Mining Entity Synonyms with Efficient Neural Set Generation

Jiaming Shen[♮], Ruiliang Lyu[†], Xiang Ren[‡], Michelle Vanni[◇], Brian Sadler[◇], Jiawei Han[♮]

[♮]Department of Computer Science, University of Illinois Urbana-Champaign, IL, USA
[†]Department of Electronic Engineering, Shanghai Jiao Tong University, China   [◇]U.S. Army Research Laboratory, MD, USA
[‡]Department of Computer Science, University of Southern California, CA, USA
[♮]{js2, hanj}@illinois.edu   [†]lvruiliang-ele@sjtu.edu.cn   [‡]xiangren@usc.edu
[◇]{michelle.t.vanni.civ, brian.m.sadler6.civ}@mail.mil

Abstract

Mining entity synonym sets (*i.e.*, sets of terms referring to the same entity) is an important task for many entity-leveraging applications. Previous work either rank terms based on their similarity to a given query term, or treats the problem as a two-phase task (*i.e.*, detecting synonymy pairs, followed by organizing these pairs into synonym sets). However, these approaches fail to model the *holistic* semantics of a set and suffer from the error propagation issue. Here we propose a new framework, named SynSetMine, that efficiently generates entity synonym sets from a given vocabulary, using example sets from external knowledge bases as distant supervision. SynSetMine consists of two novel modules: (1) a set-instance classifier that jointly learns how to represent a permutation invariant synonym set and whether to include a new instance (*i.e.*, a term) into the set, and (2) a set generation algorithm that enumerates the vocabulary only once and applies the learned set-instance classifier to detect all entity synonym sets in it. Experiments on three real datasets from different domains demonstrate both effectiveness and efficiency of SynSetMine for mining entity synonym sets.

Introduction

An *entity synonym set* is a set of terms (*i.e.*, words or phrases) that refer to the same real-world entity. For instance, {"*USA*", "*United States*", "*U.S.*"} is an entity synonym set as all terms in it refer to the same country. Entity synonym set discovery can benefit a wide range of applications such as web search (Cheng, Lauw, and Paparizos 2012), question answering (Zhou et al. 2013), and taxonomy construction (Anh, Kim, and Ng 2015). Take the query "*Emirates Airline U.S. to UAE*" as an example, understanding "*U.S.*" refers to "*United States*" and "*UAE*" stands for "*United Arab Emirates*" is crucial for an intelligent system to satisfy the user information need.

One line of work for entity synonym sets discovery takes a *ranking plus pruning* approach. Given a query term referring to one entity, it first ranks all candidate terms based on their probabilities of referring to the same entity and then prunes the rank list into an output set. By treating each term in a vocabulary as a query, this approach can finally output all the entity synonym sets in the vocabulary. A variety of features are extracted, including corpus-level statistics (Turney

2001), textual patterns (Nakashole, Weikum, and Suchanek 2012), or query contexts (Chakrabarti et al. 2012), from different data sources (*e.g.*, query log (Chaudhuri, Ganti, and Xin 2009), web table (He et al. 2016), and raw text corpus (Qu, Ren, and Han 2017)) to calculate the above probabilities. However, this approach treats each candidate term separately and computes its probability of referring to the query entity independently. As a result, it ignores relations among candidate terms which could have helped to improve the quality of discovered synonym sets. Furthermore, as shown in (Ren and Cheng 2015), the number of true synonyms varies a lot across different entities and thus converting a rank list into a set itself is a non-trivial problem and can be error-prone.

Another line of work divides the synonym set discovery problem into two sequential subtasks: (1) *synonymy detection* (*i.e.*, finding term pairs of synonymy relation), and (2) *synonymy organization* (*i.e.*, aggregating synonymous term pairs into synonym sets). Methods developed for synonymy detection leverage textual patterns (Wang and Hirst 2012) and distributional word representations (Shwartz and Dagan 2016) to train a classifier that predicts whether two terms hold the synonymy relation. Then, those predicted term pairs form a synonymy graph on which different graph clustering algorithms are applied (Hope and Keller 2013; Oliveira and Gomes 2014; Ustalov, Panchenko, and Biemann 2017). This approach is able to capture relations among candidate terms and returns all entity synonym sets in the vocabulary. However, these two-phase methods only use training data in their first phase and cannot leverage training signals in the second phase. Furthermore, the detected synonymous term pairs are usually fixed during synonymy organization and there is no feedback from the second phase to the first, which causes the error accumulation problem.

In this work, we propose a new framework, SynSetMine, which leverages existing synonym sets from a knowledge base as *distant supervision* and extracts more synonym sets not in knowledge bases from massive raw text corpus. Specifically, SynSetMine first applies an entity linker to map in-corpus text (*i.e.* entity mentions) to entities in the knowledge base. Then, it groups all mentions mapped to the same entity (with the same unique id) into an entity synonym set, which provides supervision signals. As these "training" synonym sets are created automatically and without any human effort, we refer to them as distant supervision.
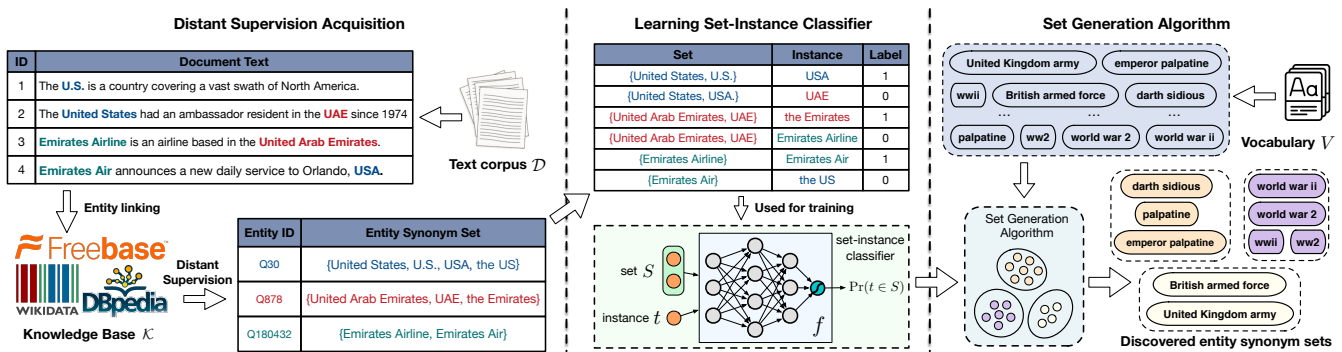
Figure 1: SynSetMine Framework Overview.

