LEAPME: Learning-based Property Matching with Embeddings

Daniel Ayala[a,*], Inma Hernández[a], David Ruiz[a], Erhard Rahm[b]

[a] *Universidad de Sevilla*
ETSII, Avda. Reina Mercedes, s/n. Sevilla, Spain
[b] *Leipzig University*
Institut für Informatik. Leipzig 04109, Germany

Abstract

Data integration tasks such as the creation and extension of knowledge graphs involve the fusion of heterogeneous entities from many sources. Matching and fusion of such entities require to also match and combine their properties (attributes). However, previous schema matching approaches mostly focus on two sources only and often rely on simple similarity measurements. They thus face problems in challenging use cases such as the integration of heterogeneous product entities from many sources.

We therefore present a new machine learning-based property matching approach called LEAPME (LEArning-based Property Matching with Embeddings) that utilizes numerous features of both property names and instance values. The approach heavily makes use of word embeddings to better utilize the domain-specific semantics of both property names and instance values. The use of supervised machine learning helps exploit the predictive power of word embeddings.

Our comparative evaluation against five baselines for several multi-source datasets with real-world data shows the high effectiveness of LEAPME. We also show that our approach is even effective when training data from another domain (transfer learning) is used.

Keywords: data integration, machine learning, knowledge graphs

1. Introduction

Data integration tasks such as the creation and refinement of knowledge graphs have to increasingly deal with the matching and fusion of data from many sources, e.g., different web sites, already created knowledge bases and repositories. Such knowledge graphs (KG) physically integrate numerous entities with their properties (attributes) and relationships as well as associated metadata about entity types and relationship types in a graph-like structure [1]. Many companies (including Google, Facebook, and Amazon) are increasingly relying on the integrated and curated information in knowledge graphs and there is also an increasing amount of research on KG creation [2, 3, 4, 5, 6, 7, 8, 9] and KG exploitation, e.g. for question answering [10, 11].

*Corresponding author
Email addresses: dayala1@us.es (Daniel Ayala), inmahernandez@us.es (Inma Hernández), druiz@us.es (David Ruiz), rahm@informatik.uni-leipzig.de (Erhard Rahm)

Integrating new data sources and their entities into a KG is challenging due to the typically large number of different kinds of entities and relationships, the high degree of heterogeneity in their representations and the often low data quality with frequently incomplete, wrong or contradicting information. Subproblems to deal with include the categorization, matching, clustering and fusion of entities. These steps in turn also require to match the properties of entities, e.g., to focus entity matching on comparable properties or to fuse the values of equivalent properties.

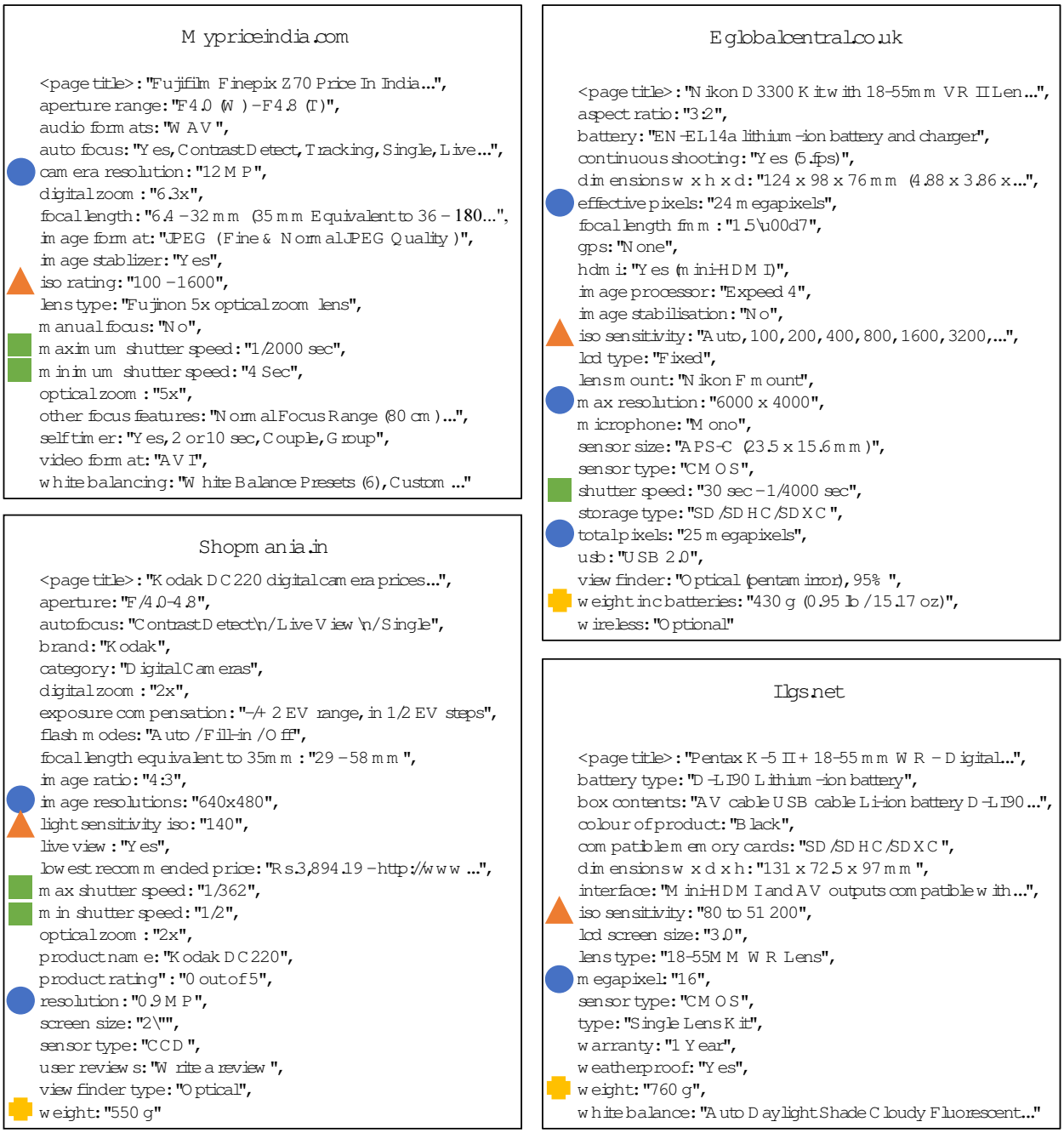


Figure 1: Camera properties from different sources. Two properties from different sources being annotated with the same shape denotes a match.

Matching properties is far from trivial, especially with many sources. As an example, Fig. 1 shows camera entities (from a real dataset used in our evaluation) from four sources that may be integrated into a product KG together with some property matches indicated by symbols of the same shape. The example shows that there are numerous similar but differently named properties with diverse instance values. Matching properties often have completely different names, e.g., for properties "camera resolution", "effective pixels" and "megapixel". A property in one source, e.g. "shutter speed", may also have several matches in another source, e.g., "min shutter speed" and "max shutter speed". The instance values also show a high degree of heterogeneity due to the use of synonyms, abbreviations, different technical units, or numeric values, making it difficult to find matches with standard techniques that rely on string similarity metrics applied to either property names or instances. Even if more sophisticated techniques are used (e.g. word embeddings), the computation of similarities is usually unsupervised, making it hard to set thresholds that consistently achieve a high similarity for related properties and a low one for unrelated ones. The number of properties per entity may also differ to a large degree between sources and even within a source, which may affect some techniques.

To help solve the property matching problem in the case of such scenarios, we present a new approach called LEAPME (LEArning-based Property Matching with Embeddings). It uses supervised machine learning and makes use of the typically good availability of instance in a KG. LEAPME applies a dense neural network and a large set of features to classify a pair of properties from different sources as related or not. The proposed features make heavy use of word embeddings computed from both the property names and their instance values. Word embeddings are numeric vectors associated to single words, created so that they preserve their semantics. The use of embeddings gives the classifier information about the semantic proximity between two properties even when their string similarity is low. For example, we expect different words related to camera resolution such as "MP", "resolution" or "megapixels" to have similar embedding vectors. The use of property values provides additional information that is not tied to the name of a property, and makes the proposal applicable to scenarios in which the properties do not have meaningful names, e.g., identifiers that are automatically generated by information extraction approaches [12]. The use of machine learning helps use these features in a smart way, learning what features are more important and how they must be combined, which is of great relevance when it comes to word embeddings, since they can have a high number of components that would make setting manual weights and similarity thresholds very difficult.

