow \#\{(a,b) \in CandPairs : t - \Delta \leq s_k(a,b) \leq t + \Delta\}$
      $ratio(j,k) \leftarrow \max\{1,\ n^{max}/(n+1)\}$
    end for
  end for
  $diffPropPairs \leftarrow \#\{(p_k, q_k) : k = 1, \ldots, K\}$
  $samePropPairs \leftarrow []$
  for $k \in \{1, \ldots, K\}$ **do**
    $samePropPairs(k) \leftarrow \#\{l : (p_l, q_l) = (p_k, q_k)\}$
  end for
  $score \leftarrow []$
  for $(a,b) \in CandPairs$ **do**
    $sum \leftarrow 0$
    for $k \in \{1, ..., K\}$ **do**
      if $s_k(a,b)$ is not missing **then**
        $j \leftarrow \lfloor s_k(a,b)/(2\Delta) + 0.5 \rfloor$
        $sum \leftarrow sum + ratio(j,k)/samePropPairs(k)$
      end if
    end for
    $score(a,b) \leftarrow sum/differentPropPairs$
  end for
  return *score*
end procedure

79: **procedure** ESTIMATE-THRESHOLD($CandPairs, score^A, score^B, \Delta$)
80: $Best^A \leftarrow \{(a,b) \in CandPairs : score^A(a,b) = \max_{(a,\tilde{b}) \in CandPairs} score^A(a,\tilde{b}) \}$
81: $Best^B \leftarrow \{(a,b) \in CandPairs : score^B(a,b) = \max_{(\tilde{a},b) \in CandPairs} score^B(\tilde{a},b) \}$
82: $CandMatches = Best^A \cup Best^B$
83: $CandNonMatches = CandPairs \setminus CandMatches$
84: $score \leftarrow []$
85: **for** $(a,b) \in CandPairs$ **do**
86: $score(a,b) \leftarrow \max\{score^A(a,b), score^B(a,b)\}$
87: **end for**
88: $ratio \leftarrow []$
89: **for** $j \in \{0, \ldots, \lceil 1/2\Delta \rceil\}$ **do**
90: $t \leftarrow 2\Delta j$
91: $n_M \leftarrow \#\{(a,b) \in CandMatches : t - \Delta \leq score(a,b) \leq t + \Delta\}$
92: $n_N \leftarrow \#\{(a,b) \in CandNonMatches : t - \Delta \leq score(a,b) \leq t + \Delta\}$
93: $ratio(j) \leftarrow n_M/(n_N + 1)$
94: **end for**
95: $t_{est} = \min\{j : ratio(j) \geq 1, ratio(j+1) \geq 1, ratio(j+2) \geq 1\}$
96: **for** $(a,b) \in CandNonMatches$ **do**
97: $score(a,b) = 0$
98: **end for**
99: **return** $score, t_{est}$
100: **end procedure**

References

[1] J. Akroyd, S. Mosbach, A. Bhave, and M. Kraft. Universal digital twin – a dynamic knowledge graph. *Data-Centric Engineering*, 2, 2021.

[2] N. Barlaug and J. A. Gulla. Neural networks for entity matching: A survey. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 15(3):1–37, 2021.

[3] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.

[4] C. Bizer, T. Heath, and T. Berners-Lee. Linked data: The story so far. In *Semantic services, interoperability and web applications: emerging concepts*, pages 205–227. IGI global, 2011.

[5] R. Cappuzzo, P. Papotti, and S. Thirumuruganathan. Creating embeddings of heterogeneous relational datasets for data integration tasks. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1335–1349, 2020.

[6] A. Chadzynski, S. Li, A. Grisiute, F. Farazi, C. Lindberg, S. Mosbach, P. Herthogs, and M. Kraft. Semantic 3d city agents — an intelligent automation for dynamic geospatial knowledge graphs. *Energy and AI*, page 100137, 2022.

[7] P. Christen. Automatic record linkage using seeded nearest neighbour and support vector machine classification. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 151–159, 2008.

[8] P. Christen. *Data matching*. Springer, 2012.

[9] W. Cohen, P. Ravikumar, and S. Fienberg. A comparison of string metrics for matching names and records. In *KDD workshop on data cleaning and object consolidation*, volume 3, pages 73–78, 2003.

[10] J. De Bruin. Python Record Linkage Toolkit: A toolkit for record linkage and duplicate detection in Python, 12 2019. URL https://doi.org/10.5281/zenodo.3559043.

[11] A. Devanand, G. Karmakar, N. Krdzavac, R. Rigo-Mariani, Y. F. Eddy, I. A. Karimi, and M. Kraft. OntoPowSys: A power system ontology for cross domain interactions in an eco industrial park. *Energy and AI*, 1:100008, 2020.

[12] M. Ebraheem, S. Thirumuruganathan, S. Joty, M. Ouzzani, and N. Tang. Distributed representations of tuples for entity resolution. *Proceedings of the VLDB Endowment*, 11(11):1454–1467, 2018.

