# Supervised Ontology and Instance Matching with MELT

Sven Hertling[1⋆][0000−0003−0333−5888], Jan Portisch[1,2⋆][0000−0001−5420−0663], and Heiko Paulheim[1][0000−0003−4386−8195]

[1] Data and Web Science Group, University of Mannheim, Germany
{jan, sven, heiko}@informatik.uni-mannheim.de
[2] SAP SE Product Engineering Financial Services, Walldorf, Germany
{jan.portisch}@sap.com

**Abstract.** In this paper, we present *MELT-ML*, a machine learning extension to the *Matching and EvaLuation Toolkit* (MELT) which facilitates the application of supervised learning for ontology and instance matching. Our contributions are twofold: We present an open source machine learning extension to the matching toolkit as well as two supervised learning use cases demonstrating the capabilities of the new extension.

**Keywords:** ontology matching · supervised learning · machine learning · knowledge graph embeddings

## 1 Introduction

Many similarity metrics and matching approaches have been proposed and developed up to date. They are typically implemented as engineered systems which apply a process-oriented matching pipeline. Manually combining metrics, also called *features* in the machine learning jargon, is typically very cumbersome. Supervised learning allows researchers and developers to focus on adding and defining features and to leave the weighting of those and the decision making to a machine. This approach may also be suitable for developing generic matching systems that self-adapt depending on specific datasets or domains. Here, it makes sense to test and evaluate multiple classifiers at once in a fair, i.e. reproducible, way. Furthermore, recent advances in machine learning – such as in the area of knowledge graph embeddings – may also be applicable for the ontology and instance matching community. The existing evaluation and development platforms, such as the *Alignment API* [3], *SEALS* [7,33] or the *HOBBIT* [25] framework, make the application of such advances not as simple as it could be.

⋆ The authors contributed equally to this paper.

In this paper, we present *MELT-ML*, an extension to the *Matching and EvaLuation Toolkit* (MELT). Our contribution is twofold: Firstly, we present a machine learning extension to the MELT framework (available in MELT 2.6) which simplifies the application of advanced machine learning algorithms in matching systems and which helps researchers to evaluate systems that exploit such techniques. Secondly, we present and evaluate two novel approaches in an exemplary manner implemented and evaluated with the extension in order to demonstrate its functionality. We show that RDF2Vec [30] embeddings derived directly from the ontologies to be matched are capable of representing the internal structure of an ontology but do not provide any value for matching tasks with differently structured ontologies when evaluated as the only feature. We further show that multiple feature generators and a machine learning component help to obtain a high precision alignment in the *Ontology Alignment Evaluation Initiative* (OAEI) *knowledge graph* track [11,8].

## 2 Related Work

*Classification* is a flavor of *supervised learning* and denotes a machine learning approach where the learning system is presented with a set of records carrying a *class* or *label*. Given those records, the system is trained by trying to predict the correct class. [18] Transferred to the ontology alignment domain, the set of records can be regarded as a collection of correspondences where some of the correspondences are correct (class *true*) and some correspondences are false (class *false*). Hence, the classification system at hand is binary.

The application of supervised learning is not new to ontology matching. In fact, even in the very first edition of the OAEI[3] in 2004 the *OLA* matching system [5] performed a simple optimization of weights using the provided reference alignments. In the past, multiple publications [14,4,31,24,16] addressed supervised learning in ontology matching, occasionally also referred to as *matching learning*. Unsupervised machine learning approaches are less often used, but have been proposed for the task of combining matchers as well [23].

More recently, Nkisi-Orji et al. [26] present a matching system that uses a multitude of features and a random forest classifier. The system is evaluated on the OAEI *conference* track [2] and the EuroVoc dataset, but did not participate in the actual evaluation campaign. Similarly, Wang et al. [32] present a system called *OntoEmma* which exploits a neural classifier together with 32 features. The system is evaluated on the *large biomed* track. However, the system did not participate in an OAEI campaign either. It should be mentioned here that a comparison between systems that have been trained with parts of the reference and systems that have not is not really fair (despite being the typical approach).

