Weak-supervision, Word Embeddings, and Match Propagation: Combining Strategies for Real-World Entity Resolution

Fatima Idrees   Mariam Riaz   Madhu Kiran Reddy Thatikonda   Pawan Kumar   Raj Rajeshwari Prasad

Abstract—**In this short report we present the results of our entry to the Entity Resolution (ER) challenge, submitted for the ACM SIGMOD Programming Contest 2020. ER is a task of identifying and linking various different manifestations of a record across multiple sources representing the same real-world object. As a part of the solution we proposed a machine learning based approach in which we have developed a specific pipeline, and we specifically studied the contribution of the components this pipeline. Our approach is found to work well on the training and test data, but to not generalize well to unseen data. We observed the reasons for these performance problems, mostly based on false positives stemming from missing training examples. Our experiments allow us to conclude that our proposed method did not perform well, when compared with other solutions from the competition which achieved high f1 scores without using machine learning, but were instead very specific to given datasets.**

I. Introduction

With the rapidly increasing amount of data in the digital age, the significance of data integration has increased manifolds. Entity resolution is one of the key steps involved in data integration, identifying and linking different manifestations of records in different data sources to the same real-world entity. Entity resolution is also significant in terms of making informed decisions as it compares records across various datasets. However, the problem lies with the heterogeneous schema across various datasets. The data is usually noisy and also has missing attributes therefore to predict different entries as the same real world entity is a difficult task and needs to be explored for various possible solutions.

In our work we aim to develop a competitive solution to submit to a programming contest of a database conference [1]. The contest consisted of solving a real-world entity resolution challenge given a dataset of camera product descriptions from various e-commerce sites. We propose a pipeline with five methods to perform the given task, starting from weak supervision, embeddings for representation learning, deep hashing for pair formation, learning models and finally consistency check. One of the limitations to our solution was we did not have any benchmark method to evaluate the results of our proposed methods against.

ER is not a new problem, there is an active research going on in this field for almost two decades. With the latest developments in the fields of machine learning and deep learning paved a way for more generalized solutions. Methods like DeepER [1] and Deep Matcher are giving promising results but on the homogeneous data. There is no clear benchmark for ER methods on heterogeneous data.

Our contributions towards the task involve, taking a dataset which is representative of the similar products spread across various sites and test the results for our five proposed methods in the pipeline. By doing this we try to make the following findings:

- The significance of reduced feature components in determining the matched and non matched pairs.
- The effect of pre-trained embeddings and our own trained embeddings.
- The difference between learned and non-learned hashing and it's effect on the output.
- The various results via hyper-parameter tuning.
- The importance of consistency check in the such a task setting.

This report contains five sections, starting from introduction then we explain the core concepts, moving on to the proposed method, the results we achieved and finally the last section is about the conclusions and future work.

II. Core Concepts

- *Weak Supervision: Manually defining rules in an algorithm to separate products and non-products*
- *Embeddings: Converting the textual data to vectors of real numbers.*
- *Deep Hashing: Mapping entities with high similarity in the same set and dissimlar entities in different sets.*
- *Learning Model: To learn the pattern over training data and predict it over an unseen data.*
- *Consistency Check: Checking the transitive property over the matched pairs.*

III. Proposed Method

With our proposed methods we try to answer the following research questions:

- To what extent can filtering of dataset impacts the final f-measure?
- To what extent can different features combination proposed contribute to distinguishing matched and non matched data items on real world entity resolution? What

[1]http://www.inf.uniroma3.it/db/sigmod2020contest/index.html

is the impact of pretrained embeddings, embeddings trained on our own dataset, our hand-crafted features?

- To what extent for high dimensional blocking does learned hashing improve the coverage and computation over non-learned hashing (in a real world dataset assuming competitive features)?
- How much does tuning of the classifier contribute to improving the f1 score on a real world entity resolution dataset assuming blocking and set selection of features?
- How much can a consistency propagation algorithm proposed by us contribute to improve f1 score over end to end pipeline for entity resolution of real world dataset?

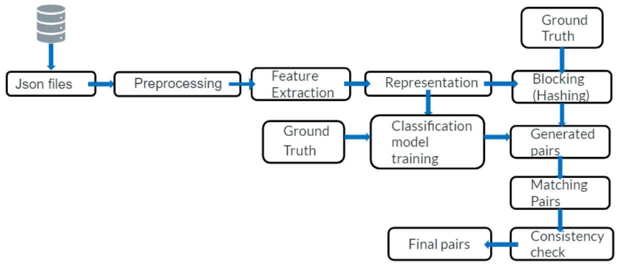


Fig. 1: *prototype*

Figure 1 shows the pipeline of our method. The dataset we used is a real world camera product specifications dataset. The specifications of the cameras are given in json format. Each specification is a key, value pair where key is the attribute name and value is the attribute value. For example: ”Manufacturer” : ”Nikon”. There are total 29,788 specifications of different camera products extracted from different web pages across 24 e-commerce websites. The task is to identify all the pairs of product specifications which represents the same real-world product.

