Unsupervised Product Entity Resolution using Graph Representation Learning

Mozhdeh Gheini
gheini@isi.edu
USC Information Sciences Institute
Marina del Rey, California

Mayank Kejriwal
kejriwal@isi.edu
USC Information Sciences Institute
Marina del Rey, California

ABSTRACT

Entity Resolution (ER) is defined as the algorithmic problem of determining when two or more entities refer to the same underlying entity. In the e-commerce domain, the problem tends to arise when the same product is advertised on multiple platforms, but with slightly (or even very) different descriptions, prices and other attributes. While ER has been well-explored for domains like bibliographic citations, biomedicine, patient records and even restaurants, work on product ER is not as prominent. In this paper, we report preliminary results on an unsupervised product ER system that is simple and extremely lightweight. The system is able to reduce mean rank reductions on some challenging product ER benchmarks by 50-70% compared to a text-only benchmark by leveraging a combination of text and neural graph embeddings.

CCS CONCEPTS

• **Information systems → Clustering and classification**.

KEYWORDS

Entity Resolution, e-commerce, unsupervised, graph embeddings

ACM Reference Format:
Mozhdeh Gheini and Mayank Kejriwal. 2019. Unsupervised Product Entity Resolution using Graph Representation Learning. In *Proceedings of the SIGIR 2019 Workshop on eCommerce (SIGIR 2019 eCom),* 5 pages.

1 INTRODUCTION

With the proliferation of datasets on the Web, it is not uncommon to find the same entity in multiple datasets, all referred to in slightly different ways [15]. Entity Resolution (ER) is the algorithmic problem of determining when two or more entities refer to the same underlying entity [9], [6]. ER is a difficult problem that has been studied for more than 50 years, in fields as wide-ranging as biomedicine, movies, bibliographic citations and census records [28], [3]. Although human level performance has not been achieved, considerable progress has been made, across research areas as diverse as Semantic and World Wide Web, knowledge discovery and databases.

A particular domain-specific kind of ER that is extremely relevant to e-commerce is *product ER*. In the e-commerce domain, the ER problem tends to arise when the same product is advertised on multiple platforms, but with slightly (or even very) different descriptions, prices and other attributes. Product ER can be difficult both because of the wide variety of available products and e-commerce platforms, but also because of artifacts like missing and noisy data, especially when the data has been acquired in the first place by crawling and scraping webpages, followed by semi-automatic information extraction techniques like wrapper induction. The need is for an unsupervised, lightweight solution that, given a query entity, can rank candidate entities in other datasets and platforms in descending order of match probability. For an unsupervised ER to be truly viable, a matching entity, if one exists, generally needs to be in the top 5, on average. Text-based methods like tf-idf are not currently able to achieve this, however, on popular benchmarks, as we illustrate subsequently.

The core contribution in this paper is a Product Entity Resolution system that is unsupervised, lightweight and that uses a combination of text and graph-theoretic techniques to leverage not just a description of the product, but also the context of the dataset in which it occurs, to make an intelligent matching decision. The core intuition is to model each product entity as a node in a graph, with other nodes (e.g., prices, manufacturer) representing the information set of the entity. Since more than one entity can have the same price, manufacturer etc., these entities end up sharing context. Next, once the entities and their information sets have been represented in an appropriate graph-theoretic way, we embed the nodes in the graph using a well-known neural embedding algorithm like DeepWalk [22]. An embedding in this context is a dense, real-valued, low-dimensional vector. The goal is to embed the product entity nodes in such a way, without using any training data, as to ensure that nodes with similar embeddings (in a cosine similarity space) will have high likelihood of being matches.

Our guiding hypothesis in this paper is that neither a pure-text approach nor graph embedding, by itself, will yield optimal performance on the product ER problem. Rather, they will have to be combined in some way to yield a low mean rank. We conduct a set of experiments to show that this is indeed the case. By using two well-known and challenging benchmarks against various baselines, we demonstrate that both text and graph-theoretic approaches can together contribute to an average mean rank of less than 5, making such a system closer to viable deployment.

The rest of this paper is structured as follows. Section 2 covers some relevant related work, while Section 3 outlines our approach. We describe empirical results in Section 4, with Section 5 concluding the paper.

