VersaMatch: Ontology Matching with Weak Supervision

Jonathan Fürst
NEC Laboratories Europe
Zurich University of Applied Sciences
jonathan.fuerst@zhaw.ch

Mauricio Fadel Argerich
NEC Laboratories Europe
Universidad Politécnica de Madrid
mauricio.fadel@alumnos.upm.es

Bin Cheng
NEC Laboratories Europe
Springer Nature
bin.cheng@springernature.com

ABSTRACT

Ontology matching is crucial to data integration for across-silo data sharing and has been mainly addressed with heuristic and machine learning (ML) methods. While heuristic methods are often inflexible and hard to extend to new domains, ML methods rely on substantial and hard to obtain amounts of labeled training data. To overcome these limitations, we propose *VersaMatch*, a flexible, weakly-supervised ontology matching system. VersaMatch employs various weak supervision sources, such as heuristic rules, pattern matching, and external knowledge bases, to produce labels from a large amount of unlabeled data for training a discriminative ML model. For prediction, VersaMatch develops a novel ensemble model combining the weak supervision sources with the discriminative model to support generalization while retaining a high precision. Our ensemble method boosts end model performance by 4 points compared to a traditional weak-supervision baseline. In addition, compared to state-of-the-art ontology matchers, VersaMatch achieves an overall 4-point performance improvement in F1 score across 26 ontology combinations from different domains. For recently released, in-the-wild datasets, VersaMatch beats the next best matchers by 9 points in F1. Furthermore, its core weak-supervision logic can easily be improved by adding more knowledge sources and collecting more unlabeled data for training.

PVLDB Reference Format:
Jonathan Fürst, Mauricio Fadel Argerich, and Bin Cheng. VersaMatch: Ontology Matching with Weak Supervision. PVLDB, 16(6): 1305 - 1318, 2023.
doi:10.14778/3583140.3583148

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/nec-research/VersaMatch.

1 INTRODUCTION

Organizations have generated large amounts of data over the past years, a trend which will continue: IDC estimates that by 2025 the datasphere will reach 160 Zettabytes [61]. Still, data is usually collected with ad-hoc methodologies and kept in silos, resulting in highly heterogeneous, non-integrated data sets. This greatly increases the effort and complexity to use data and limits its value.

Hence, there is an increasing demand to break data silos for broader data utilization, both within and across organizations, specifically in data marketplaces [25]: large enterprises need to integrate and link data from different departments to construct a unified knowledge graph for more advanced customer analysis; cities and retail businesses might share their data in synergy for better retail demand prediction for businesses and better traffic management for cities; data scientists might search for relevant data sets to train machine learning (ML) models. In all of these scenarios, a harmonized data presentation layer on top of data silos is essential for data integration, linkage, and discovery.

In the past, ontologies have been introduced as a formal and standardized way to describe concepts and their relationships for harmonized data presentation, initially in the semantic web community [18] and recently in the context of knowledge graphs, such as Amazon's Product Knowledge Graph [19]. Further, there has been an uptake in expressing ontologies using ontology languages such as the Web Ontology Language (OWL) for data integration and sharing in a much wider range of domains, e.g., Health, Smart City, and Industry 4.0 [28, 31]. Ontology matching (OM) aims to find correspondences (e.g., equality) between concepts across ontologies. Since many ontologies already exist and new ones are defined and extended over time (often in a decentralized way), enabling OM at scale is crucial to data harmonization and integration across silos.

Three types of systems can perform OM from different perspectives: ontology/schema matchers that utilize the specific syntax of classes and their relationships; entity matchers that treat each class as a generic entity with a set of flat attributes; knowledge graph alignment tools that process ontologies as a generic graph (see Section 7). There are mainly two sets of core methods behind these systems: heuristics-based and learning-based. Heuristics-based methods [24, 35, 51] have shown superior performance in many of the yearly Ontology Alignment Evaluation Initiative (OAEI) challenges [54]. However, they are inflexible and hard to extend to other domains. E.g., specialized matchers have been designed for specific domains such as biology [14]. Most important, these matchers cannot benefit from learning a better matching model from data since their performance is limited by their designed heuristics and matching strategy. In contrast, learning-based approaches are not limited by defined heuristics, but rely on substantial training data to train advanced ML models [21, 36, 40, 49]. Thus, their performance is limited by the amount of available labeled data.

To address the labeling bottleneck, others have proposed active learning [45, 48] to train ML models with a few selected samples that domain experts annotate. Still, active learning struggles to bootstrap due to a random selection strategy initially, and its efficiency is affected by the highly imbalanced data found in ontology matching [50]. Lately, weak supervision systems such as Snorkel [59] can generate high-quality training data for unlabeled data based on a set of user-defined labeling functions that provide noisy supervision

Proceedings of the VLDB Endowment, Vol. 16, No. 6 ISSN 2150-8097.
doi:10.14778/3583140.3583148

signals (e.g., based on heuristics or external knowledge) by combining their signals in a generative model. The aggregated outputs are then used to train a traditional, discriminative ML model. Weak supervision has shown to be a promising approach to address the labeling bottleneck [2, 20]. *However, directly applying weak supervision into OM does not lead to good performance due to the following gaps we identified:*

First, the discriminative ML model used for final prediction has problems in a highly imbalanced task such as OM, as it cannot generalize well to long tail and rare matching patterns. For such rare patterns, generative models have favorable performance, as has also been shown recently for information extraction—another highly imbalanced task [7]. Second, there are obstacles for domain experts to write labeling functions efficiently and collaboratively because of the lack of a common view/interface through which matching information is available to compare and judge the similarity of two ontology concepts. Without a common view, labeling functions are not transferable and reusable across datasets [2]. Third, in OM, often highly-precise labeling functions are available for few of the concept combinations (e.g., based on existing alignments/knowledge bases or strong heuristics, such as logical inference). In our experience, their outputs might be omitted in the generative modeling process when several, less precise functions with opposite outputs exist.

To address these gaps, we propose *VersaMatch*, a novel weak supervision system for OM. VersaMatch matches concepts across ontologies through (1) a self-contained, enriched, and homogenized alignment profile for each ontology concept; (2) an efficient MinHash LSH based blocking designed for OM using the information from the alignment profiles; (3) generating labels (i.e., match/no-match) for all remaining combinations of ontology concepts from a core set of generic hard and soft labeling functions into a label generator; (4) training a discriminative ML model with these labels combining lexical, semantic and structural features; (5) combining weak supervision sources and discriminative model into an ensemble to classify concept combinations into match and no-match.

Compared with existing OM approaches, VersaMatch removes the need to design and configure multiple heuristic matchers/-matching strategies and eliminates the effort and cost of collecting extensive training data. Compared with a common weak supervision approach [59], VersaMatch can sustain a high recall and a high precision with the same set of transferable labeling functions. We evaluate VersaMatch on 26 ontology combinations with $\approx$ 10M class pairs, taken from a wide range of existing benchmark datasets and in-the-wild industry datasets. VersaMatch achieves a 4 points improvement in F1 score on average, compared with state-of-the-art ontology matchers across all datasets with our core weak-supervision logic consisting of only 14 functions with a total of 262 lines of code. Even for new, in-the-wild datasets, VersaMatch achieves an improvement of 9 points in F1, indicating its applicability in the context of a data marketplace. Our experimental results show further that VersaMatch is easily extendable, and its performance can be further improved by adding more weak supervision sources based on domain knowledge or more unlabeled data. We summarize the main contributions of this paper as follows:

- We design and implement *VersaMatch*, a novel end-to-end OM system with several building blocks to customize general weak supervision to the OM problem: self-contained alignment profiles provide a uniform interface for blocking, labeling and training; a MinHash LSH based blocking approach designed for OM provides high recall with few candidate pairs; a set of transferable soft and hard weak supervision functions produces high-quality weak labels across different domains; a comprehensive feature representation of lexical, structural, and semantic distances allows learning from highly imbalanced data.
- For prediction, VersaMatch develops a novel ensemble model combining the soft and hard generative model outputs with the discriminative model for higher recall while sustaining precision levels (i.e., our model supports generalization, while retaining precision). Our classification ensemble improves performance by 4 points in F1 compared with the approach in Snorkel [59], a popular weak-supervision framework.
- Our experimental evaluation uncovers multiple interesting findings: blocking helps to reduce not only complexity but also improves matching results; training with generated weak labels achieves results on-par with a model trained on manually, expert-created labels; and our ensemble of label generator and discriminative model provides better and more robust performance over different data sets. Overall, VersaMatch shows a substantial 4-point improvement over the next best entity, ontology and schema matcher across all datasets, and a 9 point improvement on in-the-wild data.

The paper is structured as follows. In Section 2 we define the OM problem and outline its challenges. Section 3 provides an overview of VersaMatch, while Section 4 and Section 5 describe our preparation and matching phase respectively. We present results in Section 6 followed by related work in Section 7 and our conclusion in Section 8.

2 ONTOLOGY MATCHING PROBLEM

The goal of ontology matching is to find semantic mappings between concepts in different ontologies. We formally define it as follows [23, 46, 52]. Ontologies encode knowledge in form of concepts and relationships in a defined language L. Here, we assume that ontologies are modeled in the Web Ontology Language (OWL) and stored in syntax formats, such as RDF/XML, Turtle, and N-Triples. For each ontology O in language L, a function $f(O)$ defines all matchable elements (e.g., classes, properties or instances). Given two ontologies, a source ontology O_s and a target ontology O_t, the matching task is to determine mappings between the matchable elements in both ontologies, as shown in Figure 1. Mappings are 4-tuples of the form $\langle e_s, e_t, r, p\rangle$ where:

- $e_s \in f(O_s)$ and $e_t \in f(O_t)$ represent matchable elements from both ontologies.
- $r \in R$ defines the semantic relation, e.g., equality/sameAs ($\equiv$) , subsumption ($\sqsubseteq$) and supersumption ($\sqsupseteq$). Here we only consider $R = \{\equiv\}$.
- p is a probability value that represents the confidence of the mapping.

Our goal is to automate OM through machine learning, i.e., we train a ML model that discriminates between mappings and non-mappings. Further, we focus here to $f(O)$ returning classes to enable a wide comparison with related work and datasets.