Also a recent, OAEI-participating matching system applies supervised learning: The *POMap++* matching system [16] uses a local classifier which is not

[3] Back then the competition was actually referred to as *EON Ontology Alignment Contest*.

based on the reference alignment but on a locally created gold standard. The system also participated in the last two recent OAEI campaigns [17,15].

The implementations of the approaches are typically not easily reusable or available in a central framework.

## 3 The MELT Framework

*Overview* MELT [10] is a framework written in Java for ontology and instance matcher development, tuning, evaluation, and packaging. It supports both, HOBBIT and SEALS, two heavily used evaluation platforms in the ontology matching community. The core parts of the framework are implemented in Java, but evaluation and packaging of matchers implemented in other languages is also supported. Since 2020, MELT is the official framework recommendation by the OAEI and the MELT track repository is used to provide all track data required by SEALS. MELT is also capable of rendering Web dashboards for ontology matching results so that interested parties can analyze and compare matching results on the level of correspondences without any coding efforts [27]. This has been pioneered at the OAEI 2019 for the *knowledge graph* track.[4] MELT is open-source[5], under a permissive license, and is available on the maven central repository[6].

*Different Gold Standard Types* Matching systems are typically evaluated against a reference alignment. A reference alignment may be complete or only partially complete. The latter means that not all entities in the matching task are aligned and that any entity not appearing in the gold standard cannot be judged. Therefore, the following five levels of completeness can be distinguished: (i) complete, (ii) partial with complete target and complete source, (iii) partial with complete target and incomplete source, (iv) partial with complete source and incomplete target, (v) partial with incomplete source and incomplete target. If the reference is complete, all correspondences not available in the reference alignment can be regarded as wrong. If only one part of the gold standard is complete (ii, iii, and iv), every correspondence involving an element of the complete side that is not available in the reference can be regarded as wrong. If the gold standard is incomplete (v), the correctness of correspondences not in the gold standard cannot be judged. For example, given that the gold standard is partial with complete target and complete source (case ii), and given the correspondence $< a, b, =, 1.0 >$, the correspondence $< a, c, =, 1.0 >$ could be judged as wrong because it involves $a$ which is from the complete side of the alignment. On the other hand, the correspondence $< d, e, =, 1.0 >$ cannot be judged because it does not involve any element from the gold standard. This evaluation setting is used for example for the OAEI *knowledge graph* track. OAEI reference datasets are typically complete

[4] For a demo of the MELT dashboard, see `https://dwslab.github.io/melt/anatomy_conference_dashboard.html`

[5] `https://github.com/dwslab/melt/`

[6] `https://mvnrepository.com/artifact/de.uni-mannheim.informatik.dws.melt`

with the exception of the *knowledge graph* track. The completeness of references influences how matching systems have to be evaluated. MELT can handle all stated levels of completeness. The completeness can be set for every `TestCase` separately using the enum `GoldStandardCompleteness`. The completeness also influences the generation of negative correspondences for a gold standard in supervised learning. MELT supports matching system developers also in this use case.

## 4 Supervised Learning Extensions in MELT

### 4.1 Python Wrapper

As researchers apply advances in machine learning and natural language processing to other domains, they often turn to Python because leading machine learning libraries such as *scikit-learn*[7], *TensorFlow*[8], *PyTorch*[9], *Keras*[10], or *gensim*[11] are not easily available for the Java language. In order to exploit functionalities provided by Python libraries in a consistent manner without a tool break, a wrapper is implemented in MELT which communicates with a Python backend via HTTP as depicted in Figure 1. The server works out-of-the-box requiring only that Python and the libraries listed in the `requirements.txt` file are available on the target system. The MELT-ML user can call methods in Java which are mapped to a Python call in the background. As of MELT 2.6, functionality from *gensim* and *scikit-learn* are wrapped.