Specifically, we make the following contributions:

- We propose LEAPME, a new learning-based approach for property matching that is applicable for data integration in scenarios with many sources that result in a high degree of heterogeneity, e.g., as needed for KG creation and refinement. We propose the use of numerous features derived from both the property names and property values and the heavy use of word embeddings for high match quality. These features are exploited by a supervised classifier to avoid setting manual weights and similarity thresholds for such features.

- We comprehensively evaluate LEAPME on four real-world datasets with entities from several e-commerce contexts. The multi-source setting makes it reasonable to use some

sources as training data in order to match the rest. We also provide a comparison with five baselines and show that LEAPME clearly outperforms previous approaches even with little training data.

- We show that LEAPME can also achieve better results than the baselines when trained with data obtained from entities of a different type or domain. The reduced need for domain-specific training data increases the applicability and impact of the new approach, which shows that the application of supervised machine learning and the development of labelled property matching datasets including varied domains is crucial, since they allow the creation of context-independent universal classifiers.

The next section describes related work on schema matching and the previous use of machine learning for this task. Section 3 formally describes the problem of property matching. Section 4 describes LEAPME in detail. Section 5 contains the evaluation including the comparison with several baselines and the use of transfer learning. Section 6 summarizes our contributions and discusses potential future work.

2. Related work

In the last decades, a huge amount of research has been devoted to schema and ontology matching to automatically determine corresponding schema attributes (properties) and ontology concepts. As described in several survey articles and books [13, 14, 15, 16, 17], most of the proposed approaches focus on pairwise matching between two schemas or ontologies and utilize a combination of several similarity values to determine likely matches. The most common approach is to determine the linguistic similarity of properties either based on string similarity metrics, synonym information from background knowledge resources such as dictionaries (e.g., WordNet [18]), or, more recently, pre-trained word embeddings [19, 20]. Background knowledge resources can even include a corpus of formerly matched schemas as support for a new match [21, 22]. Some approaches additionally utilize the structural similarity of elements (e.g., based on the similarity of neighbors in an ontology) and the similarity of associated instance data [23, 24].

Taking these considerations into account, we have used four of the existing pairwise tools as unsupervised baselines in our comparative evaluation according to their reported performance or similarity to our proposal. Two of them, Agreementmaker Light (AML) [25] and FCA-Map [26] because of their good results in the OAEI (ontology alignment evaluation initiative). The proposal by Duan et al. [24] because of its use of property instances, and SemProp [19] because of its use of word embeddings. They do not use supervised machine learning to learn optimized similarity thresholds but require the user to fine-tune parameters manually or with the help of some technique [27]. In particular, AML compares property names by doing a full-name match and computing word similarity, string similarity, and WordNet similarity. If any of the matchers returns a similarity above a user-given threshold (0.6 by default), the pair is considered a match. FCA-Map applies lexical matching to properties based on exact token co-occurrence. The technique by Duan et al. uses local sensitive hashing to estimate the similarity between two groups of instances. SemProp uses word embeddings to identify when two concatenated property names have semantic coherence.

The use of supervised machine learning is being increasingly applied for a simplified configuration of schema and ontology matching, since it can be considered a way to aggregate several similarity metrics or matchers, removing the need to set manual thresholds or use vector distance metrics such as the cosine similarity, which give the same weight to all features [28, 29, 30, 31, 32, 33, 30, 34, 35]. The training data consists of the similarity of matching and non-matching pairs of schema/ontology elements together with multiple similarity values, e.g., according to different linguistic and structural similarities. Surprisingly, instance similarities have not been utilized so far in these approaches. As a representative baseline we consider the approach of Nezhadi et al. [29] in our evaluation. It uses 12 name string similarity metrics as well as metrics derived from background knowledge by computing the distance of two concepts in the WordNet graph, and structural metrics based on the propagation of name similarities. They also considered 5 classification alternatives to determine matches and found out that an AdaBoost aggregation of Decision Tree classifiers achieves the best results.

The main limitation of supervised machine learning techniques is the need for training data. There are two main ways to deal with this requirement: manual provision of training data or the use of transfer learning. Manually labelling selected pairs of properties or concepts is of course laborious and does not scale well. This approach thus has to be limited to relatively small amounts of training data. With transfer learning the goal is to obtain the training from another domain or use case to avoid the provision of specific training [32]. This is especially valuable for scenarios such as knowledge graphs, in which there are typically already integrated entities and properties from different sources so that matching information can likely be reused. In our evaluation, we will consider both approaches: the use of manually defined training matches as well as transfer learning.

Most previous work focuses on pairwise schema and ontology matching for two sources [22] while we have to deal with an arbitrary number of sources with different sets of properties per entity type. While multi-source property matching also builds on pairwise property matching, the degree of heterogeneity and thus the difficulty to achieve good match quality increases with more sources. In our approach, we will determine pairwise similarities between properties that can be maintained in a similarity graph of properties from several sources. Such a graph can be used as input for clustering so that all matching properties are in the same cluster that can be used as a basis to fuse these properties. Property clustering is beyond the scope of this paper but can be done with similar algorithms to those used for clustering entities based on a similarity graph, e.g., [36, 37]. Other similar approaches have been proposed to refine the initial matching [38, 39].

3. Problem definition

We first provide some preliminary definitions in Section 3.1, then we describe the problem we focus on in a formal way with well-defined input and output in Section 3.2.

3.1. Preliminaries

Source: A source S is a location from where information comes, e.g., a website, a relational database, or a SPARQL endpoint, among other examples. It typically conforms to some kind of ontology and may contain structured entities of several types or classes.

The example in Figure 1 contains camera entities from four different sources, namely e-commerce websites such as "Mypriceindia.com" and "Shopmania.in".

Entity and class: An entity e is a representation of something that can be uniquely identified, usually corresponding to some real world object. Entities belong to a certain source and a source-specific type or class C, and we denote the source and class of entity e with $S(e)$ and $C(e)$, respectively. The rectangles in Figure 1 correspond to different entities of type "camera": "Fujifilm Finepix Z20", "Nikon D3300", "Kodak DC220", and "Pentax K-5 II". Entities consist of several properties and their values.

Property and instances: A property is an attribute to describe information about entities. The values of a property are literals known as instances. Our algorithm processes a collection of property instances represented as tuples (p, e, v) where p is the property name, e is the entity (identifier), and v is the property value. An example instance is ("camera resolution", "Fujifilm Finepix Z20","12 MP"). The components of a property instance i=(p, e, v) are denoted by $p(i)$, $e(i)$, and $v(i)$. Each such tuple is implicitly tied to the originating class $C(e(i))$ and source $S(e(i))$.

Class schema: For the sake of flexibility in applications such as E-commerce, we do not assume the existence of a predefined schema with a fixed set of properties per class. Rather, we view the schema of class C as the collection of all differently named properties for entities of class C in the respective sources. Individual entities may use any subset of these class properties.

Property matching: Task of determining correspondences between the properties of different class schemas from different sources.

3.2. Definition

We address property matching for properties of the same class (e.g., camera properties). We consider the case of multi-source matching so that properties may relate to entities from an arbitrary number of sources. Correspondences are not limited to equivalence relationships but also to more complex relationships between semantically related properties. A property in one source may thus have 0, 1 or several matching properties in another source, e.g., as for property "shutter speed" in Figure 1.

Therefore, the problem is as follows: Given a collection or property instances I corresponding to properties from m sources with $m > 1$, we define property matching as a binary classification problem where every pair of properties (p_i, p_j) from two different sources is classified as related or unrelated. Alternatively, every pair of properties can be assigned a similarity score sim indicating the strength of the relatedness. To enable the application of supervised machine learning techniques, we also assume the provision of training data consisting of pairs of properties from different sources labelled as either matching or non-matching.