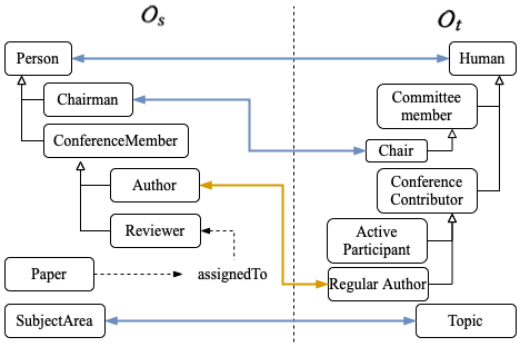

Figure 1: Depicted are selected classes from two ontologies and the corresponding matches between them, based on [22]. The blue lines denote an equality/sameAs relationship, while the orange line denotes subsumption (from left to right.)

2.1 Challenges

Although several approaches for automated OM exist (see Section 7), there are still several challenges, especially with learning-based approaches:

- **[C1]** *Lack of labeled data* for training. $N * M$ combinations of element pairs exist between two ontologies, N and M being the number of matchable elements in O_s and O_t respectively. Manual labeling requires extensive human effort and is often done by domain experts. Active learning can support labeling, but requires $\approx$ 1000 user queries for typical entity matching datasets [15].
- **[C2]** *Limited flexibility* to adapt to new datasets and domains. Many state-of-the-art matchers use multiple carefully designed matching heuristics and strategies that need to be configured (e.g., based on ontology size or domain) for high recall and precision. Many matchers specialize on domains and baseline datasets. This can lead to unstable and limited performance for in-the-wild datasets or across domains. A flexible ontology matcher should adapt to a wide range of ontologies, being able to deal with more noise and low coverage for some of the matchable elements.
- **[C3]** *High imbalance* between matches and non-matches. From the $N * M$ element combinations, most are non-matches. Machine learning models need several data samples for each class (i.e., match, non-match) to learn how to characterize and predict them. Because there are few positive samples in the OM problem, ML models struggle to generalize, especially to long-tail samples [50].
- **[C4]** $O(N * M)$ *complexity* of combinations of element pairs. The size of resulting combinations becomes too complex for larger ontologies, especially with more sophisticated matching features such as word embeddings. We need blocking techniques [55] designed explicitly for OM to make systems scale and enable interactive matching.

3 VERSAMATCH OVERVIEW

VersaMatch builds on recent advances in weak supervision [59], which we adapt and integrate into a flexible method for OM that combines both (1) matching knowledge, provided in the form of heuristic rules, and (2) machine learning to achieve a hybrid matching approach in the form of a novel classification ensemble. VersaMatch is carried out via the following steps, split into two phases, *preparation* and *matching* (see Figure 2), and mapped with the corresponding identified challenges.

(1) **Alignment Profile Construction.** We create a self-contained, homogeneous alignment profile for each class element within its overall context (i.e., its properties, neighbors in the ontology, statistical properties of its data instances if available). We also enrich information contained in the raw ontology with additional external context derived from natural language processing (NLP) and general knowledge bases (e.g., synonyms, word stems, Wikidata) and pre-compute word and sentence embeddings. [**C1, C2**]

(2) **MinHash LSH Blocking.** The overall naive combination of class pairs is of $O(N * M)$ complexity, which slows down labeling and model training. Therefore we design a MinHash locality-sensitive hashing (LSH) based blocking step, primarily to process larger ontologies. LSH uses the class context information in the previously constructed alignment profiles that we store in MinHash buckets. The output is a significantly reduced number of candidate pairs. [**C4**]

(3) **Weakly-Supervised Labeling.** We create probabilistic labeled data from the candidate combinations of class pairs based on their constructed alignment profiles using a set of generic soft and hard labeling functions that encode supervision sources. Each labeling function can judge if a pair of classes match or it can abstain. Soft functions have noisy outputs whose accuracy only needs to be $>$ 50% as in Snorkel [59], whereas hard labeling functions produce highly precise outputs ($>$ 90%) that tend to be sparse (i.e., with many abstains). Labeling functions can be flexibly removed or added. The output of all labeling functions is then used in a generative process that combines the aggregated outputs to a single probabilistic label for each covered combination. [**C1, C2**]

(4) **Discriminative ML Model.** We train a supervised, discriminative machine learning model with the covered probabilistic labels obtained in the previous step. As training data, we create a feature vector based on multiple lexical, structural, and semantic distances for each pair of classes, using state-of-the-art word embeddings. We then apply the trained model to all candidate pairs (both covered and abstained combinations of the label generator). [**C3**]

(5) **Classification Ensemble.** Last, we ensemble the generative label model's outputs and the trained discriminative ML model's, utilizing each combinations probability and coverage information. The output is an ontology alignment between source and target ontology covering all candidate pairs while achieving higher recall and precision than the generative or discriminative model alone. [**C2**]

4 PREPARATION PHASE

We create alignment profiles for each class (4.1) and then a filtered set of candidate class pairs with MinHash LSH blocking (4.2).

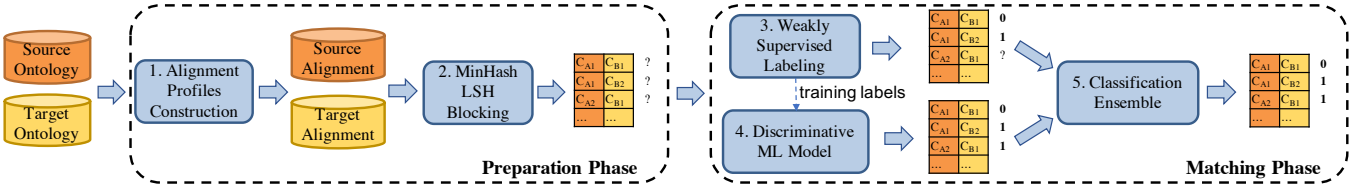

Figure 2: VersaMatch Overview. (1) The preparation phase builds alignment profiles for all source and target ontology classes and then removes unlikely class combinations based on MinHash LSH blocking. (2) For matching, we apply a weakly-supervised and a discriminative, machine learning based model jointly in a classification ensemble to the reduced set of combinations.

4.1 Alignment Profile Construction

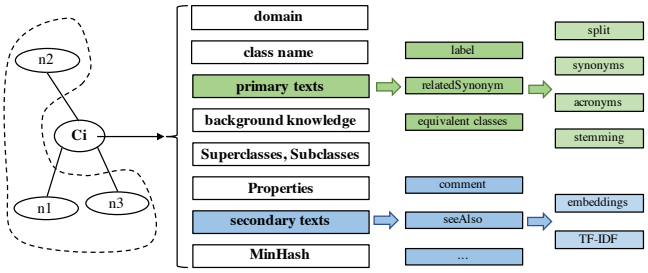

Figure 3: Alignment Profile. The alignment profile dynamically adapts to the information available in the specific ontologies and provides an easy and common interface for blocking, labeling function developers and model training.

Alignment profiles enable a comparison of two ontology elements within their context, i.e., considering their properties, instances and relationships. We also enrich them with data from open knowledge bases. E.g., a class named 'ConfPaper' might have the label 'Conference Paper', with 'Publication' as a super class, and 'report' and 'composition,' as synonyms. The self-contained and homogeneous profiles serve as common interface for all further steps, from blocking and labeling to ML model training and classification. Their simple interface allows an effortless writing of labeling functions (see Section 5.1) and they are transferable between arbitrary ontologies in the same language L. Last, they reduce the execution time of labeling and prediction as compute intense operations such as word embeddings are only computed once.

As illustrated by Figure 3, the profile includes not only basic information such as domain, comments, name, properties, and their ranges but also other context and structural information in terms of how it is related to other neighboring nodes in the overall ontology and what are the statistical features of its individuals if available. We also enrich the profile with information from external knowledge bases, such as WordNet [57] and Wikidata [69], to retrieve external, background context information (e.g., synonyms, descriptions). We design the profiles to dynamically adjust to the underlying ontologies through a set of canonical attributes. E.g., the `primary_texts` attribute contains all available "first-class" properties, such as "label" or "equivalent classes" available in the ontology language L, while `secondary_texts` contains "second-class" properties such as "comment". Further, we automatically prepare multiple variations (e.g., acronyms, splits) and pre-compute word embeddings.

4.2 MinHash LSH based Blocking

Blocking reduces the number of candidate pairs (e.g., entities, data records) that are considered for subsequent more compute intense operations, such as labeling or matching. Recently, MinHash LSH based blocking has shown to be efficient for related problems (e.g., entity matching) [63]. MinHash LSH groups similar pairs according to their Jaccard similarity. Jaccard's lexical similarity works well with noisy data; however, it has two main drawbacks: (1) a potentially different, similarity threshold needs to be selected for distinct combinations of datasets, and (2) it only represents lexical similarity and disregards semantic or structural similarity.

To address these drawbacks, we create MinHash [8] signatures for each class in both ontologies with $S_{O_s} = \{S_{s1}, S_{s2}, \ldots, S_{sn}\}$ and $S_{O_t} = \{S_{t1}, S_{t2}, \ldots, S_{tm}\}$, where n and m are the number of classes in O_s and O_t respectively. We then store signatures of S_{O_t} in a Locality Sensitive Hashing (LSH) [34] index to efficiently query it with signatures in S_{O_s}. This MinHash LSH data structure enables constant lookup times to find similar classes across ontologies and takes linear time to construct. The MinHash signatures are built from the alignment profile information. Specifically, we concatenate the available dataset/ontology independent data (e.g., primary/secondary texts and their variances and synonyms). We then remove common stopwords found in the NLTK database [4], to reduce the probability of adding spurious matches. We add the remaining data to MinHash, using w-shingling of $w = 1$, effectively omitting the order and context of tokens and solely using single words as similarity basis. This design choice is sensible because there exists no logical sentence order in the alignment profile compared to traditional text documents. E.g., the class name might be followed by its synonym. We want such tokens of two classes with a high overlap e.g., $S_{s1} = \{car, automobile\}$ and $S_{t1} = \{automobile, car\}$ to result in a high similarity, independently of the token order.