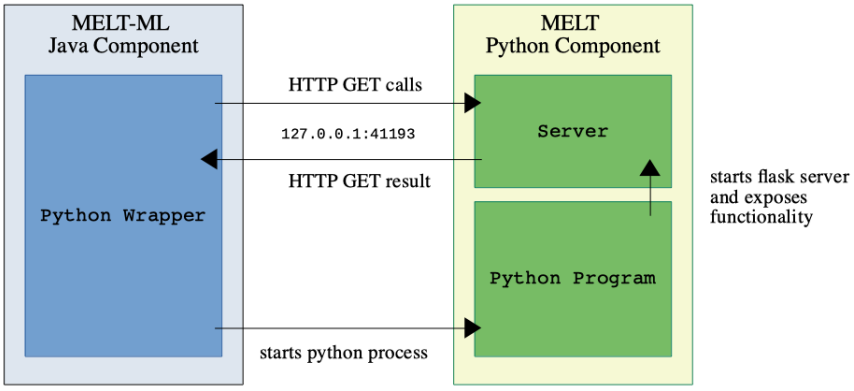


**Fig. 1.** Python code execution in MELT.

[7] `https://scikit-learn.org/`
[8] `https://www.tensorflow.org/`
[9] `https://pytorch.org/`
[10] `https://keras.io/`
[11] `https://radimrehurek.com/gensim/`

### 4.2 Generation of Training Data

Every classification approach needs features and class labels. In the case of matching, each example represents a correspondence and the overall goal is to have an ML model which is capable of deciding if a correspondence is correct or not. Thus, the matching component can only work as a filter e.g. it can only remove correspondences of an already generated alignment.

For training such a classifier, positive and negative examples are required. The positive ones can be generated by a high precision matcher or by an externally provided alignment such as a sample of the reference alignment or manually created correspondences. As mentioned earlier, no OAEI track provides a dedicated alignment for training. Therefore, MELT provides a new `sample(int n)` method in the `Alignment` class for sampling $n$ correct correspondences as well as `sampleByFraction(double fraction)` for sampling a $fraction$ in range $(0, 1)$ of correct correspondences.

Negative examples can be easily generated in settings where the gold standard is complete or partially complete (with complete source and/or target, see Section 3). The reason is that any correspondence with an entity appearing in the positive examples can be regarded as incorrect. Thus, a recall oriented matcher can generate an alignment and all such correspondences represent the negative class. In cases where the gold standard is partial and the source and/or target is incomplete, each negative correspondence has to be manually created.

### 4.3 Generation of Features

The features for the correspondences are generated by one or more matchers which can be concatenated in a pipeline or any other control flow. MELT provides an explicit framework for storing the feature values in correspondence extensions (which are by default also serialized in the alignment format). The correspondence method `addAdditionalConfidence(String key, double confidence)` is used to add such feature values (more convenience methods exist).

MELT already provides some out-of-the-box feature generators in the form of so called *filters* and *matchers*. A *matcher* detects new correspondences. As of MELT 2.6, 17 matchers are directly available (e.g., different string similarity metrics). A *filter* requires an input alignment and adds the additional confidences to the correspondences, or removes correspondences below a threshold. In MELT, machine learning is also included via a filter (`MachineLearningScikitFilter`). As of MELT 2.6, 21 filters are available. A selection is presented in the following:

*SimilarNeighboursFilter* Given an initial alignment of instances, the `SimilarNeighboursFilter` analyzes for each of the instance correspondences how many already matched neighbours the source and target instances share. It can be further customized to also include similar literals (defined by string processing methods). The share of neighbours can be added to the correspondence as absolute value or relative to the total numbers of neighbours for source and target. For the latter, the user can choose from `min` (size of the intersection divided by

minimum number of neighbours of source or target), `max`, `jaccard` (size of intersection dived by size of union), and `dice` (twice the size of intersection divided by the sum of source and target neighbours).

*CommonPropertiesFilter* This filter selects instance matches based on the overlap of properties. The idea is that equal instances also share similar properties. Especially in the case of homonyms, this filter might help. For instance, given two instances with label 'bat', the string may refer to the mammal or to the racket where the first sense has properties like 'taxon', 'age', or 'habitat' and the latter one has properties like 'material', 'quality', or 'producer'. This filter of course requires already matched properties. The added confidence can be further customized similarly to the previous filter. Furthermore, property URIs are by default filtered to exclude properties like `rdfs:label`.