To effectively leverage distant supervision signals, SynSetMine consists of two novel modules. First, we train a set-instance classifier which jointly learns how to represent an entity synonym set and whether to include a new instance (*i.e.*, a term) into the set. This set-instance classifier can model a set holistically, instead of decomposing it into separated pairs. As a result, it effectively captures relations among candidate terms and directly leverages supervision signals from the set structure. Second, we design an efficient set generation algorithm that applies the learned set-instance classifier to discover new entity synonym sets. Given a vocabulary, this algorithm processes each term in it one at a time and maintains a pool of all detected sets. For each term in the vocabulary, the algorithm applies the set-instance classifier to determine whether and which previous detected set this term should reside in. If no matching set can be found, a new set is formed, consisting of this single term, and added into the pool of detected sets. As it only enumerates the vocabulary once to generate all synonym sets, this algorithm is efficient.

Contributions. This study makes three contributions: (1) a novel framework, SynSetMine, is proposed that leverages distant supervision for entity synonym set mining; (2) a set-instance classifier is designed to model entity synonym sets holistically and is integrated into a set generation algorithm to discover new synonym sets efficiently; and (3) extensive experiments conducted on three real-world datasets from different domains show the effectiveness of the method.

Problem Formulation

We first elaborate on some important concepts and then formulate the problem.

Entity Synonym Set. An entity synonym set is a set of terms (*i.e.*, words or phrases) that refer to the same real-world entity.

Knowledge Base. A knowledge base consists of many facts about a set of entities. In this work, we focus on one particular type of facts: entity synonym. For some entities, their synonyms are manually curated and stored in a knowledge base. The knowledge base provides such training signals to help discover more entity synonym sets.

Problem Definition. Given a text corpus $\mathcal{D}$, a vocabulary V (*i.e.*, a list of terms) derived from $\mathcal{D}$, and a knowledge base $\mathcal{K}$, the task of *mining entity synonym set* aims to discover all

entity synonym sets consisting of terms in V, based on the information extracted from $\mathcal{D}$ and $\mathcal{K}$.

The SynSetMine Framework

Our proposed SynSetMine framework consists of three main steps (Figure 1): (1) an entity linker is used to map in-corpus text (*i.e.*, entity mentions) to entities in the given knowledge base, which provides some training entity synonym sets as distant supervision; (2) a classifier is constructed to determine whether a term should be included into the synonym set, based on the above distant supervision; and (3) the learned classifier is integrated into a set generation algorithm which outputs all term clusters in the vocabulary as detected entity synonym sets.

Distant Supervision Acquisition

A knowledge base contains a collection of curated entities with their known synonyms. These entities can provide distant supervision signals to help discover more entity synonym sets that are not in the knowledge base from raw text corpus. To automatically acquire the distant supervision, we first apply an existing entity linker such as DBpedia Spotlight (Mendes et al. 2011) which directly maps in-corpus text (*i.e.*, entity mentions) to entities in the knowledge base. However, most entity linkers are not perfect and heavily rely on the string-level features which could introduce additional noise, as shown in (Shen et al. 2018b) and the example below. Therefore, we follow the same procedure in (Qu, Ren, and Han 2017) to reduce the linking errors. Specifically, for each entity mention and its linked entity, if the mention surface string is not in that entity's synonym set (in the knowledge base), we remove the link between them. Finally, we group all entity mentions that linked to the same entity as a training entity synonym set, and collect all synonym sets from the linked corpus as distant supervision.

Example 1. *Given a sentence "The U.S. Steel, located in Pennsylvania, USA, is a leading steel producer in America", an entity linker may first map "U.S. Steel", "USA" and "America" to the entity "*UNITED STATES*". Then, we find that the synonym set of entity "*UNITED STATES*", retrieved from knowledge base, does not contain the surface string "U.S. Steel". Therefore, we remove the link between them and*

250

group the remaining two entity mentions into an entity synonym set {"USA", "America"}.

Learning Set-Instance Classifier

After obtaining distant supervision, we train a set-instance classifier, denoted as $f(S, t)$, to determine whether a synonym set S should include an instance t (*i.e.*, a term).

Set-Instance Classifier Architecture. One key requirement of the set-instance classifier $f(S, t)$ is that its output should be invariant to the ordering of elements in set S. An intuitive way to achieve such *permutation invariance* property is to decompose the set-instance prediction into multiple instance-instance pair predictions, as shown in Figure 2. Each pair prediction decides whether two instances are synonyms, and all pair prediction results are finally aggregated into the set-instance prediction result. However, this method completely ignores the relations between elements in set S.