The MinHash LSH index needs to be initialized with the number b and size r of bands in which we split the MinHash signatures. The probability that MinHash signatures become candidate pairs depends on b and r and follows an S-curve, with the similarity threshold t being at the steepest point. The threshold can be estimated with $t = \left(\frac{1}{b}\right)^{\frac{1}{r}}$ [38]. To choose values for b, r and thereby t, we exploit that OM is more narrowly defined than general entity matching or record linkage. In OM, there can only be a single valid match for each class at maximum. I.e., out of $N * M$ combinations, where N and M are the number of classes in O_s and O_t respectively, at most $\min(N, M)$ matches exist. Thus, we can set the LSH parameters to reach a minimum number of returned candidate

pairs, starting from a high Jaccard similarity threshold and relaxing it by a stepsize S with further iterations until sufficient candidate pairs N_C are returned. In our experiments with multiple ontologies (see Section 6), we find that $N_C \geq 5 * \min(N, M)$ provides a good trade-off between size and recall across different ontologies. We stop at a smaller coverage of 2% when the number of LSH returned matches saturate for multiple consecutive steps. This algorithm has a linear complexity of $O(N + M)$ for each tried Jaccard threshold.

We mitigate MinHash's restriction to lexical similarity through the enriched information in the alignment profile. The profiles contain information from linguistic and general knowledge bases and models such as synonyms, stemming, root, and acronyms. They also contain structural features, such as super and sub-classes (see Section 4.1 for details), transformed into lexical features for which Jaccard similarity is a good measure. Note that achieving semantic similarity with LSH has also been addressed slightly differently by DeepER [21], which applies LSH directly over word embeddings of entity attributes. We have found this approach to not work as well for ontologies, as it cannot take advantage of the structure and constraints defined by ontologies and their modeling language L.

5 MATCHING PHASE

The matching phase comprises the creation of a generative labeling model based on multiple weak supervision signals (5.1), the training of a discriminative ML model (5.2), and their classification ensemble to retrieve the overall matches (5.3), see also Figure 4.

5.1 Weakly-Supervised Labeling

VersaMatch labels class combinations from O_s and O_t into *match* and *non-match* using multiple soft and hard weak-supervision functions that access the respective alignment profiles for supporting their decision. With weak supervision, we can generate training labels for—potentially large amounts of—unlabeled data based on imprecise or limited supervision signals. These signals can be modeled in a generative process that de-noises the labels based on their agreements and disagreements for each class pair [60]. We combine all soft and hard weak supervision signals into a label generator to produce probabilistic labels for ML model training.

Labeling Functions (LFs). Labeling functions encode an indication if two classes are equivalent or not and can be flexibly modified or extended by users. Each LF $f : X \rightarrow Y \cup \emptyset$ takes a class pair $x \in X$, where $x = (c_s, c_t)$ as input and outputs the latent label $y \in Y$ or abstains from voting. In VersaMatch, labeling functions directly access the information in the alignment profile created in Step (1), which provides an easy and self-adapting interface. Functions can base their decision on domain knowledge, general, experience-based heuristics, or available knowledge bases. For example, Listing 1 shows a labeling function that matches two classes based on the intersection of their synonyms from the WORDNet knowledge base [57]. As a basis, we create a core set of functions that we generally find applicable for various domains and matching problems (see Table 3). By creating a new function, this core set can be flexibly extended with domain-specific matching heuristics (e.g., domain-specific background knowledge). Further, through its alignment profile abstraction, *VersaMatch automatically adapts to the information contained in an ontology (e.g., to the available properties). The labeling functions themselves stay the same.*

```
@soft
def Primary_texts_synonyms(r):
    synonyms_x = set(r.primary_texts_synonyms_x.values())
    synonyms_y = set(r.primary_texts_synonyms_y.values())
    common = synonyms_x & synonyms_y
    if len(common) >= 1:
        return 1
    return -1
```

Listing 1: Labeling Function uses WORDNet Synonyms and returns a match if both names share a common synonym.

Label Generator. VersaMatch uses the set of weak supervision sources to compute probabilistic labels for the generated combination of classes. Beyond *Data Programming* [59], a popular weak-supervision framework, we abstract weak-supervision sources to two types of *labeling functions* (LFs). *Soft labeling functions* correspond to the traditional LF concept in [59], while our *hard labeling functions* encode precise weak-supervision sources but usually only with sparse outputs, i.e., with low coverage. For instance, a *soft LF* could label a pair of classes that share the same string as a name as a "match", producing numerous false positives due to the ambiguity of language; while a *hard LF* could label a pair of classes as a "match" if and only if they are known to be a match in a knowledge base. For all other pairs that are not known, the LF will abstain, producing a low number of votes. Such low-coverage, but high-precision functions are common in OM and cannot be handled well by the generic generative process as they do not participate in enough outputs to adequately model their accuracy. This practically can result in their votes being canceled out by other functions. Therefore, we handle them separately: (1) For the soft functions, we learn a generative model following the approach in [59], while (2) for the hard functions we apply a majority voting model due to the small number of functions and their sparse outputs.

For the generative model, consider n labeling functions and m class pairs, resulting in the labeling matrix Λ in Equation 1.

$$\Lambda = \begin{bmatrix} y_{11} & y_{12} & y_{13} & \cdots & y_{1n} \\ y_{21} & y_{22} & y_{23} & \cdots & y_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ y_{m1} & y_{m2} & y_{m3} & \cdots & y_{mn} \end{bmatrix} \tag{1}$$

Given the observed Λ, VersaMatch learns a generative model with parameters θ to produce probabilistic estimates of the latent labels y_j with $j = 1, 2, ..., m$ by modeling the LF accuracies, propensities, and their dependencies. More details can be seen in [59]. Through this generative process, the noise and variance in accuracy and coverage of each single labeling function are taken into account, putting more weight on high accuracy labeling sources and less on low accuracy ones.

For the hard LFs, we instead derive a prediction $\hat{y}_{hard}$ through majority voting across the outputs of the hard LFs (see Equation 2).

$$\hat{y}_{hard_i} = \text{argmax} \sum_{k=1}^{K} f_k(x_i) \tag{2}$$

Note, that in our experience, there are usually not many hard outputs available for a single class pair, as functions usually cover

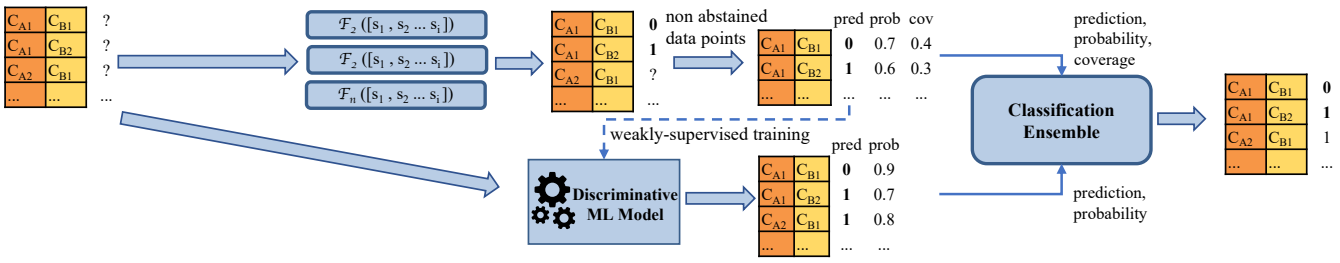

Figure 4: Detailed working of VersaMatch matching phase. A set of flexible labeling functions processes the after blocking remaining class combinations into a label generator. We use its weak-supervision signals to train a discriminative machine learning model. The outputs of the label generator and the discriminative model are then ensembled for final prediction.

different categories of matches (e.g., based on domain knowledge bases or logical inference). With such a small set, computing a generative model for the set of hard functions $F_{hard} = \{f_1, f_2, \ldots, f_k\}$ is not sensible as LFs cannot be properly modeled.

For the final training labels, we combine soft and hard models into an overall *label generator*:

$$\hat{y}_{train_i} = \begin{cases} \hat{y}_{hard_i} & \text{if } F_{hard}(x_i) \neq \emptyset \\ \hat{y}_{soft_i} & \text{otherwise} \end{cases} \tag{3}$$

where $\hat{y}_{hard}$ is the output of the majority model of the hard LFs and $\hat{y}_{soft}$ is the output of the generative model of the soft LFs. Thus, to predict the label for two ontology concepts $x_i = (c_s, c_t)$, we select $\hat{y}_{hard_i}$ if the output vector of the hard LFs for x_i is not empty, i.e., at least one hard LF exists and takes a vote for the given data point.

Post-processing of probabilistic labels. The outputs of the label generator are probabilistic labels for each covered pair of classes from the source and target ontology. We then (1) transform the probabilistic labels into deterministic labels and (2) filter out low confidence labels based on a confidence threshold. E.g., if a pair of classes has a label with a 0.55 probability of matching and our threshold is 0.7, we do not use this label for our next step.

5.2 Discriminative ML Model

There are few mappings (i.e., few positive samples) in any OM task. This constraint reverberates in our ML model design as we must use a relatively small number of features because of the curse of dimensionality [3]. We design multiple features to represent each pair of classes capturing their lexical, semantic and structural similarity and then select an optimal subset using recursive feature elimination based on ANOVA F-values (see Table 1).

We measure the lexical distance between each pair of classes by using multiple distance functions (e.g., cosine, euclidean) to represent better similarities between classes, which has been beneficial for similar tasks [42]. We also borrow the concept of TF-IDF (Term Frequency-Inverse Document Frequency) from information retrieval and adapt it to OM in the following way. First, we calculate the term frequency f for all the terms in each ontology. Then, for each pair of classes c_s, c_t from ontologies O_s and O_t we obtain the shared terms in their names $(t_1, t_2, ..., t_n)$ and calculate a score s_{c_s,c_t}

Table 1: Used features for class-pairs. Features express lexical, semantical and structural similarity between two classes.