*SimilarHierarchyFilter* This component analyzes any hierarchy for given instance matches such as type hierarchy or a category taxonomy as given in the *knowledge graph* track. Thus, two properties are needed: 1) instance to hierarchy property which connects the instance to the hierarchy (in case of type hierarchy this is `rdf:type`) 2) hierarchy property which connects the hierarchy (in case of type hierarchy this is `rdfs:subClassOf`). This filter needs matches in the hierarchy which are counted similarly to the previous filters. Additionally, the confidence can be computed by a hierarchy level dependent value (the higher the match in the hierarchy, the lower the confidence). `SimilarTypeFilter` is a reduced version of it by just looking at the direct parent.

*BagOfWordsSetSimilarityFilter* This filter analyzes the token overlap of the literals given by a specific property. The tokenizer can be freely chosen as well as the overlap similarity.

*MachineLearningScikitFilter* The actual classification part is implemented in class `MachineLearningScikitFilter`. In the standard setting, a five-fold cross validation is executed to search for the model with the best f-measure. The following models and hyper parameters are tested:

- *Decision Trees* optimized by minimum leaf size and maximum depth of tree (1-20)
- *Gradient Boosted Trees* optimized by maximum depth (1,6,11,16,21) and number of trees (1,21,41,61,81,101)
- *Random Forest* optimized by number of trees (1-100 with 10 steps) and minimum leaf size (1-10)
- *Naïve Bayes* (without specific parameter tuning)
- *Support Vector Machines* (SVM) with radial base function kernel; C and gamma are tuned according to [13]
- *Neural Network* with one hidden layer in two different sizes $F/2+2$, $sqrt(F)$, and two hidden layers of $F/2$ and $sqrt(F)$, where $F$ denotes the number of features

All of these combinations are evaluated automatically with and without feature normalization (`MinMaxScaler` which scales each feature to a range between zero and one). The best model is then trained on the whole training set and applied to the given alignment.

### 4.4 Analysis of Matches

A correspondence which was found by a matching system and which appears in the reference alignment is referred to as *true positive*. A *residual true positive* correspondence is a true positive correspondence that is not trivial as defined by a trivial alignment. The trivial alignment can be given or calculated by a simple baseline matcher. String matches, for instance, are often referred to as trivial. Given a reference alignment, a system alignment, and a trivial alignment, the *residual recall* can be calculated as the share of non trivial correspondences found by the matching system [1,6].

If a matcher was trained using a sample of the reference alignment and is also evaluated on the reference alignment, a true positive match can only be counted as meaningful if it was not available in the training set before. In MELT, the baseline matcher can be set dynamically for an evaluation. Therefore, for supervised matching tasks where a sample from the reference is used, the sample can be set as baseline solution (using the `ForwardMatcher`) so that only additionally found matches are counted as residual true positives. Using the alignment cube file[12], residual true positives can be analyzed at the level of individual correspondences.

## 5 Exemplary Analysis

### 5.1 RDF2Vec Vector Projections

*Experiment* In this experiment, the ontologies to be matched are embedded and a projection is used to determine matches. *RDF2Vec* is a knowledge graph embedding approach which generates random walks for each node in the graph to be embedded and afterwards runs the *word2vec* [21,22] algorithm on the generated walks. Thereby, a vector for each node in the graph is obtained. The RDF graph is used in *RDF2Vec* without any pre-processing such as in other approaches like *OWL2Vec* [12]. The embedding approach chosen here has been used on external background knowledge for ontology alignment before [29].

In this setting, we train embeddings for the ontologies to be matched. In order to do so, we integrate the *jRDF2Vec*[13] [28] framework into MELT in order to train the embedding spaces. Using the functionalities provided in the MELT-ML package, we train a linear projection from the source vector space into the target vector space. In order to generate a training dataset for the projection,

[12] The alignment cube file is a CSV file listing all correspondences found and not found (together with filtering properties) that is generated by the `EvaluatorCSV`.