The main challenges of the dataset are the heterogeneity and schema mismatch. The schema of the specifications are different in different websites and also within website in some cases, for example websites like ebay and alibaba. Noise is the another main challenge there are products which are not cameras. Class imbalance is the biggest challenge for the training data as there are so many non-matching pairs. We explain the pipeline(figure 1) step by step in the following subsections.

A. Data Cleaning and Feature Extraction

As we discussed earlier the main challenge of the dataset is schema mismatch. Different json files has different schema even with in the same website. Our first task is to clean the dataset and extract the common features. Page Title is the only common key in all json files. So, we first extracted the page-title from all the json files. After some basic exploratory analysis we came to a conclusion that page-title has a lot of information which can be used to identify the brand and model of a product. For example, “canon eos 7d mark ii black slr digital camera body only (20.2 mp, cf/sd card slot)”

After extracting the page-title next main challenge is to filter out the non-camera products. There are many products which are not cameras. Camera pouches, leather camera bags, portables cases, etc are a few to name which are not camera products. We found these non-products mainly in alibaba website. To make sure the non-products don’t influence the results. We used Snorkeling [2] (a labelling tool by Stanford) for labelling the entities as products and non-products. We defined a set of 9 rules that would return 0 if the entity is a non-product, return 1 if the entity is a product and -1 for abstain. The entity is labelled based on supervised majority voting, as the defined rules are made via human in the loop process so they are not always accurate and thus the whole process is termed as weak supervision. In 29,788 items there are 3108 non-camera products. So, after filtering out these non-camera products we have 26,680 items which are cameras.

Brand and model are the next two most common keys in the json files. Brand is present in 15,739 json files and model is present in 14,748 json files. These two are very important fields as they both together will tell to which real world product the item belongs to. We tried to extract the brand and model for all the items. For the files which don’t have brand and model key we tried to extract them from page-title. We used different regular expressions for different websites. For many websites there is a common pattern for the page-titles. So it became easy to extract brand and model for different websites separately. For the items which we cannot extract brand and model, mainly in alibaba and ebay websites because of so many inconsistencies, we used nobrand and nomodel instead of null values for those items. There are total 2,512 items which we cannot extract brand and 2,879 items which we cannot extract model. We explore the impact of different combinations of features on final f-measure in the result section.

We cleaned the noise from the page-titles. There are website names, and some stop words in the page-titles. We tried to clear some inconsistencies in brand names for example, fuji to fujifilm, cannon to canon, etc.

B. Word Embeddings

After extracting all the page-titles next step is the general representation. We represented page-titles as vectors by using fasttext [3]. Each page-title is represented as a 300 dimensional vector. In fasttext we used two types of models. One model is trained using the words from our page-titles. And the other is using pre-trained fasttext model. The model which we trained using our words. We used two types of models CBOW and skipgram. The impact of different embedding on the final f-measure is explored in the result section.

The brand and model features are represented using the binary encoding. Each brand and model has a unique representation.

[2]https://www.snorkel.org

C. Blocking

Locality Sensitive Hashing (LSH) is a technique to efficiently map entities with high similarity in the same set and dissimilar in different set. If $a! \sim b \rightarrow a \in p1\, and\, b \in p2,\ p1\, and\, p2 \subseteq P\ : p1 \neq p2$.
The method is evaluated based on two parameters. (i) Computation. (ii) Coverage. The goal of designer should be to maximize coverage and minimize computation.

The concept of blocking prevents the heavy computation by sampling similar items together. This allows formulation of pairs only within the buckets. The more the number of buckets the less is the computation. This is due to the reason that comparisons in the dataset is strictly limited to the bucket.

Buckets(Size)	**Pair Counts**	**Computation**
1(10)	45	100%
2(55)	20	50%
3(3,3,4)	6	34%
5(2,2,2,2,2)	5	20

TABLE I: *Impact of Hashing*

The techniques used for Blocking can be subdivided into two categories.(i) Data Dependent hashing. (ii) Data independent Hashing

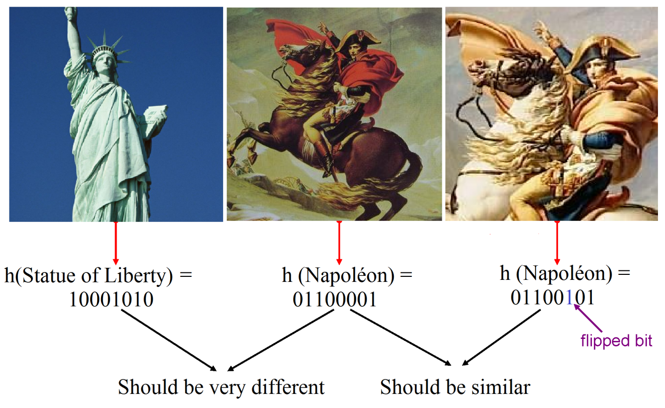


Fig. 2: *Statue of Liberty (Left) is dissimilar to Napoleon's painting (Centre and Right) and hence they have very different encoding.*

Let's suppose, we have 10 items in a dataset D. The pairs count decreases with increase in bucket size and so does computation.As shown in Table I.