Type	Feature
Lexical	Shared words count Shared words percentage Levenshtein distance for class names Shared synonyms count Shared synonyms percentage TF-IDF score
Semantic (Word Embeddings-based)	Cosine distance for labels (USE) Cosine distance for class names (USE) Euclidean distance for class names (USE) Cosine distance for class names (spaCy) Euclidean distance for class names (spaCy) Cosine distance for primary texts (spaCy) Cosine distance for second. texts (spaCy) Cosine distance for synonyms (spaCy) Cosine distance for prefix texts (spaCy)
Structural	Cosine distance for superclasses (spaCy) Cosine distance for subclasses (spaCy)

based on $f_1, f_2, \ldots, f_n$ in the following manner:

$$s_{c_s,c_t} = \sum_{i=1}^{n} \left(\frac{1}{f_{i,s}} + \frac{1}{f_{i,t}}\right) \tag{4}$$

where $f_{i,s}$ and $f_{i,t}$ represent the term frequencies in the ontologies.

To represent semantic similarity, we use word embeddings for semantic text vectorization. As existing embedding models have been trained on different corpora with different linguistic properties (e.g., a Bag-of-Words context-based model captures the domain and lexical aspect; a paraphrase-relationship-based model captures semantic similarities [39]), we utilize multiple embedding models. Multiple models have also achieved higher accuracy for other tasks [64]. Specifically, we employ Google's Universal Sentence Encoder (USE) [11] and spaCy [33] to compute embeddings. Our experiments with BERT models [16], showed no improvement. We suspect that this is because USE has explicitly been designed, among others, for semantic sentence similarity, whereas BERT has a much wider application range (e.g., text classification, query answering).

To encode structural information (i.e., class relationships) , we calculate the distance between the word embeddings of the primary and secondary texts for the immediate superclass of class A and the word embeddings of the same texts for the superclass of class B. We also do the same for the immediate subclasses of A and B.

ML Model Training. With the encoded alignment pairs and the generated labels, we train a supervised, discriminative ML model that can classify any given pair of classes as match or non-match. We exploit the ability of ML models to find patterns in the training data to filter out the noise present in the weak supervision signals and generalize beyond the logic contained in the labeling functions. This allows VersaMatch to classify all data points, including those not classified by our weakly-supervised labeling step, i.e., data points in which the label generator abstained from voting.

5.3 Classification Ensemble

For final classification, we combine the soft and hard predictions with the predictions of the weakly-supervised, discriminative ML model in a classification model in Equation 5. The goal of our ensembling technique is to support generalization through the discriminative model, while sustaining high recall and precision. In this setting, the soft and hard LFs act as a safety net to ensure a more precise classification of long-tail samples, for which the discriminative model has problems to generalize due to the lack of training samples. The outputs of the ensemble are defined as:

$$\hat{y}_{final_i} = \begin{cases} \hat{y}_{hard_i} & \text{if } F_{hard}(x_i) \neq \emptyset \\ \text{argmax} \sum_{k=1}^{K} \omega_{k,i}\, p_{k,m}(x_i) & \text{otherwise} \end{cases} \quad (5)$$

Where $p_{k,m}$ is the decision output matrix of the soft LFs and the discriminative ML model (Equation 6), and $\omega_{k,i}$ is the weight assigned to each point i and classifier k.

$$P = \begin{bmatrix} p_{1,1}(x_i) & p_{1,2}(x_i) & \cdots & p_{1,M}(x_i) \\ p_{2,1}(x_i) & p_{2,2}(x_i) & \cdots & p_{2,M}(x_i) \\ \vdots & \vdots & \ddots & \vdots \\ p_{K,1}(x_i) & p_{K,2}(x_i) & \cdots & p_{K,M}(x_i) \end{bmatrix} \quad (6)$$

Note that for ontology *sameAs* matching, there will be only two classes ($M = 1, 2$) and two classifiers ($K = 1, 2$) consisting of the (aggregated) soft LFs and the discriminative ML model. However, our method can be extended to more complex ensemble approaches such as bagging [6], where multiple ML models are trained on different sub-sets of the data.

Defining the weight of each classifier-data point combination is a problem of active research [44]. In VersaMatch, we exploit the LF coverage for each point as an indicator of the weight to be assigned to the generative label generator and the discriminative model. Specifically we compute $\omega_{k,i} = \frac{N_i}{N_k}$ where N_k is the combined number of "voters" (LFs + the discriminative model) and N_i is the number of voters that take a vote for a single data point. E.g., there might be 4 soft LFs in total with two abstaining from a data point, thus the soft LF weight results in: $\frac{2}{4+1}$ and the discriminative model weight results in: $\frac{1}{4+1}$. These weights are then multiplied with the respective probabilities vector and the resulting vectors are added to compute the ensemble of probabilities. We then select the class corresponding to the highest value with argmax. Note, that if there is also an output of a hard LF, we select this output instead ($\hat{y}_{hard_i}$).

6 EXPERIMENTAL EVALUATION

We describe datasets, baselines and explore different questions defined in our experimental design through experiments.

Datasets: we select two long-running datasets, *conference* and *anatomy*, from the Ontology Alignment Evaluation Initiative (OAEI), in which state-of-the-art ontology matchers have competed every year since 2005 (22 matching tasks over eight ontologies) [54]. We further take two ontologies, *Atmonto* and *AIRM-O*, from the *air traffic* management domain as a realistic scenario of two larger ontologies of the same domain that have been developed independently and according to [67] "...pose a challenge to automatic matching". Last, we select a set of IoT ontologies published as part of a challenge of the web and information extraction community [30], which is the topic of an ongoing AI competition *KE4WoT* [65]. For the OAEI data, we use the provided reference alignments; for the air traffic data, we use a manually created alignment [67], while for the KE4WoT data, we extract an alignment based on matching information published by the challenge organizers and manual inspection of the ontologies. We provide this new dataset with this paper [26]. Table 2 summarizes our datasets.

Table 2: Datasets used in our evaluation.

Dataset	Description	Ontologies	#classes	#matches
Conference	OAEI-2020 Conference track involves 7 ontologies from the same domain (conference organization) [54].	CMT, ConfOf, Conference, Edas, Ekaw, Iasted, Sigkdd	491	259
Anatomy	OAEI-2020 Anatomy track is a real world case that contains 2 large ontologies describing human and mouse anatomy [54].	NCI Thesaurus (Human), Mouse	6048	1516
AirTraffic	NASA and Eurocontrol air traffic management ontologies describe classes, properties, and relationships relevant to the domain of air traffic management.	Atmonto [62], AIRMO [68]	1069	32
AI4EU	IoT ontologies part of AI4AI KE4WoT [65] and WWW'18 challenge [53].	SAREF, M3-lite, FIESTA-IoT, dogont	2061	72
	Sum	**15**	**9669**	**1876**

Baseline Systems: first, we select the following state-of-the-art ontology matching systems as baseline systems for comparison: (1) LogMap; (2) AML [24]; (3) YAM++ [51] (unsupervised mode). They are the top-ranked matchers for the first two datasets according to results on OAEI. They are all unsupervised, i.e., they rely on a set of matching heuristics and potentially post-processing/filtering steps. Second, we select a set of 5 tabular schema matchers (COMA 3.0 [1, 17], Cupid [43], Distribution Based [75], JaccardLevenMatcher and Similarity Flooding [47]) based on their implementation in Valentine [37], an experimental suite that bundles multiple matchers. Last, we also select three supervised entity matchers, Magellan [36], DeepMatcher [49] and DITTO [40], based on their popularity in the entity matching domain and to compare VersaMatch to traditional supervised ML methods.

Metrics: in our experiments, we use precision, recall, and F1 score to evaluate our approach. Compared to accuracy, precision and recall evaluate the proportion of actual positive samples in all the predicted positive ones and the proportion of actual positive

samples found from the total positive samples, respectively. F1 score combines these two and offers a way to measure if a system is biased—if it is biased to output negative samples, then its precision might be very high but its recall low and vice versa.

6.1 Experimental Design

Our experiments are designed to explore the following questions with detailed evaluation results reported in the follow-up sections.

First, we evaluate the effectiveness of our weak supervision-based approach to label input data for OM. We try to answer two main questions: (1) whether weak-supervision provides sufficiently accurate labels and how much data it can cover; (2) whether additional weak-supervision sources improves the overall labeling performance and how well VersaMatch adapts to newly-added weak supervision sources to improve the quality of generated labels. The results of these questions reflect the flexibility of VersaMatch in the knowledge dimension, called *improve-by-expanding-knowledge*, meaning the performance of VersaMatch can be further improved by adding additional, domain-specific knowledge via the same mechanism.

Second, we investigate how VersaMatch learns and generalizes from the weak labels generated by our set of weak-supervision sources. We explore which ML algorithm works best and the gain compared to the label generator. We also explore if more generated weak labels could further improve the learning capability. The corresponding results reflect the flexibility of VersaMatch in the data dimension, called *improve-by-adding-more-unlabeled-data*, meaning the performance of ML models can be improved by simply adding more unlabeled datasets, which are easy to obtain.

Third, we validate the effectiveness of combining both generative and discriminative ML models to predict the final matching results. We measure if the results from the classification ensemble are better than taking either model alone. This result reflects the flexibility of VersaMatch in the model dimension, called *improve-by-combining-models*, meaning we can improve the final matching results by taking advantage of both generative and discriminative ML models.

Last, we compare VersaMatch with the selected baseline systems over four datasets to see if we can go beyond their performance by using our core set of provided weak supervision sources. This helps us see the overall performance of VersaMatch and its potential to provide well-performing OM at scale across different datasets.

6.2 Label Generator

Analysis of Labeling Functions. We start with basic experiments to analyze the performance and coverage of labeling functions (LFs). We use 14 LFs in VersaMatch of which 2 are hard functions. Table 3 summarizes them including their Lines of Code (LoC). We can make the following observations: First, the required effort to program an LF is relatively low in LoC. The average lines of code per LF are about 20, and most of them consist of less than 50 lines of code and contain a rather simple logic (e.g., they check for equality of alignment profile attributes). Second, the coverage of LFs, i.e., the percentage of the class pairs covered by each LF, varies tremendously. Notice that 10 out of 14 LFs cover $< 2\%$ of the class pairs, and only 1 LF has a coverage $> 50\%$. Most of our LFs are conservative, and provide votes only when they are certain about the cases. This is why the precision of the majority of LFs is high (macro average 80%). Figure 5 shows the fraction of class pairs covered by different numbers of LFs in the combined dataset. We notice that about 40% of the class pairs are not covered by any LF, resulting in the label generator not being able to predict/label those points. Last, we can see two hard functions. `DomainKB` uses alignment info which is known to be true from a domain specific knowledge base, which is common in many application scenarios; `Subclasses_equal` encodes a logical inductive inference (if all subclasses are equivalent, the parent classes must be too).