[13] `https://github.com/dwslab/jRDF2Vec`

the `sampleByFraction(double fraction)` method is used. For each source, the closest target node in the embedding space is determined. If the confidence for a match is above a threshold $t$, the correspondence is added to the system alignment.

Here, we do not apply any additional matching techniques such as string matching. The approach is fully independent of any stated label information. The exemplary matching system is available online as an example.[14]

*Results* For the vector training, we generate 100 random walks with a depth of 4 per node and train skip-gram (SG) embeddings with 50 dimensions, minimum count of 1, and a window size of 5. We use a sampling rate of 50% and a threshold of 0.85. While the implemented matcher fails to generate a meaningful residual recall when the two ontologies to be matched are different, it performs very well when the ontologies are of the same structure as in the *multifarm* track. Here, the approach generates many residual true positives with a residual recall of up to 61% on *iasted-iasted* as seen in Table 1. Thus, it could be shown that *RDF2Vec* embeddings do contain structural information of the knowledge graph that is embedded.

| Multifarm Test Case | P | R | R+ | F | # of TP | # of FP | # of FN |
|---|---|---|---|---|---|---|---|
| **iasted-iasted** | 0.8232 | 0.7459 | 0.6111 | 0.7836 | 135 | 29 | 46 |
| **conference-conference** | 0.7065 | 0.5285 | 0.1967 | 0.6047 | 65 | 27 | 58 |
| **confOf-confOf** | 0.9111 | 0.5541 | 0.1081 | 0.6891 | 41 | 4 | 33 |

**Table 1.** Performance of RDF2Vec projections on the same ontologies in the multifarm track. *P* stands for *precision*, *r* stands for *recall*, and *R+* for *residual recall*. R+ refers here to the fraction of correspondences found that were previously not available in the training set. *# of ...* refers to the number of *true positives (TP)*, *false positives (FP)*, and *false negatives (FN)*. Details about the track can be found in [19]

### 5.2 Knowledge Graph Track Experiments

*Experiment* In this experiment, the instances of the OAEI *knowledge graph* track are matched. First, a basic matcher (`BaseMatcher`) is used to generate a recall oriented alignment by applying simple string matching on the property values of `rdfs:label` and `skos:altLabel`. The text is compared once using string equality and once in a normalized fashion (non-ASCII characters are removed and the whole string is lowercased).

Given this alignment, the above described feature generators / filters are applied in isolation to re-rank the correspondences and afterwards the `NaiveDescendingExtractor` [20] is used to create a one-to-one alignment based on the best confidence.

In contrast to this, another supervised approach is tried out. After executing the `BaseMatcher`, all feature generators are applied after each other where each

[14] `https://github.com/dwslab/melt/tree/master/examples/RDF2VecMatcher`

filter adds one feature value. The feature values are calculated independently of each other. This results in an alignment where each correspondence has the additional confidences in its extensions. As a last step, the `MachineLearningScikitFilter` is executed. The training alignment is generated by sampling all correspondences from the `BaseMatcher` where the source *or* target is involved. The correspondence is a positive training example if the source *and* the target appear in the input alignment (which is in our case the sampled reference alignment) and a negative example in all other cases.

The search for the machine learning model is executed as a five-fold cross validation and the best model is used to classify all correspondences given by the `BaseMatcher`. The whole setup is available on GitHub[15].

*Results* In all filters, the absolute number of overlapping entities are used (they are normalized during a grid search for the best model). In the `SimilarNeighboursFilter`, the literals are compared with text equality and the hierarchy filter compares the categories of the Wiki pages. The `SimilarTypeFilter` analyzes the direct classes which are extracted from templates (indicated by the text 'infobox'). The results for this experiment are depicted in Table 2 which shows that not one feature can be used for all test cases because different Wiki combinations (test cases) require different filters. The `BaseMatcher` already achieves a good f-measure which is also in line with previous analyses [9]. When executing the `MachineLearningScikitFilter` the precision can be increased for three test cases and the associated drop in recall is relatively small. It can be further seen that there is not one single optimal classifier out of the classifiers tested.