[13] A. Eibeck, M. Q. Lim, and M. Kraft. J-Park Simulator: An ontology-based platform for cross-domain scenarios in process industry. *Computers & Chemical Engineering*, 131:106586, 2019.

[14] F. Farazi, M. Salamanca, S. Mosbach, J. Akroyd, A. Eibeck, L. K. Aditya, A. Chadzynski, K. Pan, X. Zhou, S. Zhang, et al. Knowledge graph approach to combustion chemistry and interoperability. *ACS omega*, 5(29):18342–18348, 2020.

[15] A. Ferrara, A. Nikolov, and F. Scharffe. Data linking for the Semantic Web. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 7(3):46–76, 2011.

[16] C. Ge, P. Wang, L. Chen, X. Liu, B. Zheng, and Y. Gao. CollaborEM: A self-supervised entity matching framework using multi-features collaboration. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[17] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2):199–220, 1993.

[18] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. d. Melo, C. Gutierrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier, et al. Knowledge graphs. *Synthesis Lectures on Data, Semantics, and Knowledge*, 12(2):1–257, 2021.

[19] A. Jurek, J. Hong, Y. Chi, and W. Liu. A novel ensemble learning approach to unsupervised record linkage. *Information Systems*, 71:40–54, 2017.

[20] A. Jurek-Loughrey and P. Deepak. Semi-supervised and unsupervised approaches to record pairs classification in multi-source data linkage. In *Linking and Mining Heterogeneous and Multi-view Data*, pages 55–78. Springer, 2019.

[21] M. Kejriwal and D. P. Miranker. Semi-supervised instance matching using boosted classifiers. In *European semantic web conference*, pages 388–402. Springer, 2015.

[22] P. Konda, S. Das, P. S. GC, A. Doan, A. Ardalan, J. R. Ballard, H. Li, F. Panahi, H. Zhang, J. Naughton, et al. Magellan: Toward building entity matching management systems. *Proceedings of the VLDB Endowment*, 9(12), 2016.

[23] H. Köpcke and E. Rahm. Frameworks for entity matching: A comparison. *Data & Knowledge Engineering*, 69(2):197–210, 2010.

[24] H. Köpcke, A. Thor, and E. Rahm. Evaluation of entity resolution approaches on real-world match problems. *Proceedings of the VLDB Endowment*, 3(1-2):484–493, 2010.

[25] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. Van Kleef, S. Auer, et al. DBpedia – a large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic web*, 6(2):167–195, 2015.

[26] P. Li, X. Cheng, X. Chu, Y. He, and S. Chaudhuri. Auto-FuzzyJoin: Auto-program fuzzy similarity joins without labeled examples. In *Proceedings of the 2021 International Conference on Management of Data*, pages 1064–1076, 2021.

[27] Y. Li, J. Li, Y. Suhara, A. Doan, and W.-C. Tan. Deep entity matching with pre-trained language models. *arXiv preprint arXiv:2004.00584*, 2020.

[28] S. Mudgal, H. Li, T. Rekatsinas, A. Doan, Y. Park, G. Krishnan, R. Deep, E. Arcaute, and V. Raghavendra. Deep learning for entity matching: A design space exploration. In *Proceedings of the 2018 International Conference on Management of Data*, pages 19–34, 2018.

[29] M. Nentwig, M. Hartung, A.-C. Ngonga Ngomo, and E. Rahm. A survey of current link discovery frameworks. *Semantic Web*, 8(3):419–436, 2017.

[30] K. O'Hare, A. Jurek-Loughrey, and C. de Campos. A review of unsupervised and semi-supervised blocking methods for record linkage. *Linking and Mining Heterogeneous and Multi-view Data*, pages 79–105, 2019.

[31] G. Papadakis, E. Ioannou, C. Niederée, and P. Fankhauser. Efficient entity resolution for large heterogeneous information spaces. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 535–544, 2011.

[32] N. Reimers and I. Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. URL https://arxiv.org/abs/1908.10084.

[33] J. Shao, Q. Wang, A. Wijesinghe, and E. Rahm. ErGAN: Generative adversarial networks for entity resolution. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 1250–1255. IEEE, 2020.

[34] M. A. Sherif, A.-C. Ngonga Ngomo, and J. Lehmann. WOMBAT – a generalization approach for automatic link discovery. In *European Semantic Web Conference*, pages 103–119. Springer, 2017.

[35] R. Wu, S. Chaba, S. Sawlani, X. Chu, and S. Thirumuruganathan. ZeroER: Entity resolution using zero labeled examples. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1149–1164, 2020.

[36] L. Zhou, M. Pan, J. J. Sikorski, S. Garud, L. K. Aditya, M. J. Kleinelanghorst, I. A. Karimi, and M. Kraft. Towards an ontological infrastructure for chemical process simulation and optimization in the context of eco-industrial parks. *Applied Energy*, 204:1284–1298, 2017.

[37] X. Zhou, A. Eibeck, M. Q. Lim, N. B. Krdzavac, and M. Kraft. An agent composition framework for the J-Park Simulator – a knowledge graph for the process industry. *Computers & Chemical Engineering*, 130:106577, 2019.