In this work, we are inspired by (Zaheer et al. 2017) and design a neural network architecture that directly learns to represent the permutation invariant set. The architecture of our set-instance classifier is shown in Figure 3. The bottom part of Figure 3 shows a set scorer $q(\cdot)$ which takes a set Z as input, and returns a quality score $q(Z)$ that measures how complete and coherent this set Z is. Given a synonym set S and an instance term t, our set-instance classifier $f(\cdot)$ first applies the set scorer to obtain the quality score of input set S (*i.e.*, $q(S)$). Then, we add the instance t into the set and apply the set scorer again to obtain the quality score of $S \cup \{t\}$. Finally, we calculate the difference between these two quality scores, and transform this score difference into the probability using a sigmoid unit as follows:

$$\Pr(t \in S) = f(S, t) = \phi\left(q(S \cup \{t\}) - q(S)\right), \quad (1)$$

where $\phi(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function.

Given a collection of m set-instance pairs $\{(S_i, t_i)|_{i=1}^m\}$ with their corresponding labels $\{y_i|_{i=1}^m\}$, we learn the set-instance classifier using the log-loss as follows:

$$\mathcal{L}(f) = \sum_{i=1}^m -y_i \log(f(S_i, t_i)) - (1 - y_i) \log(1 - f(S_i, t_i)),$$
$$(2)$$

where y_i equals to 1 if $t_i \in S_i$ and equals to 0 otherwise.

Note that if the set scorer $q(\cdot)$ is invariant to the ordering of elements in its input set, our set-instance classifier built on it will naturally satisfy the permutation invariance property. Following we first describe the set scorer architecture (the bottom part of Figure 3) and then discuss how to generate set-instance pairs from distant supervision.

Set Scorer Architecture. Given a set of terms $Z = \{z_1, \ldots, z_n\}$, the set scorer first passes each term z_i into an embedding layer and obtains its embedding $\mathbf{x_i}$. Then, an "embedding transformer" $\phi(\cdot)$ is applied to transform the raw embedding to a new term representation $\phi(\mathbf{x_i})$. After that, we sum all transformed term representations and obtain the "raw" set representation $\mathbf{v}(Z) = \sum_{i=1}^n \phi(\mathbf{x_i})$. Finally, we feed this set representation into the "post-transformer" which outputs the final set quality score. Since the summation operation is
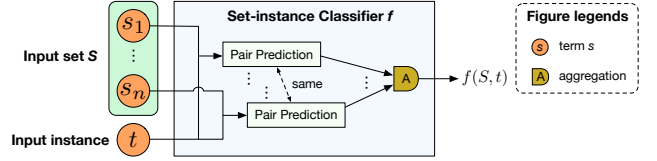

Figure 2: Aggregating instance-instance pair prediction results into set-instance prediction result.
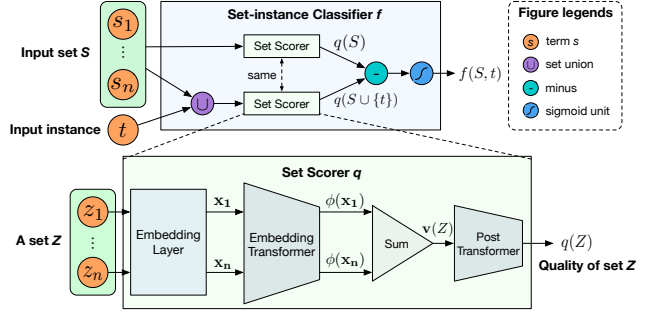

Figure 3: Architecture of our set-instance classifier.

commutative, the "raw" set representation $\mathbf{v}(Z)$ is invariant to the ordering of elements in Z and so is the set-scorer q.

In this work, we initialize the embedding layer using term embeddings pre-trained on the given corpus. We instantiate the "embedding transformer" using a fully connected neural network with two hidden layers. For the "post transformer", we construct it using another fully connected neural network with three hidden layers. We demonstrate the necessity of these two transformers and study how the size of hidden layers may influence model performance in later experiments.

Generation of Training Set-Instance Pairs. To train the set-instance classifier $f(S, t)$, we need to first generate a collection of training set-instance pairs from training synonym sets. For each entity synonym set ES, we randomly holdout one instance $t^{pos} \in ES$ and construct a positive set-instance sample (S^{pos}, t^{pos}) where $S^{pos} = ES \setminus \{t^{pos}\}$. Then, for each positive sample (S^{pos}, t^{pos}), we generate K negative samples by selecting K negative instances $t_i^{neg}|_{i=1}^K$ and pair each of them with S^{pos}. To generate each negative instance t_i^{neg}, we can either randomly choose a term from the vocabulary V (denoted as `complete-random` strategy); select a term that shares same token with some string in S^{pos} (denoted as `share-token` strategy), or combine these two methods (denoted as `mixture` strategy). We study how this negative sample size K and the sampling strategy influence the model performance in later experiments.

Set Generation Algorithm

We present our designed set generation algorithm for mining entity synonym set in Algorithm 1. This algorithm takes the above learned set-instance classifier, a vocabulary V, and a probability threshold θ as input, and clusters all terms in the vocabulary into entity synonym sets. Specifically, this algorithm enumerates the vocabulary V once and maintains

Algorithm 1: Set Generation Algorithm

Input: A set-instance classifier f; An input vocabulary $V = (s_1, s_2, \ldots, s_{|V|})$; A threshold $\theta \in [0, 1]$.