The output can be represented as a similarity graph between properties of different sources. Such a graph can be used for determining clusters of matching properties, e.g., using clustering algorithms like transitive closure or more complex approaches as in [36, 37]. A simple transitive closure would group all same-shaped properties in Figure 1 within a

cluster. This would be sub-optimal if we want to ensure that only equivalent properties are grouped together, e.g., as useful for a fusion of property values within a KG. In such a case the properties "min shutter speed" and "max shutter speed" should be in separate clusters. This can be achieved with clustering techniques like in [37] that do not permit more than one cluster member from the same source. An alternative approach is to post-process the match correspondences, adopting a similar approach as in [40] that can determine the semantic type of correspondences (such as equality and part-of) and only continue with equality correspondences. The analysis of such post-processing options is beyond the scope of this paper and left for future work.

4. Our approach

Having defined the problem of property matching, we give an overview of our proposal LEAPME (Section 4.1), and describe in detail how features are computed (Section 4.2). We discuss the use of embeddings in Section 4.3. Finally, we describe aspects related to the implementation of LEAPME in Section 4.4.

4.1. Overview

LEAPME is a supervised ML-based property matching approach that focuses on the use of novel features. It computes features from property instances, property names, and property pairs to obtain large feature vectors that can be properly handled by a classifier. For example, from the instance value "12 MP" we can compute features such as the number of digits (2), the number of white spaces (1), or the fraction of letters (0.4). LEAPME thus uses such characteristics *about* instance values (and property names) in addition to their actual values.

Algorithm 1 describes the main steps of LEAPME; the workflow is also illustrated in Figure 2.

1. First, there is the initialization of the instance feature vector IF, the property feature vector PF, the property pair feature vector PPF, and the output similarity graph (collection of matches) Sim (line 1 of Algorithm 1).

2. Next, the instance features are determined by every instance with the help of function $iFeatures$, and added to the respective property in the instance feature vector IF (lines 2-3 of Algorithm 1, step 1 in Fig. 2). The features we determine will be described below - they include meta-features about the instance values as well as an embedding vector for the specific property value.

3. In lines 4-6 of Algorithm 1 we compute property features with the help of function $pFeatures$ (steps 2 and 3 in Fig. 2). They can be derived for the property name or based on the aggregation of instance features, e.g., average values of numeric instance features.

4. For each property pair, we compute the property pairs features using function $ppFeatures$ (lines 7-9 of Algorithm 1, step 4 in Fig. 2), which may be partially based on the aggregation of property features.

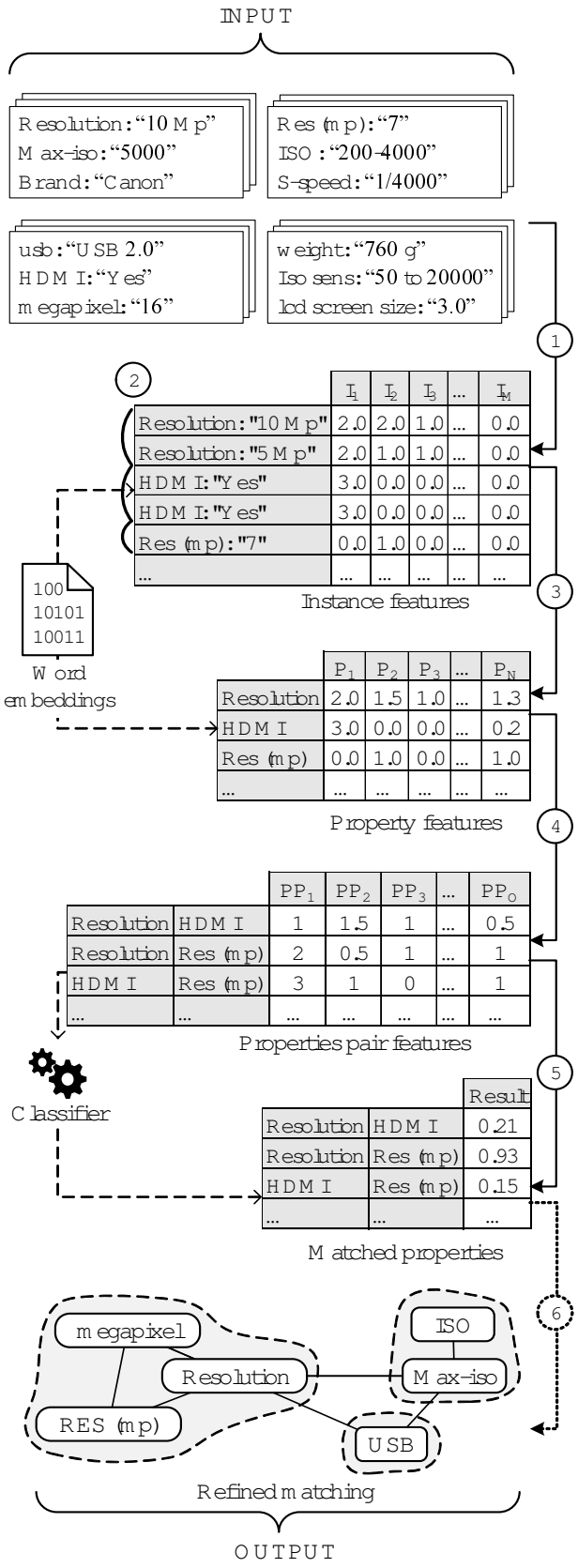


Figure 2: Workflow of LEAPME.

5. We use the input training data with their labeled property pairs and associated feature vectors to train a classification model using function $trainClassifier$ in line 10 ($labeled(PPF)$ denotes the already labeled property pairs). Then, we apply the trained classifier to the unlabeled property pairs to obtain a match decision and similarity score for each pair (lines 11-12 of Algorithm 1, step 5 in Fig. 2).

As shown in the last step of Fig. 2, the output represents a similarity graph that can be post-processed as discussed above.

Algorithm 1: LEAPME

Input:
- I: set of property instances from m sources
- labeled property pairs (training)

Output:
- Sim: set of property pairs with similarities (similarity graph)

Variables:
- IF: Map<$Property, FeaturesVectorSet$> with instance features vectors, grouped by property
- PF: Map<$Property, FeaturesVector$> with property features vectors.
- PPF: Map<$PropertyPair, FeaturesVector$> with property pair features vectors.
- m: classification model

```
1  initialize(IF, PF, PPF, Sim)
   // Steps 1-4: compute features
2  for i in I do
3      IF[p(i)] ← IF[p(i)] ∪ iFeatures(i))
4  for (p, V) in IF do
5      PF[p] ← pFeatures(p)
6  for p1 in keyset of PF do
7      for p2 from different source in keyset of PF do
8          PPF[(p1, p2)] ← ppFeatures(p1, p2)
   // Step 5: training and classification
9  m ← trainClassifier(labeled(PPF))
10 for (p1, p2) : v in unlabeled(PPF) do
11     Sim.add((p1, p2, m.classify(v)))
```

4.2. Features

Since we classify pairs of properties, the features that are ultimately fed to the classifier must be associated to a pair of properties. However, as we have mentioned, LEAPME considers features at several levels that can be later transformed into property pairs features. Next, we describe in detail each of these levels:

Instance features: These features are computed from each individual instance of a property (that is, a features vector is obtained for each property value) independently of

the property names. They provide information about the format of property values and can thus be considered as meta-features. We expect matching properties to follow similar formats, which should be reflected in these features. For example, while in Figure 2 properties "Res(mp)" and "megapixel" have a different name, both have short values with numeric characters, which could be reflected in features that measure the number of such characters or token types. While on their own these features may not be enough to properly match features (since, for example, many properties follow similar numeric formats), they could help disambiguate problematic cases. Furthermore, in some contexts the name of the properties may be unknown or only a generic identifier. For example, information extraction techniques may identify a piece of text as an instance, but not be able to infer a label with its property name [12]. In these cases, no features can be computed from the property names, and only these instance features enable matching. In addition to format-oriented meta-features we also consider the actual property values in the form of word embeddings or the numeric value (see below).

Property features: These features are computed for each individual property. They include all features computed from the property name, such as the average embeddings vector of its words. Furthermore, by grouping the instance features on a per-property basis, we can aggregate them and turn them into property features. For example, we could compute the average of each instance feature for a given property to represent the overall format followed by its instances.

Property pair features: These features are computed for each pair of properties to be classified. These are the final features actually fed to the classifier. Traditional string similarity metrics such as the Levenshtein or Jaro-Winkler distance would be part of these features, since they are computed from a pair or property names. Aggregated property features can also be used to determine property pair features. In this case, only two vectors are aggregated, e.g. by computing the numeric difference or average between the vectors, or by determining their concatenation.