In: J. Degenhardt, S. Kallumadi, U. Porwal, A. Trotman (eds.):
Proceedings of the SIGIR 2019 eCom workshop, July 2019, Paris, France, published at http://ceur-ws.org

2 RELATED WORK

This paper draws on work in two broad areas of research, namely entity resolution and graph embeddings. Below, we individually cover pertinent aspects of these fields.

2.1 Entity Resolution

Entity Resolution (ER) is a problem that has been around for more than 50 years in the AI literature [7], with the earliest versions of the problem concerning the linking of patient records. More recently, both link prediction and entity resolution (ER) were both recognized as important steps in the overall *link mining* community about a decade ago [8]. In the Semantic Web community, instance matching [15], link discovery [20], [27] and class matching [25] are specific examples of such sparse edge-discovery tasks. Other applications include protein structure prediction (bioinformatics) [14], click-through rate prediction (advertising) [10], social media and network science [17], [24]. Good overviews of ER were provided both by Getoor and Machanavajjhala [9], and in the book by Christen [6].

A variety of recent papers have started to consider product datasets in the suite of benchmarks that they evaluate. For example, [16] considers both Amazon-Google Products and Abt-Buy, the two publicly available benchmarks that we also consider in this paper, in their evaluation. Performance on these datasets is considerably lower, even for supervised systems, compared to bibliographic datasets like DBLP and ACM, demonstrating that the problem deserves to be looked at in a domain-specific way for higher performance. Recently, Zhu et al. [30] was able to achieve better performance using a high-complexity k-partite graph clustering algorithm that was also unsupervised. However, they did not consider the problem from the purview of ranking or IR and their method had considerable runtime and modeling complexity. In contrast, our method is lightweight and directly uses IR metrics to evaluate.

Broadly speaking, a typical ER pipeline consists of two phases: *blocking* and *matching* [13]. Blocking is motivated by the fact that, in the worst case, one would have to do a pairwise comparison between every pair of entities to determine the matching pairs. In this vein, blocking refers to a set of 'divide-and-conquer'-style techniques for approximately grouping a set of entities so that pairwise comparisons (matching), which are more expensive, are only conducted on pairs of entities that *share* a block. Using an indexing function known as a *blocking scheme*, a blocking algorithm clusters approximately similarly entities into (possibly overlapping) clusters known as blocks. Only entities sharing a block are candidates for further analysis in the second *similarity* step. State-of-the-art similarity algorithms in various communities are now framed in terms of machine learning, typically as binary classification [1], [8].

An alternative to a batch blocking-matching workflow is to adopt an IR-centric workflow whereby a list of candidate entities needs to be ranked when given a query entity. We adopt this IR-centric workflow, since we recognize that, for an enterprise-grade system, manual perusal and tuning will be necessary unless the accuracy of the approach is very high. Hence, blocking does not apply. However, as we show in the experimental section, a bag-of-words model can be used for blocking-like pruning of entities that are not likely to be matches, leading to significantly higher performance when combined with the graph-theoretic approach.

2.2 Graph Embeddings

With the advent of neural networks, representation learning has emerged as an influential area of research in modern times. As input, neural networks need distributed representations of raw input that are able to capture the similarity between different data points.

For instance, in the Natural Language Processing (NLP) community, the use of word representations goes back to as early as 1986, as detailed in [12] and [23]. More recently, the benefits of using distributed representations for words to statistical language modeling, a fundamental task in NLP, was shown in [2]. The need for these representations have inspired and given rise to different word embeddings, such as word2vec [19], GloVe [21], and FastText [4], without which many current NLP systems would not have been able to perform as well.

Other forms of inputs, besides textual inputs, are no exception when it comes to the need for rich distributed vector space representations. Hence in the graph community, similar ideas have been adopted to embed graph nodes and edges. In this work, we use Deep Walk [22]. Like with word embeddings, numerous other graph embedding algorithms have been proposed including, but not limited to node2vec [11], GraphEmbed [29], and LINE [26]. Conceptually, these could be substituted for DeepWalk in this paper, an option we are exploring in future research.

3 APPROACH

Graphs are an important representation and modeling tool in many domains and for many problems, ranging from social media and networks applications to information science. With the advent of machine learning approaches and their conformity to vector inputs, we need to transform graphs into vector spaces to be able to have the best of both worlds: the richness of graph representations and effectiveness of machine learning approaches.