Output: m entity synonym sets $\mathcal{C} = [C_1, C_2, \ldots, C_m]$ where $C_i \subseteq V, \cup_{i=1}^{m} C_i = V,$ $C_i \cap C_j = \emptyset, \forall i \neq j.$

1   $\mathcal{C} \leftarrow [\{s_1\}]$; // initialize the first single-element cluster;
2   **for** i *from 2 to* $|V|$ **do**
3      $best\_score = 0$;
4      $best\_j = 1$;
5   **for** j *from 1 to* $|\mathcal{C}|$ **do**
6   **if** $f(C_j, s_i) > best\_score$ **then**
7            $best\_score \leftarrow f(C_j, s_i)$;
8            $best\_j \leftarrow j$;
9   **if** $best\_score > \theta$ **then**
10        $C_{best\_j}.add(s_i)$;
11   **else**
12        $\mathcal{C}.append(\{s_i\})$; //add a new cluster into the output;
13 Return $\mathcal{C}$;

---

a pool of all detected sets $\mathcal{C}$. For each term $s_i \in V$, it applies the set-instance classifier f to calculate the probability of adding this term into each detected set in $\mathcal{C}$, and finds the best set C_j that has the largest probability, If this probability value passes the threshold θ, we will add s_i into set C_j. Otherwise, we create a new set $\{s_i\}$ with this single term and add it into the set pool $\mathcal{C}$. The entire algorithm stops after one pass of the vocabulary and returns all detected sets in $\mathcal{C}$. Note that we do not need to specify the number of entity synonym sets in the vocabulary and our set generation algorithm will determine this value on its own. In this work, we simply set the probability threshold θ to be 0.5 and study its influence on clustering performance below.

Complexity Analysis. To compare with the clustering algorithm that requires the input cluster number, we suppose our algorithm will eventually return K clusters. Then, our algorithm applies the set-instance classifier at most $O(|V| \cdot K)$ times, where $|V|$ denotes the vocabulary size. Most computation efforts of set-instance classifier are the matrix multiplication in two transformers, which can be accelerated by GPU. As a comparison, the time complexity of k-means algorithm is $O(|V| \cdot K \cdot I)$ where I denotes the number of iterations, and for most supervised clustering methods (such as $SVM^{cluster}$ (Finley and Joachims 2005)) and two-phase methods (such as WATSET (Ustalov, Panchenko, and Biemann 2017)), their time complexity is $O(|V|^2)$ as they need to explicitly calculate all pairwise similarities. Finally, we emphasize that we can further parallelize lines 5-8 in Algorithm 1 by grouping all set-instance pairs $\{(C_j, s_i) | j = 1, \cdots, |\mathcal{C}|\}$ into a batch and applying the set-instance classifier only once. In practice, this strategy can significantly reduce the running time.

Experiments

In this section, we first describe our experimental setup, and then report the experimental results. Finally, we analyze each

Table 1: Datasets Statistics.

Dataset	Wiki	NYT	PubMed
#Documents	100,000	118,664	1,554,433
#Sentences	6,839,331	3,002,123	15,051,203
#Terms in *train*	8,731	2,600	72,627
#Synonym sets in *train*	4,359	1,273	28,600
#Terms in *test*	891	389	1,743
#Synonym sets in *test*	256	117	250

component of SynSetMine in more details and show several concrete case studies. Our model implementation is available at: https://github.com/mickeystroller/SynSetMine-pytorch.

Experimental Setup

Datasets. We evaluate SynSetMine on three public benchmark datasets used in (Qu, Ren, and Han 2017):

1. **Wiki** contains 100K articles in the Wikipedia. We use Freebase[1] as the knowledge base.

2. **NYT** includes about 119K news articles from 2013 New York Times. We use Freebase as the knowledge base.

3. **PubMed** contains around 1.5M paper abstracts from PubMed[2]. We select UMLS[3] as the knowledge base.

For Wiki and NYT datasets, DBpedia Spotlight[4] is used as the entity linker, and for PubMed, we apply PubTator[5] as the entity linker. After the linking step, we randomly sample a portion of linked entities as test entities and treat the remaining as training entities. Note that there is no overlapping between training vocabulary and testing vocabulary, which makes the evaluation more realistic but also more challenging. The statistics of these datasets are listed in Table 1. All datasets are available at: http://bit.ly/SynSetMine-dataset.

Compared Methods. We select the following algorithms to compare with our method.

1. **Kmeans**: An unsupervised clustering algorithm which takes term embedding as features and returns detected synonym sets as clusters. This algorithm requires a predefined cluster number K and we set its value to the oracle number of clusters for each dataset.

2. **Louvain (Blondel et al. 2008)**[6]: An unsupervised community detection algorithm which takes a graph as input and returns discovered graph communities. To apply this algorithm, we first construct a term graph where each node represents a term. Then, we calculate the cosine similarity between each pair of term embeddings, and if the similarity is larger than a threshold α, we will add an edge into the graph. We tune this threshold α on training set.

Table 2: Quantitative results of entity synonym set mining. All metrics are in percentage scale. We run all methods except L2C five times and report the averaged score with standard deviation. Due to the bad scalability of L2C, we have not obtain its results on PubMed dataset within 120h, and indicate this using "–" mark.