Note that while only property pair features are relevant to the classification of property pairs, the other features are also used but are necessarily transformed into property pair features. For example, since a property can have hundreds of instances, there is a need to aggregate the hundreds of sets of instance features.

4.3. Embeddings and classification

When matching properties, a high value of the string similarity of the property names is usually a clear indicator of a match. Low similarity, however, can be caused by the issues we mentioned in Section 1. As discussed in Section 2, the usual way to mitigate this problem is to use external knowledge bases like WordNet to determine synonyms or name-independent similarities. These resources, however, are language-dependant and often of limited coverage. Furthermore, their use is relatively complex, and may require the use of APIs to handle the data.

As a more promising and versatile approach to overcome these limitations, we propose the use of word embeddings for both property names and property values. They can provide

rich information about the semantics of a property that can help solve some issues such as the potentially low string similarity between synonymous properties. While embeddings are language-dependant, versions for different languages can easily be trained from any large text corpus, unlike knowledge bases such as WordNet, whose creation requires a large manual effort. Furthermore, embeddings can be trained with a context-specific corpus, and are more likely to contain certain concepts. For example, the GloVe embeddings we use contains an entry for the word "28mm", which is a typical aperture value for cameras.

Embeddings vectors usually have hundreds of components with unknown meanings that may require nonlinear combinations to properly exploit their predictive power. For that reason, LEAPME uses a neural network for classification, which is also a popular choice in the related work and is able to properly weight features even when there is a large amount of them. While word embeddings have already been used in the past as discussed in Section 2, they have been exploited in an unsupervised way. Unsupervised techniques that use embeddings are forced to compute the distance (usually the cosine distance) between several embedding vectors, giving the same importance to all components, which may be detrimental when the number of components is high.

4.4. Implementation

Table 1 provides an overview about the features we have implemented. Instance features are computed with TAPON [41, 42], which includes several format-related features to which we added the embedding ones.

The rationale behind these features is the following: Features 1 and 2 contain information about the textual format of the instances, including absolute and relative frequencies of both character and token types. We expect similar properties to follow similar formats (for example, properties related to the ISO sensibility of a camera will usually contain at least 3 numeric characters). Feature 3 provides information about the specific value of purely numeric properties in order to disambiguate them according to the distribution of their values. Feature 4 provides information about the semantics of every instance. Feature 5 aggregates the instance features in order to transform them into property features by computing their average, which gives an overall idea of the format and the instance semantics of a property. Feature 6 provides information about the semantics of a property from its name. Feature 7 aggregates the property features of two properties by computing the difference of each feature in order to obtain information about the distance with regards to every feature. Features 8 to 15 use traditional string distance metrics applied to the names of the properties, since properties with similar names are usually related.

To compute embeddings, we use the pre-trained GloVe approach [43][1], specifically for the uncased Common Crawl corpus that includes 300-dimensional vectors for 1.9 million words, promising a good coverage for different domains, since the corpus should contain a great variety of tokens. Unknown words are mapped to a vector filled with zeroes. Each word in a property value or property name can thus be mapped to a point in the 300-dimensional data space so that similar words will have a small distance in it. For each property value and name we determine the average embeddings of the individual words represented by 300

[1] `https://nlp.stanford.edu/projects/glove/`

Type	Id	Description	# of features
Instance	1	The fraction and number of occurrences of several character types (letters (uppercase, lowercase, and both), mark characters, numbers, punctuation, symbols, separators, other)	18
	2	The fraction and number of occurrences of several token types (words, words starting with a lowercase letter, words starting with an uppercase letter followed by a non-separator character, uppercase words, numeric strings)	10
	3	The numeric value of the instance (-1 if it is not a number)	1
	4	The average embeddings vector of the words in the instance	300
Property	5	The average of every instance feature	329
	6	The average embeddings vector of the words in the property name	300
Properties pair	7	The difference between the features vectors of the two properties	629
	8	The optimal string alignment distance between the property names	1
	9	The Levenshtein distance between the property names	1
	10	The Full Damerau-Levenshtein distance between the property names	1
	11	The longest common substring distance between the property names	1
	12	The 3-gram distance between the property names	1
	13	The cosine distance between the 3-gram profiles of the property names	1
	14	The Jaccard distance between the 3-gram profiles of the property names	1
	15	The Jaro-Winker distance between the property names	1

Table 1: Features used in our implementation.

Prop.1	Prop.2	Label	Levenshtein	Fraction of letters	Fraction of numbers	Inst.emb. 3/300	Inst.emb. 4/300	Name emb. 65/300	Name emb. 66/300	...
camera resolution	megapixel	POS	13.00	0.00	0.10	0.13	0.07	0.70	0.04	...
camera resolution	effective pixels	POS	15.00	0.66	0.26	0.06	0.10	0.46	0.00	...
camera resolution	dynamic af mode	NEG	16.00	0.54	0.31	0.18	0.33	0.31	0.59	...
camera resolution	alarm trigger	NEG	14.00	0.81	0.42	0.23	0.41	0.38	0.69	...

Table 2: Sample feature values for matching and non-matching property pairs.

values that serve as features for our classification approach. We can deal with such relatively large feature vectors since the use of supervised machine learning with a neural network should be able to identify the most important ones features and give them an appropriate weight.

As indicated in Table 1, we currently use 28 meta-features for instance values. The actual instance values are reflected in one feature for numeric values and 300 features of the embeddings vector. The averages for these 329 features over all instance values of a property

serve as property features. They are complemented with 300 features of the embeddings vector for the property name. The final feature vectors for property pairs thus include 629 features regarding the difference between the property features. Together with 8 features about the string similarity of property names there are 637 features for property pairs in total.

Table 2 shows an example for a small subset of our features computed for four labeled property pairs (two matching and two non-matching). Note that the Levenshtein distances are similarly high in both positive and negative match cases while other features can better discriminate between them. For example, for feature "fraction of letters" the difference is 0 between properties "camera resolution" and "megapixel" and can thus help to determine such a match. Some of the embedding features (corresponding to individual components of the embeddings vectors) have good discriminating potential, with a low difference in positive and high difference in negative match cases (e.g., instance embedding 4 and name embedding 66).

Finally, regarding the architecture of the neural network behind LEAPME, it consists of two fully connected hidden layers of sizes 128 and 64. We use a batch size of 32 and perform 10 epochs with learning rate 10^{-3}, 5 with 10^{-4}, and 5 with 10^{-5}. We fine-tuned these hyper-parameters manually in preliminary tests, though most alterations (such as changing the size of the layers) do not significantly impact on the results. The final layer has two neurons from which the final score is obtained for the two possible outcomes (positive/negative). This allows the use of the positive output as a similarity score, which is useful for post-processing steps such as property clustering.

5. evaluation

We experimentally evaluate our property matching approach LEAPME on four real-word datasets with up to 24 sources. We analyze the impact of different amounts of training data and the effectiveness of the different kinds of features; in particular, the use of embeddings for both property values and property names. We further compare LEAPME with five baselines and study the use of transfer learning. The focus is on match quality with the standard metrics precision, recall and F-measure (F1 score).

We first give some details about the studied feature configurations for LEAPME and the baseline approaches. Next, we describe the four datasets and the two use cases with training data from either the same or a different domain. The results for the two use cases are discussed in subsections 5.4 and 5.5, respectively. The evaluated implementations along with the detailed results and additional material are available online[2].

5.1. Feature configurations and baselines

The rich set of features exploited by supervised learning is a main advantage of LEAPME and we therefore analyze the effectiveness of the different kinds of features in detail. Along one dimension, we compare the use of instance-related features only, name-related features only and the combined use of both kinds of features. Another dimension is the consideration

[2] `https://github.eii.us.es/dayala1/LEAPME`

of embedding-based features only, non-embedding features only or the combined use of both kinds of features. In total, this sums up to 9 possible feature configurations to analyze.

The LEAPME results are compared to the results obtained by the following baselines:

- The latest Github implementation of Agreement Maker Light [25] (AML), the highest-ranked technique in the M2 variants of the "Conferences" track of OAEI 2019, which involve the matching of only properties [3].