Graph embeddings are low-dimensional vector representations of graphs that try to *represent* the structure of the graph to guide analytical problem solving but without sacrificing efficiency [5]. As outlined and discussed in detail in [5], there exist different graph embedding methods and techniques depending on the information set of a node that needs to be preserved. For the product ER problem, we hypothesize that the second-order proximity of the nodes (which is the similarity between nodes' neighborhoods) is an important information set, considering that the nodes themselves (the product names) only contain limited information and are inherently ambiguous. In this paper, we adopt a random walk-based approach, where the goal is to encode a node's information set by collecting a set of random walks starting from that node.

More specifically, we use DeepWalk for this purpose [22], which has been made available on GitHub[1]. DeepWalk is heavily inspired by sequence embedding architectures like word2vec in the NLP community. In DeepWalk, each node of the graph roughly corresponds to a 'word', and hence, each random walk can be considered as a sentence in a language. Then, a neural model from the NLP field (in this case Skip-gram [18]) is used to capture second-order

[1] https://github.com/phanein/deepwalk

proximity information by ensuring that nodes that share a similar context (in this case, neighborhoods) would achieve vector embeddings that are close together in a cosine similarity space.

Before DeepWalk (or any graph embedding for that matter) can be employed, a product dataset, generally represented as key-value or semi-structured table with missing values, must be represented as a graph. To come up with a graph representation for a given dataset, we investigate two different strategies:

- **Simple Nodes:** In this method, we assign a node to each product entity ('row' in the table) as well as to each attribute (a 'cell' of that row e.g., the price of the product). Entity nodes are then linked to their corresponding to attribute nodes. So for instance, if 'Apple iPhone 6' is $600, the entity node corresponding to Apple iPhone 6 is linked to the attribute node representing the price value of $600. Note that if some other product similarly has a price of $600, it would also be linked to that node. For text attributes (such as description), we model a separate node for each unique token in the text value and link the entity node to all tokens occurring in its text attribute value (modeled as a bag of words). In essence, this setting can be seen as a collection of star graphs where entity nodes are the centers of local star graphs.
- **Aggregated Nodes:** This method is very similar to the previous one except that we treat prices specially by combining similar prices together and representing them with a single node. Specifically, in our experiments, we divide prices into bins of width $5 to achieve such a grouping. This is an example of domain-specific graph representation, since prices are clearly important when deciding whether to link entities. Although we have only considered one such grouping in this paper, other such groupings are also possible (and not just for prices), a possibility we are exploring in future research.

By way of example, consider the two product entities in Table 1. We now consider the graph representations for these sample records using the two strategies above. Figure 1 shows the Simple Nodes representation, and Figure 2 the Aggregated Nodes representation[2]. In both figures, entity nodes have dashed borders and attribute nodes have solid borders.

Note that, because we are only considering unsupervised Entity Resolution in this paper, there are no direct edges linking entities together (only second-order edges, where entities are linked via a token or price node etc.).

Once the product datasets have been modeled in this way, they are embedded using DeepWalk. The result is an embedding (a continuous, real-valued vector) for each node in the graph i.e. we get a vector for attributes, tokens and entities. Rather than directly compute matches, we take an IR-centric approach whereby, for each entity in the test set for which a match exists, we generate a ranking between that *source* entity and all other entities (*candidates*) by computing the cosine similarities between the vectors of the source entity and each candidate entity, followed by the ranking of the candidate entities by using the scores in descending order. Using the withheld gold standard, we know the rank of the 'true' entity matching the source entity, which is used to compute metrics like mean rank. In the next section, more details are provided on our experiments.

[2]In the figure, we assume a bin width of $10 for illustration purposes compared to the actual experimental bin width of $5.

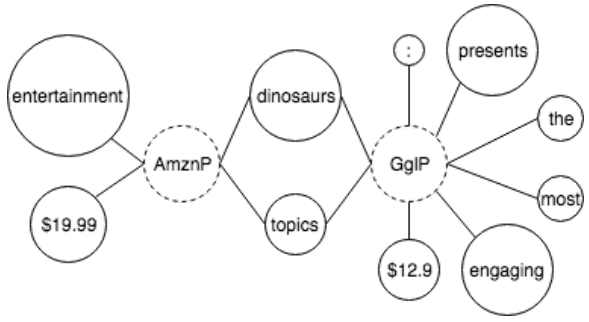

Figure 1: Graph representation of the product dataset in Table 1 using the Simple Nodes model.