Method	Wiki			NYT			PubMed		
	ARI ($\pm$std)	FMI ($\pm$std)	NMI ($\pm$std)	ARI ($\pm$std)	FMI ($\pm$std)	NMI ($\pm$std)	ARI ($\pm$std)	FMI ($\pm$std)	NMI ($\pm$std)
Kmeans	34.35 ($\pm$1.06)	35.47 ($\pm$0.96)	86.98 ($\pm$0.27)	28.87 ($\pm$1.98)	30.85 ($\pm$1.76)	83.71 ($\pm$0.57)	48.68 ($\pm$1.93)	49.86 ($\pm$1.79)	88.08 ($\pm$0.45)
Louvain	42.25 ($\pm$0.00)	46.48 ($\pm$0.00)	92.58 ($\pm$0.00)	21.83 ($\pm$0.00)	30.58 ($\pm$0.00)	90.13 ($\pm$0.00)	46.58 ($\pm$0.00)	52.76 ($\pm$0.00)	90.46 ($\pm$0.00)
SetExpan+Louvain	44.78 ($\pm$0.28)	44.95 ($\pm$0.28)	92.12 ($\pm$0.02)	43.92 ($\pm$0.90)	44.31 ($\pm$0.93)	90.34 ($\pm$0.11)	58.91 ($\pm$0.08)	61.87($\pm$0.07)	92.23 ($\pm$0.15)
COP-Kmeans	38.80 ($\pm$0.51)	39.96 ($\pm$0.49)	90.31 ($\pm$0.15)	33.80 ($\pm$1.94)	34.57 ($\pm$2.06)	87.92 ($\pm$0.30)	49.12 ($\pm$0.85)	51.92 ($\pm$0.83)	89.91 ($\pm$0.15)
SVM+Louvain	6.03 ($\pm$0.73)	7.75 ($\pm$0.81)	25.43 ($\pm$0.13)	3.64 ($\pm$0.42)	5.10 ($\pm$0.39)	21.02 ($\pm$0.27)	7.76 ($\pm$0.96)	8.79 ($\pm$1.03)	31.08 ($\pm$0.34)
L2C	12.87 ($\pm$0.22)	19.90 ($\pm$0.24)	73.47 ($\pm$0.29)	12.71 ($\pm$0.89)	16.66 ($\pm$0.68)	70.23 ($\pm$1.20)	–	–	–
SynSetMine	**56.43** ($\pm$1.31)	**57.10** ($\pm$1.17)	**93.04** ($\pm$0.23)	**44.91** ($\pm$2.16)	**46.37** ($\pm$1.92)	**90.62** ($\pm$1.53)	**74.33** ($\pm$0.66)	**74.45** ($\pm$0.64)	**94.90** ($\pm$0.97)

3. **SetExpan (Shen et al. 2017)[7]+Louvain**: A two-phase unsupervised approach that first uses SetExpan (*i.e.*, a weakly-supervised set expansion algorithm) to find each term's k nearest neighbors in embedding space, and then construct a k-NN graph on which the above Louvain algorithm is applied. We tune the variable k on training set.

4. **COP-Kmeans (Wagstaff et al. 2001)[8]**: A semi-supervised variation of the Kmeans algorithm that can incorporate pairwise constraints (*e.g.*, two elements must or cannot be clustered together) and output clusters that satisfy all constraints. We convert training synonym sets into these pairwise constraints and set the oracle number of clusters K for each dataset.

5. **SVM[9]+Louvain**: A two-phase supervised approach which first uses a SVM for synonym pair prediction and then groups all predicted pairs into a graph where the Louvain algorithm is applied. The SVM is learned on training set.

6. **L2C (Hsu, Lv, and Kira 2018)[10]**: A supervised clustering method that learns a pairwise similarity prediction neural network and a constrained clustering network on training synonym sets, then applies the learned networks on test vocabulary to detect new entity synonym sets.

7. **SynSetMine**: Our proposed approach which trains a set-instance classifier and integrates it seamlessly into an efficient set generation algorithm.

Parameter Settings and Initial Term Embedding. For a fair comparison, we use the same 50-dimension term embedding, trained on each corpus, across all compared methods. We tune hyper-parameters in all (semi-)supervised algorithms using 5-fold cross validation on training set. For SynSetMine, we use a neural network with two hidden layers (of sizes 50, 250) as embedding transformer, and another neural network with three hidden layers (of sizes 250, 500, 250) as post transformer (c.f. Figure 3). We optimize our model using Adam with initial learning rate 0.001 and apply dropout technique with dropout rate 0.5. For the set generation algorithm, we set the probability threshold θ be 0.5. We will discuss the influence of these hyper-parameters later.

---

[7] https://github.com/mickeystroller/SetExpan

[8] https://github.com/Behrouz-Babaki/COP-Kmeans

[9] http://scikit-learn.org/stable/modules/svm.html

[10] https://github.com/GT-RIPL/L2C

Evaluation Metrics. As all compared methods output entity synonym sets in the form of clusters, we evaluate them using three standard clustering evaluation metrics.

- **ARI** measures the similarity of two cluster assignments. Given the ground truth cluster assignment $\mathcal{C}^*$ and model predicted cluster assignment $\mathcal{C}$, we use TP (TN) to denote the number of element pairs that are in the same (different) cluster(s) in both $\mathcal{C}^*$ and $\mathcal{C}$, respectively. We denote the total number of element pairs in $\mathcal{C}^*$ as N, and then calculate ARI as follows:

$$\text{ARI} = \frac{\text{RI} - \mathbb{E}(\text{RI})}{\max(\text{RI}) - \mathbb{E}(\text{RI})}, \quad \text{RI} = \frac{(TP + TN)}{N},$$

where $\mathbb{E}(\text{RI})$ is the expected RI of random assignments.

- **FMI** is another similarity measure of two cluster assignments. Besides the above TP, we use FP (FN) to denote the number of element pairs that belong to the same clusters in $\mathcal{C}^*$ ($\mathcal{C}$) but in different clusters in $\mathcal{C}$ ($\mathcal{C}^*$), respectively. Then, we calculate FMI as follows:

$$\text{FMI} = \frac{TP}{\sqrt{(TP + FP)(TP + FN)}}.$$

- **NMI** calculates the normalized mutual information between two cluster assignments. This metric is widely used in literature and its calculation details can be found in (Nguyen, Epps, and Bailey 2009).

Experimental Results

Clustering Performance. We first compare all methods in terms of the clustering performance. Results are shown in Table 2. We can see that overall SynSetMine outperforms baseline methods across all three datasets. The main disadvantage of unsupervised methods (Kmeans, Louvain, and SetExpan+Louvain) is that they cannot utilize supervision signals, which limits their performance when supervision is available. Compared with Kmeans, COP-Kmeans leverages additional supervision information from the training set and thus has an improvement in performance. However, the incorporation of supervision signals is not always straightforward. We find that the SVM+Louvain method fails to capture the synonymy relation, probably due to the limited expressive power of SVM. Also, the supervised clustering method L2C also does not work very well on synonym datasets regarding both efficiency and the quality of returned clusters. The
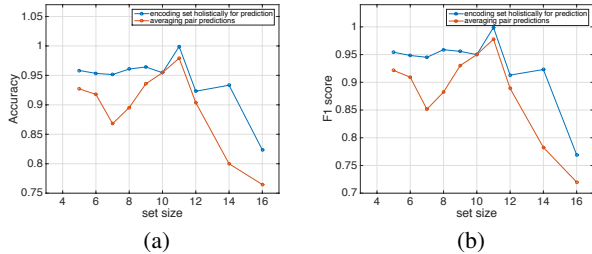
Figure 4: Evaluation of set-instance classifiers on PubMed dataset using (a) Accuracy and (b) F1 score.