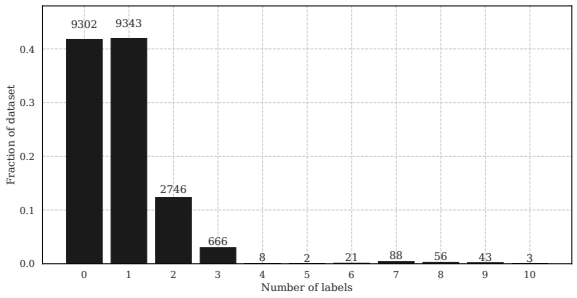

Figure 5: LF coverage across datasets (without anatomy).

Adding Knowledge Sources. To explore how additional knowledge supports the label generator, we separate LFs into different categories based on their "knowledge source". E.g., the "Name" category contains functions that only perform simple lexical comparisons between names of two classes, the "Embedding" category contains functions accessing more advanced models such as Universal Sentence Encoder [11]. We perform the same set of experiments on all datasets so that we can examine how adding new knowledge sources changes the label quality across datasets. Figure 6 shows the label generator's average performance results and coverage for all four data sets with increased knowledge sources. We compute F1, precision, and recall with ground truth data. Coverage measures the percentage of the class pairs that the label generator can label.

Overall, the following observations can be seen in Figure 6. First, *adding knowledge sources helps improve F1 and recall of the label generator gradually and consistently.* This indicates that the label generator can de-noise the signals from labeling functions and then consolidate them to achieve better label quality. Second, by adding more knowledge sources, the label generator can cover and label $\approx$ 20 times more class pairs, increasing from 0.02 to 0.37. More important, as more class pairs are covered, overall quality of the generated labels is also improved: F1 score increases by 16%, from 0.62 to 0.72, and recall increases by 40%, from 0.51 to 0.71. Only precision slightly decreases after we add the 4th source. Overall, VersaMatch balances precision and recall with growing number of knowledge sources, reflected by an increasing F1 score.

6.3 Discriminative ML Model

Based on the weak labels generated by the label generator we train and evaluate a discriminative ML model via a set of experiments.

Model Selection. Using the features described in Table 1, we evaluate four machine learning models: (1) Logistic Regression (LR), (2) Random Forests (RF), (3) XGBoost (XGB), and (4) a Neural Network (NN). The NN consists of 3 dense hidden layers of 256 neurons

Table 3: Core Labeling Function Evaluation Analysis. (see [26] for the implementation of these functions)

Labeling Function	Description	Type	Cov.	Prec.	LoC
`Name_equal`	Matches classes if they have the exact same name.	soft	1.11 %	87.8 %	3
`Split_equal`	Splits names (e.g., based on camel case names) for match.	soft	0.78 %	70.2 %	65
`Class_name_spacy`	Matches based on spacy distance.	soft	11.82 %	88.2 %	7
`Split_stemmed_equal`	Splits names and stemmes them for match.	soft	0.47 %	85.7 %	22
`Primary_texts_equal_0`	Matches based on exact matches of primary texts (label, equivalent classes...)	soft	1.16 %	87.6 %	7
`Primary_texts_leven`	Matches based on Levenshtein distance of primary texts	soft	23.03 %	75.5 %	34
`Primary_texts_use`	Matches based on USE distance of primary texts.	soft	51.34 %	84.5 %	33
`Primary_texts_synonyms`	Matches based on overlap in synonyms.	soft	1.01 %	79.1 %	9
`Acronyms`	Matches acronyms based on regex, such as VLDB and Very Large Data Bases.	soft	0.16 %	50.0 %	22
`Superclasses_equal`	Matches based on superclasses equality.	soft	0.26 %	54.0 %	18
`Subclasses_equal`	Matches based on subclasses equality.	hard	0.02 %	100 %	18
`DomainKB`	Matches based on domain knowledge alignments (Uberon alignments [32]).	hard	3.20 %	97.4 %	5
`Wikidata`	Matches based on WikiData KB info.	soft	0.10 %	79.2 %	13
`Root_nouns`	Matches based on equality of word roots.	soft	1.10 %	88.3 %	6

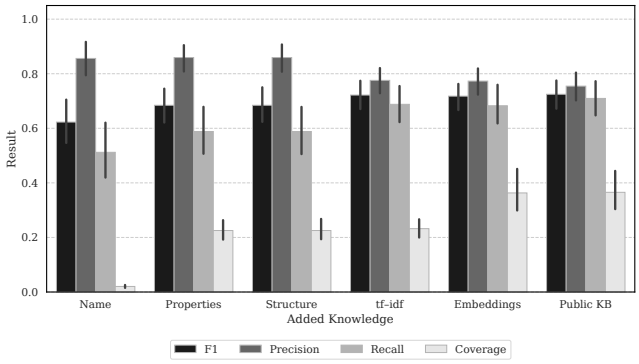

Figure 6: Adding knowledge sources. We iteratively add knowledge sources, i.e., "Name" contains only simple distance-related functions, "tf-idf" accesses *additionally* term frequency–inverse document frequency knowledge etc.

and was implemented using Keras and Tensorflow. This selection covers a wide range of ML models used in related works [19, 49] for supervised learning approaches. We carry out a group of experiments to measure the performance of these ML models in each dataset so that the trained ML model can adapt to the characteristic of each dataset. This design is on purpose because VersaMatch can always generate weak labels to train the discriminative ML models adapting them to new datasets. The whole process does not involve extra human effort. In each experiment, the training is done over part of the dataset for the class pairs with available weak labels, while the test is performed over the whole dataset for all class pairs.

Table 4 shows the performance results of all models over each dataset. First, model performance varies largely across datasets with different trade-offs between precision and recall. RF shows the best F1 score for three out of four datasets, and only for Anatomy, its performance is slightly behind XGB. Second, the most advanced and compute-intensive model (NN) cannot outperform simpler models, such as LR and RF. There are two reasons behind that: (1) the derived features from the structured alignment profiles enable those simple models to learn effectively and (2) the learning capability of NN is limited due by the lack of samples (especially positive samples). A similar pattern, where a NN underperforms for structured data, has been reported in related entity matching works [19]. We select RF as the best model and use it for further analysis of VersaMatch.

Table 4: Comparison of results for different discriminative models across all datasets (macro average).

Dataset	Model	F1	Precision	Recall	No. Labels and Features
AI4EU	LR	0.76	0.92	0.65	7781 weak labels
	RF	**0.80**	0.96	0.68	
	XGB	0.77	0.77	0.78	17 features
	NN	0.78	0.88	0.69	
Conference	LR	0.69	0.83	0.59	2546 weak labels
	RF	**0.75**	0.83	0.68	
	XGB	0.73	0.71	0.76	17 features
	NN	0.68	0.82	0.58	
AirTraffic	LR	0.52	0.86	0.38	2714 weak labels
	RF	**0.60**	0.93	0.44	
	XGB	0.58	0.63	0.53	17 features
	NN	0.45	0.83	0.31	
Anatomy	LR	0.83	0.98	0.72	190510 weak labels
	RF	0.82	0.98	0.71	
	XGB	**0.86**	0.93	0.80	17 features
	NN	0.79	0.97	0.67	

Weak Label vs. Golden Label. Using Random Forest as the selected discriminative ML model, we further explore how learning could be improved when training with more generated weak labels and also compare that to training it with ground-truth *golden labels*. We create 10 stratified folds with 75% train and 25% test size for all datasets combined. For each fold, we then train the RF model with different training data sizes, starting with only 5% of the combined training data at first and then increasing train size in steps of 5% until we use the full 75%. The same experiment is carried out for two cases: 1) training with the weak labels and 2) training with the golden labels. Figure 7 plots the average value and standard

error confidence interval of F1, precision, and recall for both cases. We can observe the following. First, increasing training data helps improve ML model performance, and the improvement can be seen from all 3 metrics for both weak and golden labels. The improvement is substantial initially when increasing from 5% to 40%, but later becomes negligible. Second, *training with weak labels achieves nearly the same F1 score as with golden labels and even provides higher recall*, meaning that our approach with weak supervision can capture more true matches in its prediction result. This is preferred in many practical cases because often, the predicted matches will be further verified by domain experts. Surprisingly, training with weak labels sometimes leads to an even higher F1 score than training with golden labels. We suspect that this is because the label generator might label some non-matches as matches, which still positively impacts the model training as other actual matches have a high similarity to the wrongly labeled non-match.

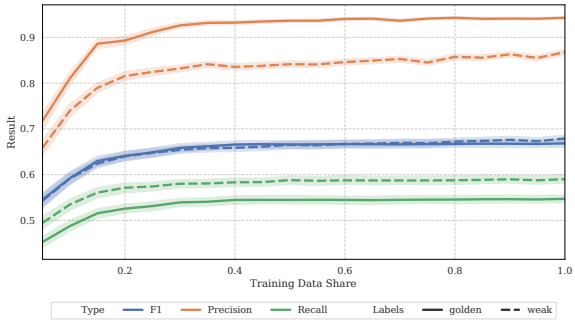

Figure 7: Performance of Random Forest ML model over increased training data size for weak and golden labels.

6.4 Model Ensemble

The discriminative model is trained with labels generated by the generative model over non-abstain data covered by the core LFs. After that, we can apply it directly to the abstain data, not covered by any LF. This is because the discriminative model can learn hidden patterns in the data and generalize its prediction capability from the non-abstain data to the abstain data. In VersaMatch, the model ensemble is designed to take advantage of both the label generator and the discriminative model to provide the best prediction results for the entire dataset. To examine the benefit of the model ensemble, we perform a set of experiments to measure the performance of all these three models in terms of F1, precision, and recall in two different cases: one with only the non-abstain data covered by LFs and the other with all data, including both the non-abstain data and the abstain data that LFs do not cover.