## 6 Conclusion and Outlook

With MELT-ML, we have presented a machine learning extension for the MELT framework which facilitates feature generation and feature combination. The latter are included as *filters* to refine existing matches. MELT also allows for the evaluation of ML-based matching systems.

In the future, we plan to extend the provided functionality by the Python wrapper to further facilitate machine learning in matching applications. We further plan to extend the number of feature generators. With our contribution we hope to encourage OAEI participants to apply and evaluate supervised matching techniques. In addition, we intend to further study different strategies and ratios for the generation of negative examples.

We further would like to emphasize that a special machine learning track with dedicated training and testing alignments might benefit the community, would increase the transparency in terms of matching system performance, and might further increase the number of participants since researchers use OAEI datasets for supervised learning but there is no official channel to participate if parts of the reference alignment are required.

[15] `https://github.com/dwslab/melt/tree/master/examples/supervisedKGTrackMatcher`

| | mcu-marvel | | | memoryalpha-memorybeta | | | memoryalpha-stexpanded | | | starwars-swg | | | starwars-swtor | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Approach** | **P** | **R** | **F** | **P** | **R** | **F** | **P** | **R** | **F** | **P** | **R** | **F** | **P** | **R** | **F** |
| BaseMatcher | 0.8548 | 0.6796 | 0.7572 | 0.8740 | 0.8978 | 0.8858 | 0.8675 | 0.9264 | 0.8960 | 0.9001 | 0.7318 | 0.8072 | 0.9007 | 0.9146 | 0.9076 |
| CommonPropertiesFilter | 0.8823 | 0.6614 | 0.7560 | 0.9310 | 0.8785 | 0.9040 | 0.9370 | 0.8968 | 0.9165 | 0.9257 | 0.7162 | 0.8076 | 0.9371 | 0.8999 | 0.9181 |
| SimilarHierarchyFilter | 0.8823 | 0.6614 | 0.7560 | 0.9361 | 0.8830 | 0.9088 | 0.9527 | 0.9107 | 0.9312 | 0.9281 | 0.7181 | 0.8097 | 0.9440 | 0.9057 | 0.9245 |
| BagOfWordsSetSimilarityFilter | 0.8823 | 0.6614 | 0.7560 | 0.9340 | 0.8810 | 0.9067 | 0.9406 | 0.8991 | 0.9194 | 0.9292 | 0.7190 | 0.8107 | 0.9348 | 0.8976 | 0.9159 |
| SimilarNeighboursFilter | 0.8912 | 0.6687 | 0.7641 | 0.9467 | 0.8916 | 0.9183 | 0.9600 | 0.9171 | 0.9380 | 0.9375 | 0.7254 | 0.8179 | 0.9317 | 0.8947 | 0.9128 |
| SimilarTypeFilter | 0.8823 | 0.6614 | 0.7560 | 0.9247 | 0.8727 | 0.8980 | 0.9303 | 0.8899 | 0.9096 | 0.9222 | 0.7135 | 0.8045 | 0.9326 | 0.8962 | 0.9140 |
| ML (sample=0.2) | 0.8831 | 0.6620 | 0.7567 | 0.9636 | 0.8592 | 0.9084 | 0.9648 | 0.8887 | 0.9252 | 0.9292 | 0.7190 | 0.8107 | 0.9621 | 0.8778 | 0.9180 |
| | SVM | | | Random Forest | | | SVM | | | SVM | | | Random Forest | | |
| ML (sample=0.4) | 0.8831 | 0.6620 | 0.7567 | 0.9636 | 0.8599 | 0.9088 | 0.9734 | 0.8690 | 0.9182 | 0.9315 | 0.7199 | 0.8121 | 0.9445 | 0.8903 | 0.9166 |
| | Random Forest | | | Random Forest | | | Neural Network | | | Neural Network | | | Random Forest | | |
| ML (sample=0.6) | 0.8831 | 0.6620 | 0.7567 | 0.9685 | 0.8575 | 0.9096 | 0.9667 | 0.8916 | 0.9276 | 0.9367 | 0.7153 | 0.8112 | 0.9565 | 0.8903 | 0.9222 |
| | Random Forest | | | Decision Tree | | | Neural Network | | | SVM | | | SVM | | |

**Table 2.** Precision (P), recall (R), and f-measure (F) for all five test cases of the *knowledge graph* track using different matching approaches. Details about the track can be found in [9]. For the ML approaches, the optimal classifier (given the evaluated ones outlined in Subsection 4.3) is stated below the scores.