Table 3: Efficiency analysis. Model and data loading time are excluded. GPU time are marked with $\star$. We use "–" to indicate that either there is no training time for unsupervised methods, or the model is too slow to train and thus we don't get one trained model for prediction.

Method	Training			Prediction		
	Wiki	NYT	PubMed	Wiki	NYT	PubMed
Kmeans	–	–	–	1.82s	0.88s	2.95s
Louvain	–	–	–	3.94s	20.59s	74.6s
SetExpan+Louvain	–	–	–	323s	120s	4143s
COP-KMeans	–	–	–	249s	37.94s	713s
SVM+Louvain	4.9m	37s	1.3h	29.21s	5.80s	101.32s
L2C	16.8h$\star$	30.7m$\star$	>120h$\star$	20.9m$\star$	56.6s$\star$	–
SynSetMine	48m$\star$	6.5m$\star$	7.5h$\star$	0.852s$\star$	0.348s$\star$	1.84s$\star$

reason is that during the training stage, L2C needs to calculate each example's class distribution, and this computation effort is proportional to the training cluster number which is in the scale of thousands[11]. Another major deficiency of both SVM+Louvain and L2C is that their learned model is based on pairwise similarity and does not have a holistic view of set. On the other hand, SynSetMine encodes synonym sets directly and therefore is able to capture set-level features beyond pairwise similarity.

Set-instance Pair Prediction Performance. To further analyze the importance of encoding set structure, we compare our method with the approach that averages instance-instance pair prediction results, as shown in Figure 2. Specifically, given a trained set-instance classifier $f(S,t)$, we first construct an instance-instance classifier $g(t_1, t_2)$ to determine whether two instances should be put into the same set. We let $g(t_1, t_2)$ to be the mean of $f(\{t_1\}, t_2)$ and $f(\{t_2\}, t_1)$. Then, for each set-instance pair (S, t), we apply the classifier g to obtain all instance pair prediction results (*i.e.*, $g(t', t), \forall t' \in S$) and average them into the final prediction.

For evaluation, we convert the testing synonym sets in PubMed dataset into a collection of 3486 set-instance pairs, among which 1743 pairs are positive. Results are shown in Figure 4. We can see clearly that the set-encoding based approach performs much better than the "averaging pair prediction" approach in terms of both accuracy and F1 score. This demonstrates the importance of capturing entity relations among the set and modeling the set structure holistically.

Efficiency Analysis. We implement our model based on the PyTorch library, same as the L2C baseline. We train two neural models (SynSetMine and L2C) on one Quadro P4000 GPU and run all the other methods on CPU. Results are shown in Table 3. Compared with the other supervised neural model L2C, our method can predict significantly faster. In addition, our set generation algorithm avoids explicitly computing all pairwise term similarities and thus is more efficient during the prediction stage.

---

[11] For comparison, L2C originally runs on MNIST and CIFAR-10 datasets where examples with the same class label is viewed as a cluster and the cluster number is of scale 10-100.

Model Analysis and Cases Studies

Following we conduct more experiments to analyze each component of our SynSetMine framework in more details and show some concrete case studies.

Effect of training set-instance pair generation strategy and negative sample size. In order to train the set-instance classifier, we need to first convert the training entity synonym sets (obtained from distant supervision) to a collection of set-instance pairs. We study how such conversion strategy and different negative sample sizes will affect the model performance on Wiki dataset. Results are shown in Figure 5a. First, we find that the complete-random strategy actually performs better than the share-token strategy. One possible explanation is that the complete-random strategy can generate more *diverse* negative samples and thus provide more supervision signals. Second, we observe that by combining the above two strategies and generating more mixed negative samples, our model can be further improved, which again may contribute to the *diversity* of negative samples.

Effects of different set-scorer architectures. To demonstrate the necessity of our model components, we first compare the current set-scorer architecture (c.f. Figure 3) with its two ablations. Specifically, we leave either the Embedding Transformer (ET) or the Post Transformer (PT) out, and test the performance of remaining models. As shown in Table 4, both ET and PT are essential to our model, and removing either of them will significantly damage our model's performance. Furthermore, the Post Transformer, which operates on the set representation, is particularly important to our model.

To further explore the effect of above two transformers, we train our model with different hidden layer sizes. Specifically, we use Both-X-Y to denote a set-scorer composed of an Embedding Transformer (of hidden sizes $\{50, X\}$) and a Post Transformer (of hidden sizes $\{X, Y, X\}$). Results are shown in Table 4. We discover that the performance first keeps increasing when the hidden layer size grows and then drops slightly. Also, the best model sizes are consistent for both Wiki and NYT datasets.

Effect of probability threshold θ. As shown in Algorithm 1, our set generation algorithm requires an input probability threshold θ to determine whether a term should be added into one of the already detected sets. Intuitively, the higher this threshold θ is, the more conservative our set gener-

Table 4: Analysis of set scorer architecture. "No-ET" means no Embedding Transformer (ET) module, "No-PT" means no Post Transformer (PT) module, and `Both-X-Y` stands for using both ET (of hidden sizes {50, X}) and PT (of hidden sizes {X, Y, X}).