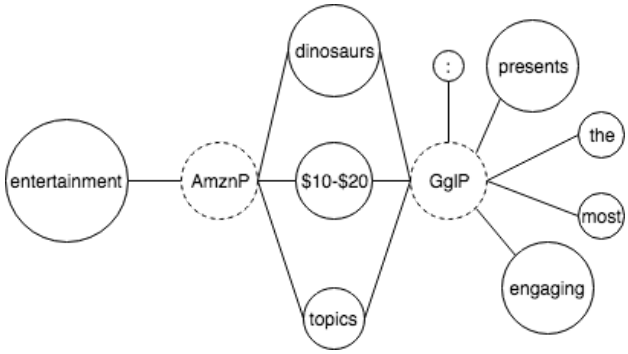

Figure 2: Graph representation of the product dataset in Table 1 using Aggregated Nodes model.

4 EXPERIMENTS

We conducted a preliminary set of experiments to explore answers to two research questions:

First, How well does the graph embedding (whether on the simple or aggregated nodes representation) do on the product ER problem compared to traditional text-based baselines? Is the graph embedding adequate by itself?

Second, Does the aggregated nodes representation help compared to the simple nodes representation?

We conduct a preliminary set of experiments to examine our approach on two benchmark datasets from the e-commerce domain: Amazon-Google Products and Abt-Buy Products [16]. Amazon-Google Products, as the name suggests, contains product entities from the online retailers *Amazon* and *Google*; similarly, Abt-Buy features products from two different sources. Besides the record 'ID', each record in both datasets has 'title', 'description', 'manufacturer', and 'price' fields. In the Amazon-Google dataset 1,363 Amazon records need to compared against 3,226 Google records, with the ground truth containing 1,300 matching pairs. In the Abt-Buy dataset, 1,081 Abt records need to be compared against 1,092 Buy records with the ground truth containing 1,097 matching pairs.

Dataset	Title	Description	Manufacturer	Price
AmznP	dinosaurs	–	topics entertainment	$19.99
GglP	topics presents:dinosaurs	the most engaging ...	–	$12.9

Table 1: One sample record each from the product collection in the Amazon-Google Products dataset (two separate tables but with the same schema). The description of the product from Google products have been trimmed due to space considerations. Empty fields, one of the reasons that makes Amazon-Google Products a challenging dataset, are marked with –. The discrepancy between the two records, which represent the same entity, is quite frequently observed across the dataset and is another factor that contributes to the difficulty of the task on this dataset.

4.1 Methods

tf-idf+Cosine Similarity. In countless prior empirical studies across IR, tf-idf has continued to work well as a baseline. We use it as a feature-engineered, word-centric baseline that we can compare our graph embedding based method to. Specifically, we tokenize the values of the 'description' fields, since the product descriptions are highly indicative features. Next, for each record, we obtain a tf-idf representation based on the tokens we obtained from the product descriptions. For each record in Dataset 1 (which could be Amazon or Google in Amazon-Google Products, and similarly, Abt or Buy in Abt-Buy), we rank all the records in Dataset 2 by using the cosine similarity between the records' tf-idf representations.

Graph Embedding+Cosine Similarity. This method is similar to the above, except that we use the cosine similarity between the graph embeddings of the entity nodes.

tf-idf Re-ranking. For the previous two baselines, we got the ranks from tf-idf and graph embedding methods separately. However, we consider a 'two-level' approach also, whereby we *re-rank* the top 100 entries in the original ranked list obtained by the tf-idf method using the graph embedding+cosine similarity scores. In this setting, the tf-idf serves as a 'pruning' mechanism (weeding out everything except the top 100 entries), similar to blocking algorithms in the ER literature, with graph embeddings having the final say.