## References


1. José Luis Aguirre, Bernardo Cuenca Grau, Kai Eckert, Jérôme Euzenat, Alfio Ferrara, Robert Willem Van Hague, Laura Hollink, Ernesto Jiménez-Ruiz, Christian Meilicke, Andriy Nikolov, et al. Results of the ontology alignment evaluation initiative 2012. 2012.
2. Michelle Cheatham and Pascal Hitzler. Conference v2.0: An uncertain version of the OAEI conference benchmark. In *ISWC 2014. Proceedings, Part II*, pages 33–48, 2014.
3. Jérôme David, Jérôme Euzenat, François Scharffe, and Cássia Trojahn dos Santos. The alignment API 4.0. *Semantic Web*, 2(1):3–10, 2011.
4. Kai Eckert, Christian Meilicke, and Heiner Stuckenschmidt. Improving ontology matching using meta-level learning. In *ESWC 2009, Proceedings*, pages 158–172, 2009.
5. Jérôme Euzenat, David Loup, Mohamed Touzani, and Petko Valtchev. Ontology alignment with OLA. In *EON 2004, Evaluation of Ontology-based Tools, Proceedings of the 3rd International Workshop on Evaluation of Ontology-based Tools held at ISWC 2004*, 2004.
6. Jérôme Euzenat and Pavel Shvaiko. *Ontology Matching*, chapter 9, pages 285–317. Springer, New York, 2nd edition, 2013.
7. Raúl García-Castro, Miguel Esteban-Gutiérrez, and Asunción Gómez-Pérez. Towards an infrastructure for the evaluation of semantic technologies. In *eChallenges e-2010 Conference*, pages 1–7. IEEE, 2010.
8. Sven Hertling and Heiko Paulheim. Dbkwik: A consolidated knowledge graph from thousands of wikis. In *2018 IEEE International Conference on Big Knowledge, ICBK 2018, Singapore, November 17-18, 2018*, pages 17–24, 2018.
9. Sven Hertling and Heiko Paulheim. The knowledge graph track at OAEI - gold standards, baselines, and the golden hammer bias. In *ESWC 2020, Proceedings*, pages 343–359, 2020.
10. Sven Hertling, Jan Portisch, and Heiko Paulheim. MELT - matching evaluation toolkit. In *Semantic Systems. The Power of AI and Knowledge Graphs - 15th International Conference, SEMANTiCS 2019, Proceedings*, pages 231–245, 2019.
11. Alexandra Hofmann, Samresh Perchani, Jan Portisch, Sven Hertling, and Heiko Paulheim. Dbkwik: Towards knowledge graph creation from thousands of wikis. In *Proceedings of the ISWC 2017 Posters & Demonstrations and Industry Tracks, Vienna, Austria, October 23rd - to - 25th, 2017*, 2017.
12. Ole Magnus Holter, Erik Bryhn Myklebust, Jiaoyan Chen, and Ernesto Jiménez-Ruiz. Embedding OWL ontologies with owl2vec. In *ISWC 2019 Satellite Tracks (Posters & Demonstrations, Industry, and Outrageous Ideas)*, pages 33–36, 2019.
13. Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. A practical guide to support vector classification. 2003.
14. Ryutaro Ichise. Machine learning approach for ontology mapping using multiple concept similarity measures. In *Seventh IEEE/ACIS International Conference on Computer and Information Science (icis 2008)*, pages 340–346. IEEE, 2008.
15. Amir Laadhar, Faiza Ghozzi, Imen Megdiche, Franck Ravat, Olivier Teste, and Faïez Gargouri. OAEI 2018 results of pomap++. In *OM@ISWC 2018*, pages 192–196, 2018.
16. Amir Laadhar, Faiza Ghozzi, Imen Megdiche, Franck Ravat, Olivier Teste, and Faïez Gargouri. The impact of imbalanced training data on local matching learning of ontologies. In *Business Information Systems - 22nd International Conference, BIS 2019. Proceedings, Part I*, pages 162–175, 2019.