Method	Wiki			NYT		
	ARI	FMI	NMI	ARI	FMI	NMI
No-ET	46.48	47.23	91.57	39.86	42.67	90.46
No-PT	1.50	0.50	89.95	0.82	1.70	82.20
`Both-100-200`	49.38	49.56	91.21	37.64	39.37	89.33
`Both-150-300`	53.06	53.27	91.96	43.20	44.08	89.57
`Both-200-400`	53.82	53.99	92.36	47.03	49.65	91.00
`Both-250-500`	**57.34**	**58.13**	**93.10**	**48.89**	**51.33**	**91.19**
`Both-300-600`	56.26	56.51	92.92	46.65	47.30	90.01
`Both-350-600`	55.93	56.10	92.69	47.40	48.37	90.14

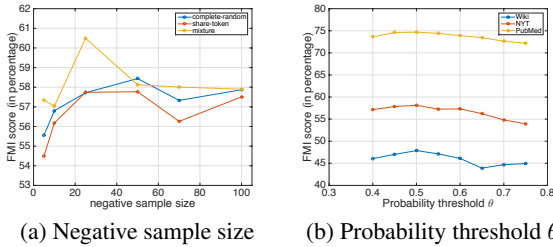(a) Negative sample size   (b) Probability threshold θ

Figure 5: Hyper-parameters analysis.

ation algorithm will be and more sets will be generated. In all the above experiments, we set the threshold θ to be 0.5. In this experiment, we intend to empirically study how this hyper-parameter will influence the clustering performance and how SynSetMine is sensitive to the choice of θ. Therefore, we run our set generation algorithm with fixed set-instance classifier and varied thresholds. The results are shown in Figure 5b. First, we notice that the performance of our clustering algorithm is insensitive to θ and a value within 0.4 and 0.6 is generally good for θ. Second, we find that setting θ to be 0.5 is robust and works well across all three datasets.

Case Studies. Table 5 presents some example outputs of SynSetMine. We can see that our method is able to detect different types of entity synonym sets across different domains. Then, in Table 6, we show a concrete case comparing our set-instance classifier with the approach that aggregates the instance-instance pair predictions. Clearly, the set encoding based method can detect more accurate entity synonyms.

Related Work

Entity Synonym Discovery. Most of previous efforts on entity synonym discovery focus on discovering entity synonyms from (semi-)structured data such as web table schemas (Cafarella et al. 2008) and query logs (Chaudhuri, Ganti, and Xin 2009; Ren and Cheng 2015). In this work, we aim to mine entity synonym sets directly from raw text corpus, which has a border application scope. Given a corpus, one can leverage co-occurrence statistics (Baroni and Bisi 2004), textual pattern (Nakashole, Weikum, and Suchanek 2012; Yahya et al. 2014;

Table 5: Example outputs on three datasets.

Dataset	Distant Supervision	Discovered Synonym Sets
Wiki	{"*londres*", "*london*"}	{"*gas*", "*gasoline*", "*petrol*"}
	{"*mushroom*", "*toadstool*"}	{"*roman fort*", "*castra*"}
NYT	{"*myanmar*", "*burma*"}	{ "*royal dutch shell plc*", "*royal dutch shell*", "*shell*" }
	{"*honda motor*", "*honda*"}	{"*chief executive officier*", "*ceo*" }
PubMed	{"*alzheimers disease*", "*Alzheimer's dementia*"}	{"*dna microarrays*", "*dna chip*", "*gene expression array*", "*dna array*"}

Table 6: Comparison of set-instance classifier with the approach that aggregates instance-instance pair predictions.

Method	Set-instance Classifier	Aggregate Pair Predictions
Synonym set	{"*u.k.*", "*britain*"}	{"*u.k.*", "*britain*"}
Ranked terms	"*uk*" "*united kingdom*" "*great britain*" "*elizabeth ii*"	"*uk*" "*indie*" "*united kingdom*" "*america*"

Shen et al. 2018a), distributional similarity (Pantel et al. 2009; Wang et al. 2015), or their combinations (Qu, Ren, and Han 2017) to extract synonyms. These methods, however, only find synonymous *term pairs* or *a rank list* of query entity's synonym, instead of entity synonym sets. Some work attempt to further cut-off the rank list into a set output (Ren and Cheng 2015) or to build a synonym graph and then apply graph clustering techniques to derive synonym sets (Oliveira and Gomes 2014; Ustalov, Panchenko, and Biemann 2017). However, these two-phase approaches suffer from the noise accumulation problem, and cannot directly model the entity set structure. Comparing to them, our approach can model entity synonym sets holistically and capture important relations among terms in the set. Finally, there exist some studies, such as finding lexical synonyms from dictionary (Ustalov et al. 2017) or attribute synonyms from query logs (He et al. 2016), but this work focuses on mining entity synonyms from raw text corpora.

Set Representation Learning. Our work is also related to set representation learning, which aims to construct permutation invariant representations of sets. PointNet (Qi et al. 2017) models points in a space as sets and uses multi-layer perceptrons and max pooling function to learn their representation. DeepSets (Zaheer et al. 2017) establishes the general form of permutation invariant functions and proposes to learn these functions using a deep neural network consisting of multi-layer perceptrons and aggregation functions. AutoEncSets (Hartford et al. 2018) further extends DeepSets by modeling the interactions across sets. In this work, in addition to just representing given sets, we go beyond one step and aim to *predict* sets (*i.e.*, entity synonym sets) from the vocabulary.

Conclusions

In this paper, we study how to mine entity synonym sets from raw text corpus. We propose a framework named SynSetMine which effectively leverages distant supervision from knowledge bases to learn a set-instance classifier and integrates it in an efficient set generation algorithm to detect

new entity synonym sets. Extensive experiments on three real-world datasets demonstrate both effectiveness and efficiency of our framework. In the future, we plan to further integrate set-instance classifier into the set generation algorithm and learn both of them in an end-to-end fashion.