- The latest Github implementation of FCA-Map [26], the best-performing property matching technique in the "Knowledge Graph" track of OAEI 2019 [4].

- An implementation of the machine learning proposal by Nezhadi et al. [29]. It was selected among machine learning proposals for having the largest features catalogue, from which we removed the structural features since they were not applicable to our evaluation datasets. We use AdaBoost with decision trees as classifier, which achieved the best results in [29].

- An implementation of SemProp [19], selected as a representative of existing proposals that use word embeddings. We used the proposed matchers graph by removing the use StructS, which is not applicable since we only match properties, where there are no class hierarchies involved. We tested all combinations of values $0.2, 0.4, 0.6, 0.8$ for the thresholds used by the SynM, SeMa(-), and SeMa(+) matchers. For our final experiments we used the combination that yielded the highest average F1 score across our datasets: 0.2 for SynM, 0.2 for SeMa(-), and 0.4 for SeMa(+).

- An implementation of the proposal by Duan et al. [24] based on local-sensitive hashing (LSH), selected as a representative of existing proposals that use property instances for matching. We tested both variants (random projections and minhash) with the proposed number of hash functions (1000 and 500 respectively) and the following band sizes: all integers from 1 to 10, and integers from 10 to 50 in steps of 5. For our final experiments, we used the combination that yielded the highest average F1 score across our datasets: minhash with a band size of 1.

5.2. Datasets

For our evaluation, we use four real-word datasets with different kinds of e-commerce products (cameras, headphones, phones, and TV sets) from multiple sources.

All datasets align the properties in each source to a reference ontology. We consider that two properties are related (matching) when they are both aligned to the same reference property. All four datasets have been extracted from the Web using information extraction techniques, and contain noise that is typical of real world scenarios, making matching more challenging.

Table 3 provides main statistics for the four datasets. The camera dataset comes from the DI2KG19 challenge [44]. It is the largest dataset with 24 sources, more than 3200 properties

[3] `http://oaei.ontologymatching.org/2019/results/conference/index.html`

[4] `http://oaei.ontologymatching.org/2019/results/knowledgegraph/index.html`

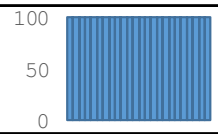
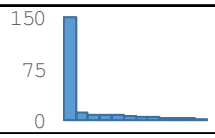
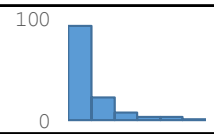
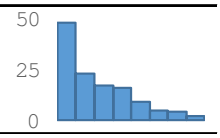
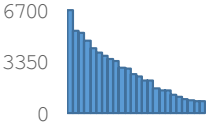
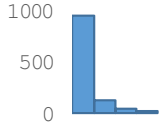
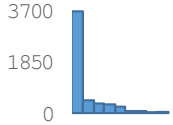
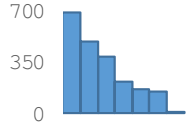
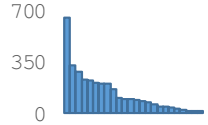
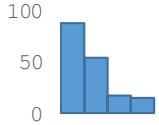
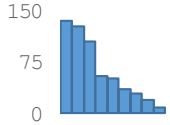
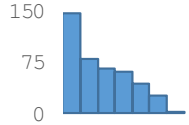
	Cameras	Headphones	Phones	TVs
# of Sources	24	6	12	8
Entities	2400	128	208	124
Properties	3245	172	554	415
Instances	65615	1129	5195	2069
Positives	9199	412	2677	1062
Entities per source				
Instances per source				
Properties per source				

Table 3: Datasets metadata. Each vertical bar in a plot represents a source within a dataset.

and about 9200 matching property pairs. We limited the number of entities to 100 per source in order to balance their size and impact. But, as shown in the lower part of Table 3, the number of different properties and the number of property entities differs substantially between different sources, with almost 700 properties for one of the sources (EBay.com). The other datasets contain headphones, phones and TV product entities and correspond to the WDC Gold Standard for Product Matching and Product Feature Extraction [45]. These are much smaller than the camera dataset and there are different numbers of entities per source leading to a less balanced setting than that of the camera dataset. In our analysis of the results, we will refer to the three smaller and imbalanced datasets as low-quality datasets as opposed to the high-quality camera dataset.

5.3. Use cases and training data

For training, we differentiate two use cases in which the training data either refers to properties from the same domain (entity type) or to properties from a different domain. For the first use case, which we call Single Domain (SD), we take a fraction of the sources of a dataset (at random) for training. We use the examples that involve two sources of data in the training set to train the classifier, and test it with the rest. We performed experiments using different training fractions: 0.2, 0.4, 0.6, and 0.8. For each of these fractions and for each dataset, we ran LEAPME 25 times, using different random combinations of training sources.

For the second use case, called Transfer Learning (TL), we train the classifier for one dataset (entity type) with training data from the other datasets.

For this purpose, we tested all possible combinations of using 1, 2, or 3 datasets for training.

Figure 3 shows an example of how data is divided into training and testing for both use cases. The example illustrates SD training for the camera dataset where each labeled

property pair uses properties from two of the 24 sources. For the TL use case, there are many possible configurations. The shown example refers to training pairs from two datasets to be used for evaluating property matching for the rest.

For all datasets and use cases, the training data consists of two negative (non-matching) pairs of properties for every positive (matching) pair, and the negative pairs are randomly selected.

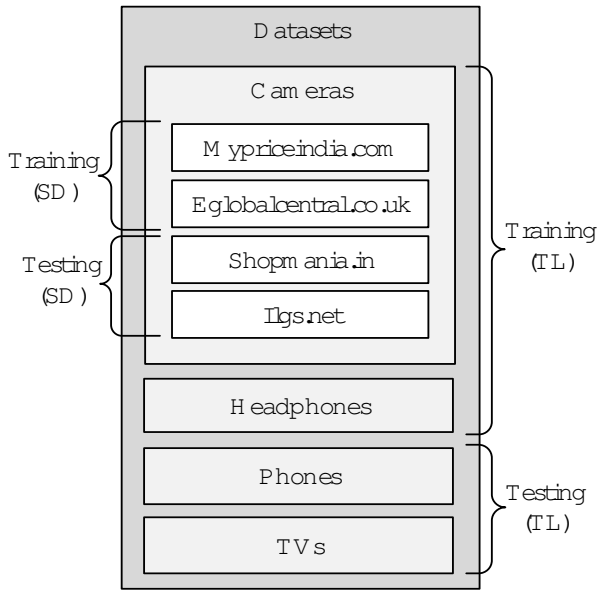


Figure 3: Use cases testing example.

5.4. Single-domain results

Next, we compare the results obtained by different configurations of LEAPME, as well as those obtained by the five baselines.

We first evaluate the results in the single-domain case and compare LEAPME with the five baselines. Figure 4 shows the precision (in blue) and recall (in red) results for the four datasets for different amounts of training data (0.2 to 0.8) for the supervised approaches LEAPME and Nezhadi. The results for the unsupervised approaches are shown as horizontal lines since they do not depend on training data. The results for LEAPME are obtained by using both features for property names and instances but we differentiate the three cases of the use of embedding features only, the use of non-embedding features only and the use of all features. The value ranges for the supervised approaches reflect the different training configurations involving a random selection of the sources in a dataset. The variations are generally higher for the smaller and unbalanced datasets like headphones that provide fewer training data. We make the following observations:

- Unsupervised techniques can achieve a high precision but struggle to reach a similar recall.
- LEAPME achieves better overall results than all the baselines, with a dramatic increase of recall when compared to AML and FCA-Map and both recall and precision improvements when compared to the rest.