4.2 Metrics

We use *mean rank* to evaluate our methods. For a given query record (an 'entity'), we consider the penalty to be the rank of the match to that query based on the gold truth in the ranking of the similarity-based method (whether based on graph embeddings, tf-idf or their combination). For the Abt-Buy benchmark, query records can be from either Abt or Buy (conversely, candidate entities that are ranked for the query would be from Buy or Abt), while for Amazon-Google Products we only tested using Google records as queries for the Simple Node Representation. For a given dataset, we average this penalty for all records to get the mean rank. For example, if A perfect entity resolver would therefore achieve a mean rank of 1.

4.3 Results and Discussion

In response to the first question, we conducted a preliminary experiment whereby on the Amazon-Google Products dataset, using Amazon entities as queries (and the simple nodes representation), we computed the mean rank for all three methods mentioned earlier. The tf-idf baseline was found to achieve average mean rank of 11.01, while the graph embedding achieved mean rank of 1719.49. Thus, by itself, the graph embedding was not a lot better than random, and clearly a lot less viable than the tf-idf based method. Next, we computed the mean rank for the re-ranking method, and found the average mean rank to be 3.51, a significant reduction from both of the other two methods. Hence, when used for re-ranking, the graph embeddings significantly improve the tf-idf ranking. Although simple, we believe that this is the first time a purely text-based method and a graph embedding have been used in conjunction for the product ER problem, and outperformed both individually.

Second, Table 2 tabulates the results for all four benchmark settings (both datasets, using queries from Abt/Buy and Google/Amazon respectively) with graph embeddings using the simple nodes representation. These results show that DeepWalk, although very poor by itself, can significantly improve results when applied over TFIDF to rerank its ranking. This improvement is consistent across datasets as Table 2 shows.

As mentioned earlier, we study two graph construction methods (simple nodes and aggregated nodes). Our second research question specifically asked whether the aggregated nodes representation can help improve performance compared to the simple nodes representation. As there are many empty price fields in the Abt-Buy dataset, and we focus on the price field for node aggregation, we only use the Amazon-Google dataset in this part. The results of re-ranking TFIDF when we use aggregated nodes graph representation is shown in Table 3.

Our results show that, compared to simple node representation, there is no significant or consistent gain when we aggregate price nodes. While results do improve (by 0.04) compared to the simple nodes representation (when using Google product entities as queries), there is a reversal when using Amazon product entities as queries. One reason for this may be due to how we build the graph with respect to text attributes. Recall that we tokenize the text in the description field and create a node for each token in the graph. An entity node is then linked to a token node if and only if that token appears in one of its text attributes. With this setting, for an entity node at the center of a star, most outgoing edges will be focused on text-based features and the price attribute only accounts for one outgoing edge.

One option that we're exploring in future work is to upsample the price nodes so that they have a stronger presence (via more occurrences) in the random walks. Another option that we're considering is using other aggregation strategies.

5 CONCLUSION

Product Entity Resolution is a difficult problem with the potential for high commercial impact, even with modest increases in metrics

Simple Nodes Representation			
Dataset	Query Record	tf-idf Mean Rank	tf-idf Re-ranked Mean Rank
Amazon-Google	Google	8.27	**4.18**
Amazon-Google	Amazon	11.01	**3.51**
Abt-Buy	Abt	9.39	**2.66**
Abt-Buy	Buy	11.88	**2.76**

Table 2: Experiment results across datasets.

Aggregated Nodes Representation			
Dataset	Query Record	tf-idf Mean Rank	tf-idf Re-ranked Mean Rank
Amazon-Google	Amazon	11.01	3.69
Amazon-Google	Google	8.27	4.14

Table 3: Experiment results with aggregated node graph representation.

like Mean Rank. In this paper, we illustrated a simple 'two-level' scheme that leverages both text and graph information to reduce the mean rank on some competitive benchmarks from more than 11 to less than 5. Although the aggregated nodes representation was found not to have an impact, this is likely due to the preliminary nature of our experiments, since we did not try many such aggregated representations. In the future, we will explore more options for aggregation, and will explore variants of the re-ranking scheme, along with exploring more options for graph embeddings (e.g., LINE, node2vec). We believe that the best approach will be a graph embedding specifically optimized for products rather than a generic approach like LINE or DeepWalk.

REFERENCES

[1] Mohammad Al Hasan, Vineet Chaoji, Saeed Salem, and Mohammed Zaki. 2006. Link prediction using supervised learning. In *SDMÂŋ06: Workshop on Link Analysis, Counter-terrorism and Security*.