Table 5 reports the performance results. No result is provided for the label generator over all data because it does not apply to the abstained data. Our model ensemble achieves the best performance in terms of F1 score and recall in both cases. Especially for all data, the ensemble model in VersaMatch gains 4 points on F1 score compared to the best discriminative model. In this case, the LG output is used for 48% of class pairs, while the DM is used for 52% of class pairs. These results clearly show the benefit of combining both models to predict the final matching result.

Table 5: Performance of Label Generator (LG), Discriminative Model (DM), Ensemble Model (EM).

	all			non-abstain		
	F1	P	R	F1	P	R
LG				0.86	0.82	0.90
DM	0.74	0.93	0.63	0.84	0.93	0.77
$EM_{(VersaMatch)}$	**0.78**	0.82	0.74	**0.86**	0.82	0.90

6.5 Comparison with Baselines

Overall Performance. Last, we compare VersaMatch against state-of-the-art matchers on our test datasets, among them also two entity matchers, Magellan [36] and DeepMatcher [49] (see Table 6). Overall, *VersaMatch achieves the best F1 score for 3 out of 4 datasets.* Only for the anatomy dataset, AML is ahead by 2 points. However, AML performs a post-matching coherence/repair step, which greatly improves its performance. When run without this repair step, $AML_{NoRepair}$ only achieves 0.81 F1, 11 points lower than VersaMatch's 0.92. Currently, VersaMatch has no coherence/repair functionality. Therefore, we suspect that our results can be further improved with similar functionality. More importantly, we see that for the two in-the-wild datasets (AirTraffic & AI4EU), VersaMatch achieves 8 and 9 points higher F1 than the other methods. One reason could be that existing methods have been well-tuned to the two datasets published by OAEI in its past competitions but fail to provide the same competitive performance over new datasets. VersaMatch's big performance gain over these new datasets demonstrates its flexibility and adaptation capabilities based on its alignment profile, a set of core labeling functions, and the ensemble of generative and discriminative models for final classification. Overall, we see that VersaMatch achieves its high F1 score through a substantially higher recall (8% better than the 2nd), while sustaining a competitive precision. We trace this behavior to several features of our design. First, the enriched information in our alignment profile allows labeling functions to take different perspectives and knowledge into their decision. Second, our label generator can consolidate the weak signals from various labeling functions to achieve a better trade-off between recall and precision. Last, as seen from Table 5, our ensemble model is able to leverage both the performance of the learned label generator for the non-abstained points and the performance of the discriminative model for the abstained points.

Schema Matchers. We evaluate the 5 schema matchers contained in Valentine [37] by transforming all alignment profiles of O_s and O_t into two tables with where each column represents a different class and the rows contain the attributes of the alignment profiles. On average, this leads only to a performance of 0.114 ± 0.0163 in F1 (in Table 6 we provide detailed results for two of the 5 tested schema matchers). There are no matches found for the anatomy dataset as the column names (based on the class names) are IDs that do not provide any meaningful similarity comparison.

Entity Matchers. We evaluate Magellan, DeepMatcher and DITTO by providing them with two flattened tables of the ontologies, with one class per row, while columns correspond to attributes of the alignment profiles. We use the Magellan blocker

Table 6: VersaMatch vs. Baselines. VersaMatch achieves highest F1 score on 3 datasets, while for Anatomy, AML can only achieve 2 point higher results with an additional repair step, not implemented in VersaMatch.

		Anatomy			Conference			AirTraffic			AI4EU			Overall		
		F1	Prec.	Recall	F1	Prec.	Recall	F1	Prec.	Recall	F1	Prec.	Recall	F1	Prec.	Recall
Ontology Matchers	**VersaMatch**	0.92	**0.96**	0.88	**0.78**	0.79	**0.76**	**0.60**	0.68	**0.53**	**0.81**	**0.84**	**0.79**	**0.78**	0.82	**0.74**
	AML	**0.94**	**0.96**	**0.93**	0.76	0.83	0.70	0.51	0.80	0.38	0.68	0.70	0.65	0.74	0.82	0.66
	$AML_{NoRepair}$	0.81	0.77	0.86	0.71	**0.84**	0.62	0.50	0.75	0.38	0.68	0.70	0.65	0.68	0.76	0.63
	Yam++	0.85	**0.96**	0.77	0.63	0.74	0.55	0.47	0.91	0.31	0.71	0.68	0.75	0.67	0.82	0.60
	LogMap	0.88	0.91	0.84	0.73	**0.84**	0.64	0.48	**1.00**	0.31	0.70	0.72	0.68	0.70	0.87	0.62
Schema Matchers	Distribution Based	0	0	0	0.11	0.07	0.41	0.10	0.06	0.28	0.24	0.16	0.56	0.11	0.07	0.31
	JaccardLevenMatcher	0	0	0	0.15	0.10	0.41	0.10	0.06	0.28	0.24	0.16	0.56	0.12	0.08	0.32
Entity Matchers	DeepMatcher	0.24	0.33	0.19	0.63	0.67	0.58	0.20	0.47	0.10	0.70	0.82	0.63	0.44	0.60	0.38
	Magellan	0.61	0.87	0.47	0.58	0.77	0.50	0.37	0.97	0.24	0.73	0.93	0.61	0.57	**0.89**	0.45
	DITTO	0.74	0.91	0.63	0.46	0.58	0.40	0.00	0.00	0.90	0.50	0.55	0.51	0.43	0.51	0.61

(mean recall 0.81, see Table 7) and then create 5 stratified randomized folds per ontology combination with 50% train, 25% validation, and 25% test data, using gold labels for training. For DITTO, we use RoBERTa [41] and 40 epochs. We run all of DITTO's different optimization techniques, and select the parameters which give the best result on the validation data. As shown in Table 6, all entity matchers have a much lower performance than VersaMatch, despite being trained with 50% gold training data. Also, in terms of F1, their performance is lower than all OM methods. Some of our suspected reasons are: OM datasets are more imbalanced than EM data, and even less positive samples are available (usually $< 0.1\%$ of combinations are matches). This phenomenon is most apparent in DITTO, which works better than the other EM for the large anatomy dataset, but cannot learn a precise model for AirTraffic, due to the small number of positive training data (32 matches overall, see Table 2). Another difficulty is that ontology properties are heterogeneous and not always aligned (e.g., 'label' in O_s might need to be compared with 'class name' in O_t). Last, entity matchers are agnostic to the ontology structure and semantics (e.g., they treat 'superclasses' just as another attribute, but are unaware of its meaning).

6.6 Blocking and Runtime

Blocking Impact. While blocking filters out highly unlikely matches, it also impacts the final predictions. We first evaluate the performance of our blocking method for all 4 datasets in terms of sample reduction and recall (see Table 7). Overall, our MinHash LSH based approach can reduce about 95% of the total class combinations but still achieves a high overall recall score of 0.90, speeding up the training and prediction process by nearly 20 times. The performance varies slightly across datasets due to the nature of MinHash LSH hashing, the information contained in the alignment profiles (i.e., the 'features' used in our blocking approach), and our developed threshold selection algorithm (see Section 4). E.g., as the AirTraffic dataset contains some exotic matches with little additional descriptive data in their properties, they are filtered out after our blocking phase despite being matches. To examine the prediction impact of blocking, we further evaluate VersaMatch without blocking (see Table 8). *Blocking does not hurt the final matching result, but instead results in a 2 point higher F1 score.* Investigating precision and recall results, we see that blocking achieves higher precision for 3 out of 4 datasets, while $VersaMatch_{NoBlocking}$ achieves only a slightly higher recall in 2 out of 4 datasets. Two observations could reason this result: 1) blocking helps to alleviate the class imbalance that affects the training; 2) blocking removes some exotic and challenging true matches, which can be considered outliers, so the training data is less noisy and more efficient for machine learning.

Runtime Analysis Table 8 also summarizes VersaMatch's execution times. We used an Intel Xeon Silver 4208 CPU, a NVIDIA GeForce RTX 2080 Ti GPU and 128GB of RAM. We perform 3 runs per combination. The time distribution for the different steps is: Alignment Profile Construction 10.32 %; Blocking 19.51 %; Label Generation 39.15 %; ML Model Training 29.75 % and Classification Ensemble 1.27 %. Especially for larger ontology pairs (anatomy), traditional ontology matchers are faster (< 2min). However, comparable supervised methods such as the tested entity matchers require an execution time much larger than ours (e.g., DITTO training alone required on average 1947s with 40 epochs across all datasets).

Table 7: Blocking efficiency compared to Magellan Blocker.

	$Comb_{total}$	$Comb_{versa}$	$Comb_{mag}$	$Recall_{versa}$	$Recall_{mag}$
AI4EU	181148	11290	10785	0.89	0.94
Conference	98688	7738	9729	0.91	0.83
AirTraffic	140910	3250	3510	0.81	0.72
Anatomy	9066176	465091	14600	0.98	0.76
Unweighted Average				**0.90**	**0.81**

7 RELATED WORK

Ontology & Schema Matching (SM & OM). LogMap [35] first computes a set of anchor mappings, based on nearly exact string similarities with a confidence value assigned. Next, LogMap repeatedly applies repair and discovery steps to refine and discover new mappings. Agreement Maker Light (AML) [24] employs iteratively different string-equivalence based matchers and background knowledge from existing knowledge bases such as WordNet. In each step,

Table 8: VersaMatch performance with and without blocking.

	$\text{VersaMatch}_{\text{Blocking}}$				$\text{VersaMatch}_{\text{NoBlocking}}$			
	F1	Prec.	Recall	Time (s)	F1	Prec.	Recall	Time (s)
AI4EU	0.81	0.84	0.79	112.8±1.2	0.77	0.71	0.83	328.2±2.7
Confer.	0.78	0.79	0.76	81.5±0.8	0.75	0.74	0.76	144.1±3.2
AirTraffic	0.60	0.68	0.53	98.1±2.9	0.58	0.60	0.56	254.2±3.5
Anatomy	0.92	0.96	0.88	663.5±32	0.92	0.96	0.88	10630±106
Mean	**0.78**	**0.82**	**0.74**	**239.0**	**0.76**	**0.75**	**0.76**	**2839.1**

AML aggregates values using a weighted average. YAM++ [51] can use either heuristics or supervised machine learning (Decision Tree, SVM) to classify between match and no-match. The matching output is then put through a similarity propagation process and constraint checking in a second step. OntoEmma [70] enriches ontologies with external natural language information, generates a feature vector, and uses it for supervised ML model training.