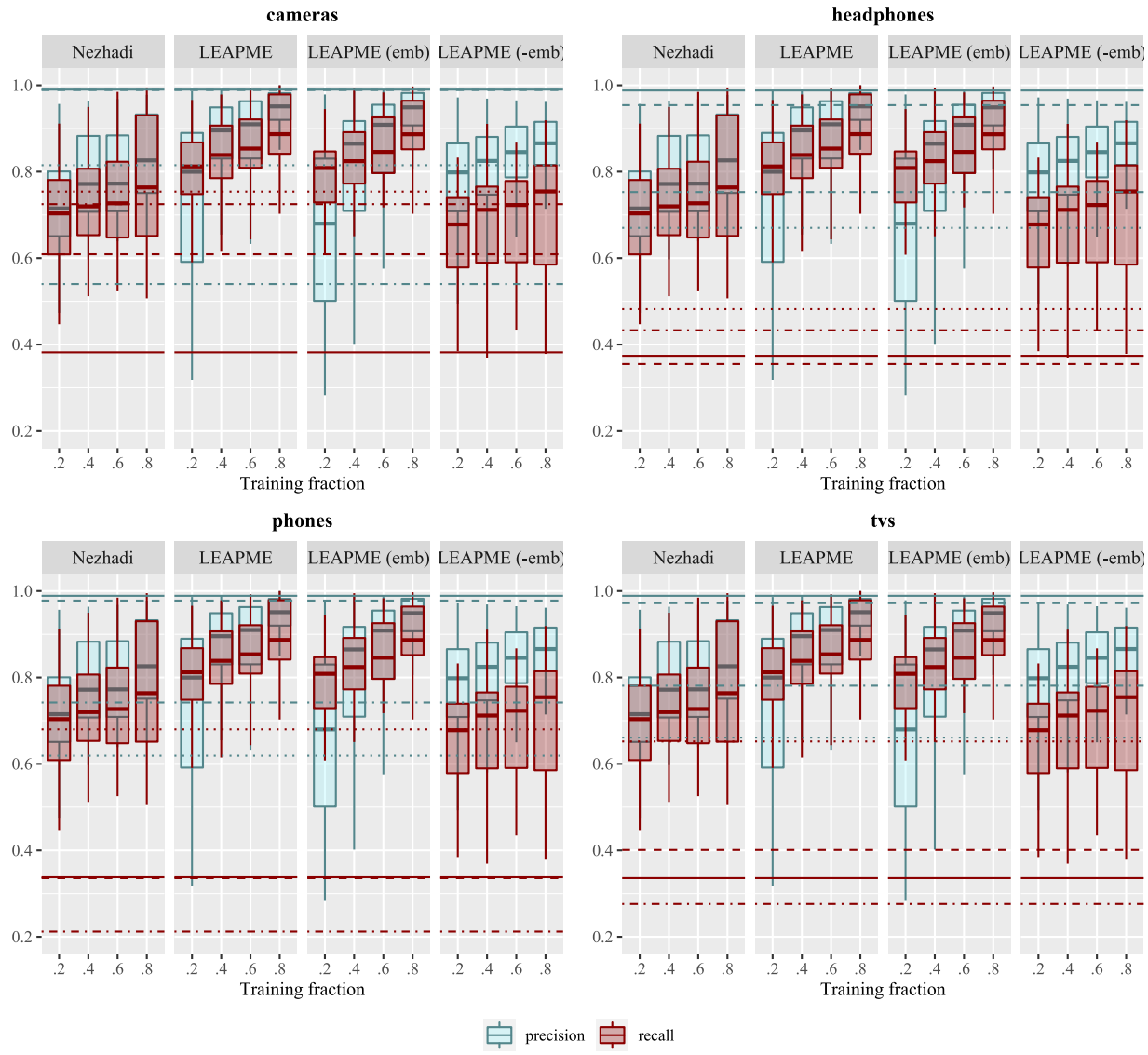


Figure 4: SD use case results. Matching with both property names and values. LEAPME(emb) = LEAPME with embedding features only. LEAPME(-emb) = LEAPME without embedding features. Dashed line = AML. Solid line = FCA-Map. Dotted line = SemProp. DotDash line = LSH.

Info	Dataset	Train. %	LEAPME P	LEAPME R	LEAPME F1	LEAPME (emb) P	LEAPME (emb) R	LEAPME (emb) F1	LEAPME (-emb) P	LEAPME (-emb) R	LEAPME (-emb) F1	Nezhadi P	Nezhadi R	Nezhadi F1	AML P	AML R	AML F1	FCA-Map P	FCA-Map R	FCA-Map F1	SemProp P	SemProp R	SemProp F1	LSH P	LSH R	LSH F1
Instances	cameras	20%	0.66	0.55	0.59	0.72	0.52	0.58	0.55	0.43	0.44	-	-	-	-	-	-	-	-	-	-	-	-	0.54	0.73	0.62
		80%	0.93	0.75	0.83	0.91	0.77	0.83	0.64	0.59	0.61	–	–	–	–	–	–	–	–	–	–	–	–			
	headphones	20%	0.54	0.61	0.56	0.61	0.64	0.60	0.54	0.57	0.54	-	-	-	-	-	-	-	-	-	-	-	-	0.75	0.43	0.55
		80%	0.76	0.70	0.69	0.64	0.70	0.64	0.60	0.51	0.53	–	–	–	–	–	–	–	–	–	–	–	–			
	phones	20%	0.60	0.59	0.58	0.58	0.63	0.59	0.47	0.41	0.42	–	–	–	–	–	–	–	–	–	–	–	–	0.74	0.21	0.33
		80%	0.84	0.75	0.79	0.85	0.74	0.79	0.59	0.44	0.50	–	–	–	–	–	–	–	–	–	–	–	–			
	tvs	20%	0.61	0.62	0.60	0.61	0.62	0.60	0.49	0.57	0.52	–	–	–	–	–	–	–	–	–	–	–	–	0.78	0.28	0.41
		80%	0.83	0.74	0.78	0.84	0.73	0.78	0.65	0.60	0.61	–	–	–	–	–	–	–	–	–	–	–	–			
Names	cameras	20%	0.89	0.88	0.88	0.87	0.86	0.86	0.91	0.75	0.82	0.86	0.82	0.83	0.99	0.61	0.75	0.99	0.38	0.55	0.82	0.75	0.78	-	-	-
		80%	0.99	0.98	0.98	0.98	0.98	0.98	0.95	0.76	0.84	0.96	0.93	0.94										–	–	–
	headphones	20%	0.68	0.79	0.73	0.67	0.81	0.72	0.82	0.62	0.70	0.73	0.69	0.70	0.95	0.36	0.52	0.99	0.37	0.54	0.67	0.48	0.56	-	-	-
		80%	0.84	0.82	0.82	0.83	0.81	0.81	0.91	0.58	0.70	0.80	0.72	0.75										–	–	–
	phones	20%	0.70	0.71	0.70	0.65	0.74	0.67	0.80	0.51	0.61	0.64	0.56	0.59	0.98	0.34	0.50	0.99	0.34	0.50	0.62	0.68	0.65	-	-	-
		80%	0.93	0.84	0.88	0.91	0.85	0.88	0.92	0.51	0.66	0.74	0.68	0.71										–	–	–
	tvs	20%	0.62	0.77	0.67	0.70	0.78	0.72	0.85	0.68	0.75	0.67	0.70	0.68	0.97	0.40	0.57	0.99	0.34	0.50	0.66	0.65	0.66	-	-	-
		80%	0.95	0.86	0.90	0.93	0.84	0.88	0.93	0.70	0.80	0.83	0.79	0.81										–	–	–
Both	cameras	20%	0.91	0.83	0.87	0.83	0.77	0.79	0.88	0.74	0.80	0.86	0.82	0.83	0.99	0.61	0.75	0.99	0.38	0.55	0.82	0.75	0.78	0.54	0.73	0.62
		80%	0.99	0.97	0.98	0.98	0.97	0.98	0.93	0.82	0.87	0.96	0.93	0.94												
	headphones	20%	0.74	0.81	0.76	0.65	0.80	0.70	0.79	0.68	0.73	0.73	0.69	0.70	0.95	0.36	0.52	0.99	0.37	0.54	0.67	0.48	0.56	0.75	0.43	0.55
		80%	0.89	0.87	0.88	0.88	0.90	0.89	0.80	0.68	0.72	0.80	0.72	0.75												
	phones	20%	0.71	0.72	0.70	0.59	0.70	0.63	0.66	0.52	0.56	0.64	0.56	0.59	0.98	0.34	0.50	0.99	0.34	0.50	0.62	0.68	0.65	0.74	0.21	0.33
		80%	0.93	0.85	0.89	0.92	0.86	0.89	0.83	0.57	0.68	0.74	0.68	0.71												
	tvs	20%	0.64	0.80	0.70	0.60	0.81	0.67	0.71	0.67	0.67	0.67	0.70	0.68	0.97	0.40	0.57	0.99	0.34	0.50	0.66	0.65	0.66	0.78	0.28	0.41
		80%	0.95	0.89	0.92	0.94	0.86	0.90	0.88	0.77	0.82	0.83	0.79	0.81												

Table 4: SD use case summary with F1 scores. LEAPME(emb) = LEAPME with embedding features only. LEAPME(-emb) = LEAPME without embedding features.