17. Amir Laadhar, Faiza Ghozzi, Imen Megdiche, Franck Ravat, Olivier Teste, and Faïez Gargouri. Pomap++ results for OAEI 2019: Fully automated machine learning approach for ontology matching. In *OM@ISWC 2019*, pages 169–174, 2019.
18. Bing Liu. *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*. Data-centric systems and applications. Springer, Heidelberg ; New York, 2 edition, 2011.
19. Christian Meilicke, Raul Garcia-Castro, Fred Freitas, Willem Robert van Hage, Elena Montiel-Ponsoda, Ryan Ribeiro de Azevedo, Heiner Stuckenschmidt, Ondrej Sváb-Zamazal, Vojtech Svátek, Andrei Tamilin, Cássia Trojahn dos Santos, and Shenghui Wang. Multifarm: A benchmark for multilingual ontology matching. *J. Web Semant.*, 15:62–68, 2012.
20. Christian Meilicke and Heiner Stuckenschmidt. Analyzing mapping extraction approaches. In *Proceedings of the 2nd International Conference on Ontology Matching-Volume 304*, pages 25–36. CEUR-WS. org, 2007.
21. Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *1st International Conference on Learning Representations, ICLR 2013, Workshop Track Proceedings*, 2013.
22. Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.
23. Alexander C Müller and Heiko Paulheim. Towards combining ontology matchers via anomaly detection. In *OM*, pages 40–44, 2015.
24. DuyHoa Ngo, Zohra Bellahsene, and Remi Coletta. A generic approach for combining linguistic and context profile metrics in ontology matching. In *OTM Confederated International Conferences*, pages 800–807. Springer, 2011.
25. Axel-Cyrille Ngonga Ngomo and Michael Röder. Hobbit: Holistic benchmarking for big linked data. *ERCIM News*, 2016(105), 2016.
26. Ikechukwu Nkisi-Orji, Nirmalie Wiratunga, Stewart Massie, Kit-Ying Hui, and Rachel Heaven. Ontology alignment based on word embedding and random forest classification. In *ECML PKDD 2018, Proceedings, Part I*, pages 557–572, 2018.
27. Jan Portisch, Sven Hertling, Heiko Paulheim, A Visual, and Confusion Matrix. Visual analysis of ontology matching results with the melt dashboard. In *The Semantic Web: ESWC 2020 Satellite Events*, 2020.
28. Jan Portisch, Michael Hladik, and Heiko Paulheim. RDF2Vec Light - A Lightweight Approach for Knowledge Graph Embeddings. In *Proceedings of the ISWC 2020 Posters & Demonstrations*, 2020. in print.
29. Jan Portisch and Heiko Paulheim. Alod2vec matcher. In *OM@ISWC 2018*, pages 132–137, 2018.
30. Petar Ristoski, Jessica Rosati, Tommaso Di Noia, Renato De Leone, and Heiko Paulheim. Rdf2vec: RDF graph embeddings and their applications. *Semantic Web*, 10(4):721–752, 2019.
31. Bita Shadgara, Azadeh Haratian Nejhadia, and Alireza Osareha. Ontology alignment using machine learning techniques. *International Journal of Computer Science and Information Technology*, 3, 2011.
32. Lucy Lu Wang, Chandra Bhagavatula, Mark Neumann, Kyle Lo, Chris Wilhelm, and Waleed Ammar. Ontology alignment in the biomedical domain using entity definitions and context. 2018.
33. Stuart N. Wrigley, Raul Garcia-Castro, and Lyndon J. B. Nixon. Semantic evaluation at large scale (SEALS). In *Proceedings of the 21st World Wide Web Conference, WWW 2012*, pages 299–302, 2012.