Acknowledgements

References

Anh, T. L.; Kim, J.-j.; and Ng, S. K. 2015. Incorporating trustiness and collective synonym/contrastive evidence into taxonomy construction. In *EMNLP*.

Baroni, M., and Bisi, S. 2004. Using cooccurrence statistics and the web to discover synonyms in a technical language. In *LREC*.

Blondel, V. D.; Guillaume, J.-L.; Lambiotte, R.; and Lefebvre, E. 2008. Fast unfolding of communities in large networks.

Cafarella, M. J.; Halevy, A. Y.; Wang, D. Z.; Wu, E.; and Zhang, Y. 2008. Webtables: exploring the power of tables on the web. *PVLDB* 1:538–549.

Chakrabarti, K.; Chaudhuri, S.; Cheng, T.; and Xin, D. 2012. A framework for robust discovery of entity synonyms. In *KDD*.

Chaudhuri, S.; Ganti, V.; and Xin, D. 2009. Exploiting web search to generate synonyms for entities. In *WWW*.

Cheng, T.; Lauw, H. W.; and Paparizos, S. 2012. Entity synonyms for structured web search. *IEEE Transactions on Knowledge and Data Engineering* 24:1862–1875.

Finley, T., and Joachims, T. 2005. Supervised clustering with support vector machines. In *ICML*.

Hartford, J. S.; Graham, D. R.; Leyton-Brown, K.; and Ravanbakhsh, S. 2018. Deep models of interactions across sets. In *ICML*.

He, Y.; Chakrabarti, K.; Cheng, T.; and Tylenda, T. 2016. Automatic discovery of attribute synonyms using query logs and table corpora. In *WWW*.

Hope, D. R., and Keller, B. 2013. Maxmax: A graph-based soft clustering algorithm applied to word sense induction. In *CICLing*.

Hsu, Y.-C.; Lv, Z.; and Kira, Z. 2018. Learning to cluster in order to transfer across domains and tasks. In *ICLR*.

Mendes, P. N.; Jakob, M.; García-Silva, A.; and Bizer, C. 2011. Dbpedia spotlight: shedding light on the web of documents. In *I-SEMANTICS*.

Nakashole, N.; Weikum, G.; and Suchanek, F. M. 2012. Patty: A taxonomy of relational patterns with semantic types. In *EMNLP-CoNLL*.

Nguyen, X. V.; Epps, J.; and Bailey, J. 2009. Information theoretic measures for clusterings comparison: is a correction for chance necessary? In *ICML*.

Oliveira, H. G., and Gomes, P. 2014. Eco and onto.pt: a flexible approach for creating a portuguese wordnet automatically. *Language Resources and Evaluation* 48:373–393.

Pantel, P.; Crestan, E.; Borkovsky, A.; Popescu, A.-M.; and Vyas, V. 2009. Web-scale distributional similarity and entity set expansion. In *EMNLP*.

Qi, C. R.; Su, H.; Mo, K.; and Guibas, L. J. 2017. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*.

Qu, M.; Ren, X.; and Han, J. 2017. Automatic synonym discovery with knowledge bases. In *KDD*.

Ren, X., and Cheng, T. 2015. Synonym discovery for structured entities on heterogeneous graphs. In *WWW*.

Shen, J.; Wu, Z.; Lei, D.; Shang, J.; Ren, X.; and Han, J. 2017. Setexpan: Corpus-based set expansion via context feature selection and rank ensemble. In *ECML/PKDD*.

Shen, J.; Wu, Z.; Lei, D.; Zhang, C.; Ren, X.; Vanni, M. T.; Sadler, B. M.; and Han, J. 2018a. Hiexpan: Task-guided taxonomy construction by hierarchical tree expansion. In *KDD*.

Shen, J.; Xiao, J.; He, X.; Shang, J.; Sinha, S.; and Han, J. 2018b. Entity set search of scientific literature: An unsupervised ranking approach. In *SIGIR*.

Shwartz, V., and Dagan, I. 2016. Cogalex-v shared task: Lexnet - integrated path-based and distributional method for the identification of semantic relations. In *CogALex@COLING*.

Turney, P. D. 2001. Mining the web for synonyms: Pmi-ir versus lsa on toefl. In *ECML*.

Ustalov, D.; Chernoskutov, M.; Biemann, C.; and Panchenko, A. 2017. Fighting with the sparsity of synonymy dictionaries for automatic synset induction. In *AIST*.

Ustalov, D.; Panchenko, A.; and Biemann, C. 2017. Watset: Automatic induction of synsets from a graph of synonyms. In *ACL*.

Wagstaff, K.; Cardie, C.; Rogers, S.; and Schrödl, S. 2001. Constrained k-means clustering with background knowledge. In *ICML*.

Wang, T., and Hirst, G. 2012. Exploring patterns in dictionary definitions for synonym extraction. *Natural Language Engineering* 18:313–342.

Wang, H.; Gao, B.; Bian, J.; Tian, F.; and Liu, T.-Y. 2015. Solving verbal comprehension questions in iq test by knowledge-powered word embedding. *CoRR* abs/1505.07909.

Yahya, M.; Whang, S. E.; Gupta, R.; and Halevy, A. Y. 2014. Renoun: Fact extraction for nominal attributes. In *EMNLP*.

Zaheer, M.; Kottur, S.; Ravanbakhsh, S.; Póczos, B.; Salakhutdinov, R.; and Smola, A. J. 2017. Deep sets. In *NIPS*.

Zhou, G.; Liu, Y.; Liu, F.; Zeng, D.; and Zhao, J. 2013. Improving question retrieval in community question answering using world knowledge. In *IJCAI*.