- The use of embedding features always improves recall compared to the sole use of non-embedding features, but they need enough training data to reach or surpass their precision. As expected, using all features achieves the best results for LEAPME.
- When only using embeddings, LEAPME achieves, with 20% of training data, significantly better results than SemProp, except in the cameras dataset, where results are similar. When using more training data, LEAPME achieves much better results, showing that the use of embeddings greatly benefits from supervised learning.

For a more detailed analysis, we summarize the average results, including F1 scores, in Table 4 for both 20% and 80% training data. The table also provides results for the sole use of instance features and the sole use of name features, again differentiated by the use of embedding features only, non-embedding features only or both. The best F1 results of each row have been marked in bold. We make the following additional observations.

- For all datasets, LEAPME achieves a better F1 score than all baseline approaches even when using only 20% training data. For 80% training data, it achieves excellent F1 scores from 88% (for headphones) to 98% (for cameras). In this case, the baselines are outperformed especially for the low-quality and more challenging datasets (headphones, phones, and TVs). The unsupervised baselines were outperformed by up to 42 F1 percentage points (50 vs 92% for the TV dataset) and the supervised baseline of Nezhadi by up to 18 percentage points (71 vs 89% for the phones dataset).
- When only using property names LEAPME without embedding features already outperforms the baselines. The embedding features for property names are the most

effective features in LEAPME. Their use alone is more more effective than the use of non-embedding features relying on string similarities.

- Only using instance features achieves weaker results for LEAPME than using name features especially with little training. Again, using embedding features is more effective than using the non-embedding ones that focus on format-oriented meta-features. Still, the combination of both instance and name features helps to achieve a slight improvement over the sole use of name features in most cases.

Table 4 includes a number of further interesting results such as that SemProp outperforms the other unsupervised baseline approaches due to its use of embeddings. For the TV dataset, the use of non-embedding features proved to be slightly more effective than the use of embeddings for 20% of data for training showing that the effectiveness of embeddings can depend on the availability of a sufficient amount of training.

5.5. Results for transfer learning

Figure 5 shows the precision and recall results of our experiments in the TL use case. Each point corresponds to a different combination of three or less datasets used for training, the rest being used for validation. Note that baseline results are the same when using both name and instances or only of them, since even when both are available, they only use one. The figure also distinguishes between the use of high-quality and low-quality datasets for training.

We observe that, similarly as in the SD use case, the unsupervised baselines can achieve high precision but suffer from a lower recall. LEAPME again achieves the best results for embedding features especially for names or both names and instances. The best results are generally achieved when the training includes data from the high quality dataset (camera dataset) that can also provide more training samples than the low-quality datasets. LEAPME achieves near-perfect precision and recall when using it for training, demonstrating that it can make excellent use of transfer learning for good training data from a different domain. By contrast, the supervised technique by Nezhadi et al. only achieves better precision for training from a high-quality dataset but worse recall making it less suitable for transfer learning. SemProp can achieve results similar to LEAPME, but only when low quality training data is used. This indicates that the use of embeddings in LEAPME in a supervised way is a key reason for its highly effective use of transfer learning.

6. Conclusions

We have presented LEAPME, a new powerful approach for matching properties from many sources. It is a machine learning approach that utilizes a large spectrum of features, in particular embedding features, on both property names and instance values. Our evaluation with four real-world multi-source datasets shows that LEAPME clearly outperforms several baseline approaches representing the current state-of-the art. The improvements are even achieved for relatively little training data. Moreover, we showed that the use of embeddings in LEAPME in a supervised way enables an effective use of transfer learning so that existing high-quality training data from different domains can be utilized to reduce the effort for providing labeled training data.

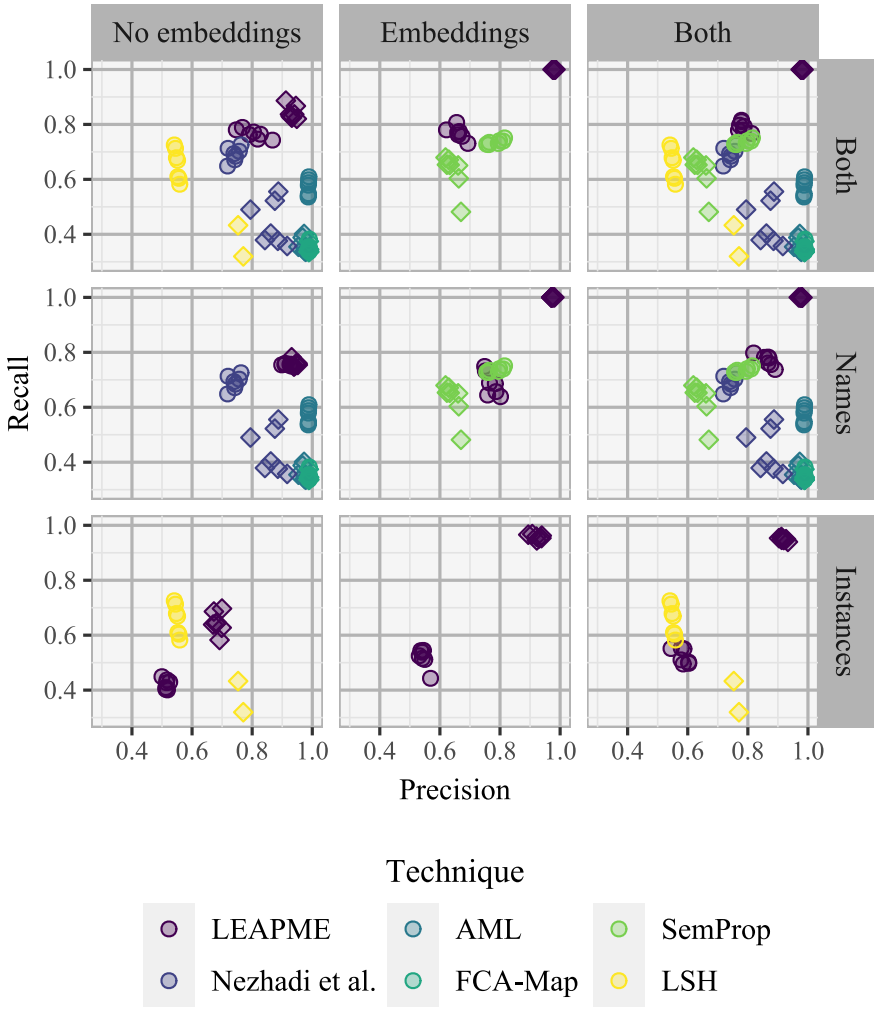


Figure 5: TL use case results. Rhombuses = high quality datasets in training split. Circles = high quality datasets in testing split.

In future work, we will investigate the use of LEAPME within a more comprehensive data integration approach for knowledge graphs that also includes entity matching and clustering as well as data fusion. In particular, we plan to evaluate different methods for deriving clusters of equivalent properties from the match results determined with LEAPME.

7. Acknowledgements

Our work was supported by the Spanish R&D&I programme with grants TIN2016-75394-R, PID2019-105471RB-I00, and P18-RT-1060.

References

[1] E. Rahm, The case for holistic data integration, in: Proc. ADBIS, Springer, 2016, pp. 11–27.

[2] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, W. Zhang, Knowledge vault: A web-scale approach to probabilistic knowledge fusion, in: Proc. 20th Int. conference on Knowledge discovery and data mining (KDD), 2014, pp. 601–610.

[3] P. Szekely, C. A. Knoblock, J. Slepicka, A. Philpot, A. Singh, C. Yin, D. Kapoor, P. Natarajan, D. Marcu, K. Knight, et al., Building and using a knowledge graph to combat human trafficking, in: Int. Semantic Web Conference (ISWC), Springer, 2015, pp. 205–221.