These works either rely on a set of carefully designed heuristics and repair steps or a set of labeled data to enable OM. Our results show that this leads to performance drops, especially for unknown datasets (e.g., AI4EU for heuristics based matchers). These drops can only be avoided by re-designing matching heuristics or providing new labeled data—both associated with high effort. In contrast, VersaMatch adapts weak-supervision to OM, including an enriched alignment profile that flexibly adapts to the specific dataset and the information available. Incomplete coverage of our labeling functions is compensated in our classification ensemble by a trained discriminative model. This achieves robust performance across different datasets without additional user effort.

Entity Matching (EM). EM aims to identify which records refer to the same real-world entity. Most recent EM works rely on some form of machine learning: Magellan [36] (traditional supervised learning) and DeepMatcher [49] (deep learning) entity matchers lay the grounds for BigGorilla [12], an open-source ecosystem for data preparation and integration. Similar to DeepMatcher, DeepER [21] develops a deep learning-based entity resolution system, including LSH based blocking. DeepER uses existing word embeddings and allows to tune the data representation (features) to the specific matching task. Several works address the label data bottleneck through crowd-sourcing & active learning [15, 29], pre-trained transformer-based language models [9, 40], generative similarity distribution modeling [72], transfer learning [76] and last weak-supervision [73].

As our experiments show, the assumptions made in entity matching solutions do not fully apply for OM (e.g., they assume a large amount of training data for both positive and negative samples, attributes in both datasets are assumed to be aligned, they are ignoring the structure and constraints in ontologies), leading to inferior results compared to VersaMatch and other ontology matchers. For our blocking approach, we are inspired by the LSH based blocking of DeepER. However, our method achieves dataset adaptation through our alignment profile abstraction, which contains all the information and variances needed for effective blocking for OM. Pandas [73] presents some initial efforts closest to ours in that it aims to adapt weak supervision to entity matching but focuses more on providing an integrated development environment (IDE) to ease the design of labeling functions.

Knowledge Graphs Alignment. A Knowledge Graph (KG) describes real-world entities and their interrelations in a graph. A KG can be seen as an instance of an ontology, often with a lower level of formalization [56]. As such, KG alignment is related to OM. There are two main paradigms in KG alignment. TransE [5] introduced the paradigm of translation-based embeddings, which uses relation triples and seed alignments to align KGs. Several other works improve upon TransE: MTransE [13] to align KGs in different languages and [74], which incorporates attribute information. The second paradigm is based on Graph Neural Networks (GNNs) and builds embeddings by passing messages between graph nodes, encoding each node's neighborhood information: [71] uses Graph Convolutional Networks (GCNs) to align KGs in different languages, utilizing not only structural information but also attributes information, [77] utilizes Graph Attention Networks (GATs) [66] to aggregates neighborhood information with the attention mechanism to automatically weight nodes with higher relevance.

Ontologies define the schema while KGs mostly contain instance data. This causes ontologies to be sparser than KGs and to usually not be fully connected, two characteristics that the works mentioned above depend on. Another limitation is the need for user-provided seed alignments for each pair of KGs, which is not an easy task when the user is not familiar with the data domain. With VersaMatch, we avoid this dependency through a flexible, weakly-supervised matching approach based on homogeneous alignment profiles, making our approach directly applicable to new datasets.

8 CONCLUSION AND FUTURE WORK

VersaMatch is to our knowledge the first to adapt weak-supervision to OM. To enable this adaption, we designed multiple building blocks from homogeneous alignment profiles and a MinHash based blocking method to a generic set of labeling functions and discriminative ML model features. To support generalization while retaining precision, we created a novel classification ensemble of weak supervision sources and a ML model, achieving higher accuracy and flexibility than any of these models on their own. VersaMatch achieves, on average, a 4-point higher F1 score than state-of-the-art matching systems across four representative datasets consisting of 26 ontology combinations and > 10M class pair combinations with a generic, core set of labeling functions, applicable to a wide range of datasets. For new, in-the-wild data, VersaMatch is even 9 points ahead of the next best matcher. Thanks to the alignment profile and labeling function abstractions, we achieve these results with a fraction of effort needed for traditional supervised ML.

In the future, we will extend VersaMatch to match other data sources such as relational or semi-structured data to ontologies. We further plan to (1) extend VersaMatch with more advanced graph embeddings (e.g., EmbDI [10], EP [27]) on top of initially sparsely connected ontologies to find further matches and (2) train the discriminative model with a more comprehensive feature vector and larger datasets (e.g., recently, Intermediate Training of BERT has shown to be effective for product matching [58]). A third direction for future work is the combination of weak supervision with active learning to avoid the bootstrapping problem of active learning.

REFERENCES

[1] David Aumueller, Hong-Hai Do, Sabine Massmann, and Erhard Rahm. 2005. Schema and ontology matching with COMA++. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. 906–908.

[2] Stephen H Bach, Daniel Rodriguez, Yintao Liu, Chong Luo, Haidong Shao, Cassandra Xia, Souvik Sen, Alex Ratner, Braden Hancock, Houman Alborzi, et al. 2019. Snorkel drybell: A case study in deploying weak supervision at industrial scale. In *Proceedings of the 2019 International Conference on Management of Data*. 362–375.

[3] Richard Bellman. 1961. Adaptive control processes: a guided tour princeton university press. *Princeton, New Jersey, USA* (1961), 96.

[4] SG Bird and Edward Loper. 2004. NLTK: the natural language toolkit. Association for Computational Linguistics.

[5] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems* 26 (2013).

[6] Leo Breiman. 1996. Bagging predictors. *Machine learning* 24, 2 (1996), 123–140.

[7] Eran Bringer, Abraham Israeli, Yoav Shoham, Alex Ratner, and Christopher Ré. 2019. Osprey: Weak supervision of imbalanced extraction problems without code. In *Proceedings of the 3rd International Workshop on Data Management for End-to-End Machine Learning*. 1–11.

[8] Andrei Z Broder. 2000. Identifying and filtering near-duplicate documents. In *Annual Symposium on Combinatorial Pattern Matching*. Springer, 1–10.

[9] Ursin Brunner and Kurt Stockinger. 2020. Entity matching with transformer architectures-a step forward in data integration. In *23rd International Conference on Extending Database Technology, Copenhagen, 30 March-2 April 2020*. OpenProceedings.

[10] Riccardo Cappuzzo, Paolo Papotti, and Saravanan Thirumuruganathan. 2020. Creating embeddings of heterogeneous relational datasets for data integration tasks. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1335–1349.

[11] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Céspedes, Steve Yuan, Chris Tar, et al. 2018. Universal sentence encoder. *arXiv preprint arXiv:1803.11175* (2018).

[12] Chen Chen, Behzad Golshan, Alon Y Halevy, Wang-Chiew Tan, and AnHai Doan. 2018. BigGorilla: An Open-Source Ecosystem for Data Preparation and Integration. *IEEE Data Eng. Bull.* 41, 2 (2018), 10–22.

[13] Muhao Chen, Yingtao Tian, Mohan Yang, and Carlo Zaniolo. 2016. Multilingual knowledge graph embeddings for cross-lingual knowledge alignment. *arXiv preprint arXiv:1611.03954* (2016).

[14] Xi Chen, Weiguo Xia, Ernesto Jiménez-Ruiz, and Valerie V Cross. 2014. Extending an ontology alignment system with BioPortal: a preliminary analysis.. In *International semantic web conference (posters & demos)*. Citeseer, 313–316.

[15] Sanjib Das, Paul Suganthan GC, AnHai Doan, Jeffrey F Naughton, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, Vijay Raghavendra, and Youngchoon Park. 2017. Falcon: Scaling up hands-off crowdsourced entity matching to build cloud services. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 1431–1446.

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[17] Hong-Hai Do and Erhard Rahm. 2002. COMA—a system for flexible combination of schema matching approaches. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*. Elsevier, 610–621.

[18] AnHai Doan, Jayant Madhavan, Robin Dhamankar, Pedro Domingos, and Alon Halevy. 2003. Learning to match ontologies on the semantic web. *The VLDB journal* 12, 4 (2003), 303–319.

[19] Xin Luna Dong. 2018. Challenges and innovations in building a product knowledge graph. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2869–2869.

[20] Jared A Dunnmon, Alexander J Ratner, Khaled Saab, Nishith Khandwala, Matthew Markert, Hersh Sagreiya, Roger Goldman, Christopher Lee-Messer, Matthew P Lungren, Daniel L Rubin, et al. 2020. Cross-modal data programming enables rapid medical machine learning. *Patterns* 1, 2 (2020), 100019.

[21] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq Joty, Mourad Ouzzani, and Nan Tang. 2018. Distributed representations of tuples for entity resolution. *Proceedings of the VLDB Endowment* 11, 11 (2018), 1454–1467.

[22] Jérôme Euzenat, Christian Meilicke, Heiner Stuckenschmidt, Pavel Shvaiko, and Cássia Trojahn. 2011. Ontology alignment evaluation initiative: six years of experience. In *Journal on data semantics XV*. Springer, 158–192.

[23] Jérôme Euzenat, Pavel Shvaiko, et al. 2007. *Ontology matching*. Vol. 18. Springer.

[24] Daniel Faria, Catia Pesquita, Emanuel Santos, Matteo Palmonari, Isabel F Cruz, and Francisco M Couto. 2013. The agreementmakerlight ontology matching system. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 527–541.

[25] Raul Castro Fernandez, Pranav Subramaniam, and Michael J Franklin. 2020. Data Market Platforms: Trading Data Assets to Solve Data Problems [Vision Paper]. *arXiv preprint arXiv:2002.01047* (2020).