[2] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of machine learning research* 3, Feb (2003), 1137–1155.

[3] Indrajit Bhattacharya and Lise Getoor. 2007. Collective entity resolution in relational data. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 1, 1 (2007), 5.

[4] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics* 5 (2017), 135–146.

[5] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. 2018. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering* 30, 9 (2018), 1616–1637.

[6] Peter Christen. 2012. *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection.* Springer.

[7] Ivan P Fellegi and Alan B Sunter. 1969. A theory for record linkage. *J. Amer. Statist. Assoc.* 64, 328 (1969), 1183–1210.

[8] Lise Getoor and Christopher P Diehl. 2005. Link mining: a survey. *ACM SIGKDD Explorations Newsletter* 7, 2 (2005), 3–12.

[9] Lise Getoor and Ashwin Machanavajjhala. 2012. Entity resolution: theory, practice & open challenges. *Proceedings of the VLDB Endowment* 5, 12 (2012), 2018–2019.

[10] Thore Graepel, Joaquin Q Candela, Thomas Borchert, and Ralf Herbrich. 2010. Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft's bing search engine. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. 13–20.

[11] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 855–864.

[12] Geoffrey E Hinton et al. 1986. Learning distributed representations of concepts. In *Proceedings of the eighth annual conference of the cognitive science society*, Vol. 1. Amherst, MA, 12.

[13] Mayank Kejriwal and Daniel P Miranker. 2013. An unsupervised algorithm for learning blocking schemes. In *Data Mining (ICDM), 2013 IEEE 13th International Conference on*. IEEE, 340–349.

[14] Lawrence A Kelley and Michael JE Sternberg. 2009. Protein structure prediction on the Web: a case study using the Phyre server. *Nature protocols* 4, 3 (2009), 363–371.

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

[16] Hanna Köpcke, Andreas Thor, and Erhard Rahm. 2010. Evaluation of entity resolution approaches on real-world match problems. *Proceedings of the VLDB Endowment* 3, 1-2 (2010), 484–493.

[17] Linyuan Lü and Tao Zhou. 2011. Link prediction in complex networks: A survey. *Physica A: Statistical Mechanics and its Applications* 390, 6 (2011), 1150–1170.

[18] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[19] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[20] Axel-Cyrille Ngonga Ngomo. 2011. A time-efficient hybrid approach to link discovery. *Ontology Matching* (2011), 1.

[21] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[22] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: Online Learning of Social Representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '14)*. ACM, New York, NY, USA, 701–710. https://doi.org/10.1145/2623330.2623732

[23] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. 1988. Learning representations by back-propagating errors. *Cognitive modeling* 5, 3 (1988), 1.

[24] Salvatore Scellato, Anastasios Noulas, and Cecilia Mascolo. 2011. Exploiting place features in link prediction on location-based social networks. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1046–1054.

[25] Pavel Shvaiko and Jérôme Euzenat. 2013. Ontology matching: state of the art and future challenges. *Knowledge and Data Engineering, IEEE Transactions on* 25, 1 (2013), 158–176.

[26] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*. International World Wide Web Conferences Steering Committee, 1067–1077.

[27] Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. 2009. Discovering and maintaining links on the web of data. In *The Semantic Web-ISWC 2009*. Springer, 650–665.

[28] William E Winkler and Yves Thibaudeau. 1991. *An application of the Fellegi-Sunter model of record linkage to the 1990 US decennial census.* Citeseer.

[29] Chao Zhang, Keyang Zhang, Quan Yuan, Haoruo Peng, Yu Zheng, Tim Hanratty, Shaowen Wang, and Jiawei Han. 2017. Regions, Periods, Activities: Uncovering Urban Dynamics via Cross-Modal Representation Learning. In *Proceedings of the 26th International Conference on World Wide Web (WWW '17)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 361–370. https://doi.org/10.1145/3038912.3052601

[30] Linhong Zhu, Majid Ghasemi-Gol, Pedro Szekely, Aram Galstyan, and Craig A Knoblock. 2016. Unsupervised entity resolution on multi-type graphs. In *International semantic web conference*. Springer, 649–667.