[4] L. Shi, S. Li, X. Yang, J. Qi, G. Pan, B. Zhou, Semantic health knowledge graph: Semantic integration of heterogeneous medical knowledge and services, BioMed research international 2017 (2017).

[5] K. Xu, L. Wang, M. Yu, Y. Feng, Y. Song, Z. Wang, D. Yu, Cross-lingual knowledge graph alignment via graph matching neural network, arXiv preprint arXiv:1905.11605 (2019).

[6] D. Ayala, A. Borrego, I. Hernández, C. R. Rivero, D. Ruiz, Aynec: All you need for evaluating completion techniques in knowledge graphs, in: Proc. European Semantic Web Conference (ESCW), Springer, 2019, pp. 397–411.

[7] A. Borrego, D. Ayala, I. Hernández, C. R. Rivero, D. Ruiz, Generating rules to filter candidate triples for their correctness checking by knowledge graph completion techniques, in: Proc. 10th Int. Conf. on Knowledge Capture, 2019, pp. 115–122.

[8] D. Obraczka, A. Saeedi, E. Rahm, Knowledge graph completion with FAMER, in: Proc. DI2KG, 2019.

[9] A. Saeedi, E. Peukert, E. Rahm, Incremental multi-source entity resolution for knowledge graph completion, in: Proc. ESWC (to appear), 2020.

[10] W. Zheng, J. X. Yu, L. Zou, H. Cheng, Question answering over knowledge graphs: question understanding via template decomposition, Proc. of the VLDB Endowment 11 (11) (2018) 1373–1386.

[11] X. Huang, J. Zhang, D. Li, P. Li, Knowledge graph embedding based question answering, in: Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, 2019, pp. 105–113.

[12] J. C. Roldán, P. Jiménez, R. Corchuelo, On extracting data from tables that are encoded using html, Knowledge-Based Systems (2019) 105157.

[13] E. Rahm, P. A. Bernstein, A survey of approaches to automatic schema matching, the VLDB Journal 10 (4) (2001) 334–350.

[14] J. Euzenat, P. Shvaiko, et al., Ontology matching, Vol. 18, Springer, 2007.

[15] Z. Bellahsene, A. Bonifati, E. Rahm, Schema Matching and Mapping, Springer, 2011.

[16] P. A. Bernstein, J. Madhavan, E. Rahm, Generic schema matching, ten years later, Proc. of the VLDB Endowment 4 (11) (2011) 695–701.

[17] L. Otero-Cerdeira, F. J. Rodríguez-Martínez, A. Gómez-Rodríguez, Ontology matching: A literature review, Expert Systems with Applications 42 (2) (2015) 949–971.

[18] F. Lin, K. Sandkuhl, A survey of exploiting Wordnet in ontology matching, in: IFIP Int. Conf. on Artificial Intelligence in Theory and Practice, Springer, 2008, pp. 341–350.

[19] R. C. Fernandez, E. Mansour, A. A. Qahtan, A. Elmagarmid, I. Ilyas, S. Madden, M. Ouzzani, M. Stonebraker, N. Tang, Seeping semantics: Linking datasets using word embeddings for data discovery, in: 2018 IEEE 34th International Conference on Data Engineering (ICDE), IEEE, 2018, pp. 989–1000.

[20] P. Kolyvakis, A. Kalousis, D. Kiritsis, Deepalignment: Unsupervised ontology matching with refined word vectors, in: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), 2018, pp. 787–798.

[21] J. Madhavan, P. A. Bernstein, A. Doan, A. Halevy, Corpus-based schema matching, in: 21st International Conference on Data Engineering (ICDE'05), IEEE, 2005, pp. 57–68.

[22] E. Rahm, Towards large-scale schema and ontology matching, in: Schema matching and mapping, Springer, 2011, pp. 3–27.

[23] H. Qin, D. Dou, P. LePendu, Discovering executable semantic mappings between ontologies, in: OTM Confederated Int. Conferences "On the Move to Meaningful Internet Systems", Springer, 2007, pp. 832–849.

[24] S. Duan, A. Fokoue, O. Hassanzadeh, A. Kementsietsidis, K. Srinivas, M. J. Ward, Instance-based matching of large ontologies using locality-sensitive hashing, in: International Semantic Web Conference, Springer, 2012, pp. 49–64.

[25] D. Faria, C. Pesquita, T. Tervo, F. M. Couto, I. F. Cruz, AML and AMLC Results for OAEI 2019 2536 (2019) 123–130.

[26] F. Chang, G. Chen, S. Zhang, FCAMap-KG results for OAEI 2019, in: CEUR Workshop Proceedings, Vol. 2536, RWTH, 2019, pp. 138–145.

[27] C. R. Rivero, D. Ruiz, Selecting suitable configurations for automated link discovery, in: Symp. On Applied Computing, 2020, pp. 907–915.

[28] D. Spohr, L. Hollink, P. Cimiano, A machine learning approach to multilingual and cross-lingual ontology matching, in: Int. Semantic Web Conference (ISWC), Springer, 2011, pp. 665–680.

[29] A. H. Nezhadi, B. Shadgar, A. Osareh, Ontology alignment using machine learning techniques, Int. Journal of Computer Science & Information Technology 3 (2) (2011) 139.

[30] K. M. Shenoy, K. Shet, U. D. Acharya, Nn-based ontology mapping, in: Int. Conf. on Advances in Information Technology and Mobile Communication, Springer, 2012, pp. 122–127.

[31] R. Ichise, Machine learning approach for ontology mapping using multiple concept similarity measures, in: Int. Conf. on Computer and Information Science (ICIS), IEEE, 2008, pp. 340–346.

[32] K. Eckert, C. Meilicke, H. Stuckenschmidt, Improving ontology matching using meta-level learning, in: European Semantic Web Conference (ESWC), Springer, 2009, pp. 158–172.

[33] W. E. Djeddi, M. T. Khadir, Ontology alignment using artificial neural network for large-scale ontologies, Int. Journal of Metadata, Semantics and Ontologies 8 (1) (2013) 75–92.

[34] C. Curino, G. Orsi, L. Tanca, X-som: A flexible ontology mapper, in: 18th Int. Workshop on Database and Expert Systems Applications (DEXA), IEEE, 2007, pp. 424–428.

[35] A. Marie, A. Gal, Boosting schema matchers, in: OTM Confederated International Conferences" On the Move to Meaningful Internet Systems", Springer, 2008, pp. 283–300.

[36] O. Hassanzadeh, F. Chiang, H. C. Lee, R. J. Miller, Framework for evaluating clustering algorithms in duplicate detection, Proc. of the VLDB Endowment 2 (1) (2009) 1282–1293.

[37] A. Saeedi, E. Peukert, E. Rahm, Using link features for entity clustering in knowledge graphs, in: European Semantic Web Conference (ESWC), Springer, 2018, pp. 576–592.

[38] I. Megdiche, O. Teste, C. Trojahn, An extensible linear approach for holistic ontology matching, in: Int. Semantic Web Conference (ISWC), Springer, 2016, pp. 393–410.

[39] P. Roussille, I. Megdiche Bousarsar, O. Teste, C. Trojahn, Holontology: results of the 2018 oaei evaluation campaign, CEUR-WS: Workshop proceedings, 2018.

[40] P. Arnold, E. Rahm, Enriching ontology mappings with semantic relations, Data & Knowledge Engineering 93 (2014) 1–18.

[41] D. Ayala, I. Hernández, D. Ruiz, M. Toro, Tapon: A two-phase machine learning approach for semantic labelling, Knowledge-Based Systems 163 (2019) 931–943.

[42] D. Ayala, I. Hernández, D. Ruiz, M. Toro, Tapon-mt: A versatile framework for semantic labelling, Information Systems 83 (2019) 57–68.

[43] J. Pennington, R. Socher, C. D. Manning, Glove: Global vectors for word representation, in: Empirical Methods in Natural Language Processing (EMNLP), 2014, pp. 1532–1543. URL http://www.aclweb.org/anthology/D14-1162

[44] 1st Int. Workshop on challenges, experiences from Data Integration to Knowledge Graphs, Di2kg challenge, http://di2kg.inf.uniroma3.it/, accessed: 2020-02-13.

[45] Web Data Commons, Gold standard for product matching and product feature extraction, http://webdatacommons.org/productcorpus/, accessed: 2020-02-13.