[26] Jonathan Fuerst, Mauricio Fadel Argerich, and Bin Cheng. 2023. VersaMatch Data and Code. Retrieved 2023-02-15 from https://github.com/nec-research/VersaMatch

[27] Alberto García-Durán and Mathias Niepert. 2017. Learning Graph Representations with Embedding Propagation. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 5125–5136.

[28] Celia Garrido-Hidalgo, Jonathan Fürst, Bin Cheng, Luis Roda-Sanchez, Teresa Olivares, and Ernö Kovacs. 2022. Interlinking the Brick Schema with Building Domain Ontologies. In *Proceedings of the Twentieth ACM Conference on Embedded Networked Sensor Systems*. 1026–1030.

[29] Chaitanya Gokhale, Sanjib Das, AnHai Doan, Jeffrey F Naughton, Narasimhan Rampalli, Jude Shavlik, and Xiaojin Zhu. 2014. Corleone: Hands-off crowdsourcing for entity matching. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 601–612.

[30] Amelie Gyrard, Manas Gaur, Swati Padhee, Amit Sheth, and Mihaela Juganaru-Mathieu. 2018. Knowledge Extraction for the Web of Things (KE4WoT) WWW 2018 Challenge Summary. In *Companion Proceedings of the The Web Conference 2018*. 1935–1936.

[31] Amelie Gyrard, Antoine Zimmermann, and Amit Sheth. 2018. Building IoT-based applications for smart cities: How can ontology catalogs help? *IEEE Internet of Things Journal* 5, 5 (2018), 3978–3990.

[32] Melissa Haendel, Georgios Gkoutos, Suzanna Lewis, and Chris Mungall. 2009. Uberon: towards a comprehensive multi-species anatomy ontology. *Nature precedings* (2009), 1–1.

[33] Matthew Honnibal and Ines Montani. 2017. spaCy: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. (2017).

[34] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. 604–613.

[35] Ernesto Jiménez-Ruiz and Bernardo Cuenca Grau. 2011. Logmap: Logic-based and scalable ontology matching. In *International Semantic Web Conference*. Springer, 273–288.

[36] Pradap Konda, Sanjib Das, AnHai Doan, Adel Ardalan, Jeffrey R Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeff Naughton, Shishir Prasad, et al. 2016. Magellan: toward building entity matching management systems over data science stacks. *Proceedings of the VLDB Endowment* 9, 13 (2016), 1581–1584.

[37] Christos Koutras, George Siachamis, Andra Ionescu, Kyriakos Psarakis, Jerry Brons, Marios Fragkoulis, Christoph Lofi, Angela Bonifati, and Asterios Katsifodimos. 2021. Valentine: Evaluating Matching Techniques for Dataset Discovery. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 468–479.

[38] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. 2020. *Mining of massive data sets*. Cambridge university press.

[39] Omer Levy and Yoav Goldberg. 2014. Dependency-based word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. 302–308.

[40] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep Entity Matching with Pre-Trained Language Models. *Proc. VLDB Endow.* 14, 1 (Sept. 2020), 50–60. https://doi.org/10.14778/3421424.3421431

[41] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[42] Emilia López-Iñesta, Francisco Grimaldo, and Miguel Arevalillo-Herráez. 2017. Learning similarity scores by using a family of distance functions in multiple feature spaces. *International Journal of Pattern Recognition and Artificial Intelligence* 31, 08 (2017), 1750027.

[43] Jayant Madhavan, Philip A Bernstein, and Erhard Rahm. 2001. Generic schema matching with cupid. In *vldb*, Vol. 1. 49–58.

[44] Andrés Masegosa, Stephan Lorenzen, Christian Igel, and Yevgeny Seldin. 2020. Second order PAC-Bayesian bounds for the weighted majority vote. *Advances in Neural Information Processing Systems* 33 (2020), 5263–5273.

[45] Venkata Vamsikrishna Meduri, Lucian Popa, Prithviraj Sen, and Mohamed Sarwat. 2020. A comprehensive benchmark framework for active learning methods in entity matching. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1133–1147.

[46] Christian Meilicke, Heiner Stuckenschmidt, and Andrei Tamilin. 2007. Repairing ontology mappings. In *AAAI*, Vol. 3. 6.

[47] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. 2002. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings 18th international conference on data engineering*. IEEE, 117–128.

[48] Barzan Mozafari, Purna Sarkar, Michael Franklin, Michael Jordan, and Samuel Madden. 2014. Scaling up crowd-sourcing to very large datasets: a case for active learning. *Proceedings of the VLDB Endowment* 8, 2 (2014), 125–136.

[49] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep learning for entity matching: A design space exploration. In *Proceedings of the 2018 International Conference on Management of Data*. 19–34.

[50] Stephen Mussmann, Robin Jia, and Percy Liang. 2020. On the importance of adaptive data collection for extremely imbalanced pairwise tasks. *arXiv preprint arXiv:2010.05103* (2020).

[51] DuyHoa Ngo and Zohra Bellahsene. 2012. YAM++: a multi-strategy based approach for ontology matching task. In *International Conference on Knowledge Engineering and Knowledge Management*. Springer, 421–425.

[52] Mathias Niepert, Christian Meilicke, and Heiner Stuckenschmidt. 2010. A probabilistic-logical framework for ontology matching. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*.

[53] Mahda Noura, Amelie Gyrard, Sebastian Heil, and Martin Gaedke. 2019. Automatic Knowledge Extraction to Build Semantic Web of Things Applications. *IEEE Internet Things J.* 6, 5 (2019), 8447–8454.

[54] Ontology Alignment Evaluation Initiative. 2019. Conference track. Retrieved 2020-07-03 from http://oaei.ontologymatching.org/2019/conference

[55] George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. 2020. Blocking and filtering techniques for entity resolution: A survey. *ACM Computing Surveys (CSUR)* 53, 2 (2020), 1–42.

[56] Heiko Paulheim. 2017. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic web* 8, 3 (2017), 489–508.

[57] Ted Pedersen, Siddharth Patwardhan, Jason Michelizzi, et al. 2004. WordNet:: Similarity-Measuring the Relatedness of Concepts.. In *AAAI*, Vol. 4. 25–29.

[58] Ralph Peeters, Christian Bizer, and Goran Glavaš. 2020. Intermediate training of BERT for product matching. In *CEUR Workshop Proceedings*, Vol. 2726. RWTH, 1–2.

[59] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. 2019. Snorkel: Rapid training data creation with weak supervision. *The VLDB Journal* (2019), 1–22.

[60] Alexander J Ratner, Christopher M De Sa, Sen Wu, Daniel Selsam, and Christopher Ré. 2016. Data programming: Creating large training sets, quickly. *Advances in neural information processing systems* 29 (2016).

[61] David Reinsel, John Gantz, and John Rydning. 2017. Data age 2025: the evolution of data to life-critical don't focus on big data; focus on the data that's big. *IDC, Seagate, April* (2017).

[62] Richard M. Keller, NASA. 2018. The NASA Air Traffic Management Ontology (atmonto). Retrieved 2021-11-21 from https://data.nasa.gov/ontologies/atmonto/ATM

[63] Rebecca C Steorts, Samuel L Ventura, Mauricio Sadinle, and Stephen E Fienberg. 2014. A comparison of blocking methods for record linkage. In *International conference on privacy in statistical databases*. Springer, 253–268.

[64] Nguyen Huy Tien, Nguyen Minh Le, Yamasaki Tomohiro, and Izuha Tatsuya. 2019. Sentence modeling via multiple word embeddings and multi-level comparison for semantic textual similarity. *Information Processing & Management* 56, 6 (2019), 102090.

[65] AI4EU Trialog. 2021. Knowledge Extraction for the Web of Things (KE4WoT). Retrieved 2022-05-16 from https://www.ai4eu.eu/ke4wot

[66] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[67] Audun Vennesland, Richard M Keller, Christoph G Schuetz, Eduard Gringinger, and Bernd Neumayr. 2019. Matching Ontologies for Air Traffic Management: a Comparison and Reference Alignment of the AIRM and NASA ATM Ontologies.. In *OM@ ISWC*. 1–12.

[68] Vennesland, A., Neumayr, B., Schuetz, C. G., Savulov, A., Wilson, S., Gringinger, E., Gorman, J. 2017. AIRM-O – ATM Information Reference Model Ontology. Retrieved 2021-11-21 from https://airm-o.github.io/airm-o/

[69] Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Commun. ACM* 57, 10 (2014), 78–85.

[70] Lucy Wang, Chandra Bhagavatula, Mark Neumann, Kyle Lo, Chris Wilhelm, and Waleed Ammar. 2018. Ontology alignment in the biomedical domain using entity definitions and context. In *Proceedings of the BioNLP 2018 workshop*. 47–55.

[71] Zhichun Wang, Qingsong Lv, Xiaohan Lan, and Yu Zhang. 2018. Cross-lingual knowledge graph alignment via graph convolutional networks. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 349–357.

[72] Renzhi Wu, Sanya Chaba, Saurabh Sawlani, Xu Chu, and Saravanan Thirumuruganathan. 2020. Zeroer: Entity resolution using zero labeled examples. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1149–1164.

[73] Renzhi Wu, Prem Sakala, Peng Li, Xu Chu, and Yeye He. 2021. Demonstration of Panda: A Weakly Supervised Entity Matching System. *arXiv preprint arXiv:2106.10821* (2021).

[74] Kai Yang, Shaoqin Liu, Junfeng Zhao, Yasha Wang, and Bing Xie. 2020. Cotsae: Co-training of structure and attribute embeddings for entity alignment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 3025–3032.

[75] Meihui Zhang, Marios Hadjieleftheriou, Beng Chin Ooi, Cecilia M Procopiuc, and Divesh Srivastava. 2011. Automatic discovery of attributes in relational databases. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. 109–120.

[76] Chen Zhao and Yeye He. 2019. Auto-em: End-to-end fuzzy entity-matching using pre-trained deep models and transfer learning. In *The World Wide Web Conference*. 2413–2424.

[77] Renbo Zhu, Meng Ma, and Ping Wang. 2021. RAGA: Relation-Aware Graph Attention Networks for Global Entity Alignment.. In *PAKDD (1)*. Springer, 501–513.