1) Data Independent Hashing: Super-Bit LSH (SBLSH) formulates the relation between entities using Cosine similarity.
$similarity = (cos(\Theta)) = (A.B)/\left\|A\right\| \left\|B\right\|$

SBLSH Converts random projection vectors to orthogonal vectors (each consisting of L items, buckets). The resulting bits of these independent samples are then grouped together as N-Super Bit where N is the Super-Bit depth(stages).

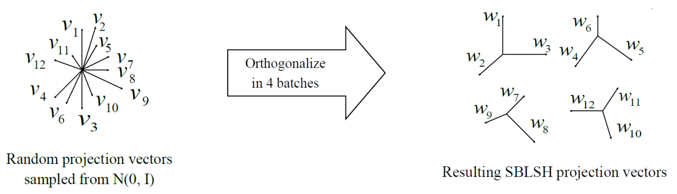

v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11} v_{12}
Random projection vectors sampled from N(0, I)
Orthogonalize in 4 batches
w_1 w_2 w_3 w_4 w_5 w_6 w_7 w_8 w_9 w_{10} w_{11} w_{12}
Resulting SBLSH projection vectors

Fig. 3: *Super-Bit Orthogonal Projection Vectors of Random Projection Vectors from Normal Distribution.*

2) Data-Dependent Hashing: We used a extension of pairwise deep hashing technique. We call it 'Triplet Trained Deep supervised Hashing' (TTDSH). In this method, instead of pair, triplets are passed as an input. With two entities matching to each other and the third as a non-match. The inputs are part of the ground-truth thus making it a supervised method.

The method was implemented using a Multi-layer Perceptron architecture with hyper-parameters categorized into two categories. (i) Hashing formulation – Code length and similarity weight. (ii) Network Level – Architecture, Optimizer, Learning rate, Number of iterations, Number of epochs, Activation functions, Dropouts.

The architecture comprises of a fully Connected Network with inputs as a triplet.The input share weights with the 1st hidden layer. The codes are generated after the 2nd hidden layer and the output is fed to a loss function. Finally the loss is used to back propagate the error and learn the weights.

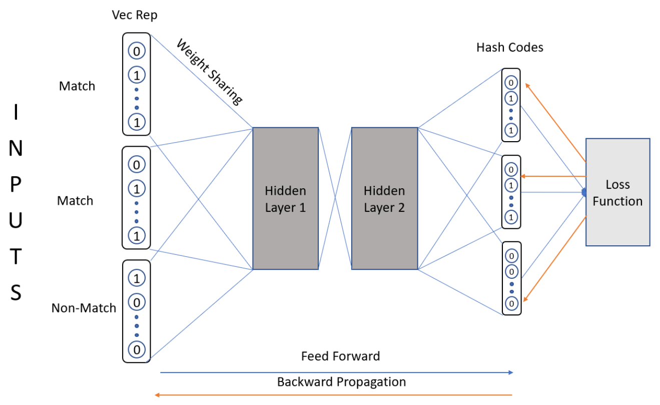


Fig. 4: *Neural Network Architecture for TTDSH.*

D. Classification and predictions

We tried two machine learning models for the classification part. We used basic Decision Tree [5] classifier as the baseline and compared it's performance with the XGBOOST [4]. XGBoost(eXtreme Gradient Boosting) is an open source decision-tree based machine learning algorithm which uses gradient boosting framework. It is a state-of-the-art boosting algorithm for semi-structured data. Gradient boosting algorithms are a type of ensemble methods, after each iteration method fits the new model to new residuals of the previous prediction and minimizes the loss during the new prediction. When updating the error they use gradient descent method. XGBoost is a more regularized model than other gradient boosting algorithms which gives better performance by controlling the over-fitting. [4]

In the following, we explain the steps for data preparation and training of the models.

- Data Preparation
  - For each pair in the ground truth, We take the absolute difference of the page-title vectors of left-spec-id and right-spec-id.
  - For the features brand and model, we do the AND operation on the binary encodes. i.e if the left-spec-id and right-spec-id is of same brand then the value is '1' else the value is '0'. Similarly for the model.
  - After these two steps, we have a 302 dimensions for each pair of the ground truth. 300 dimensions from the absolute difference of the page-title vectors and 2 dimensions from the brand and model.
- Training the model
  - From the entire ground truth we divided 70 percent as training data and 30 percent as test data.
  - We trained both models on the training data. We did hyper-parameter tuning on the XGBOOST.
- Prediction
  - The trained model is used to predict the pairs generated from the hash codes.
  - Same steps of the data preparation are applied to the generated pairs.
  - Each pair is predicted if it is a match or not a match.

IV. Results and discussion

The evaluation method we used is f-measure. We evaluate our models on two datasets. One is the test dataset which is a subset of ground truth and the other is the matches generated from complete dataset. We divided this section into subsections based on our research questions and discussed them separately.

A. *Weak Supervision*

After filtering out the non-products from the dataset, we can see in the Table II that F-measure is increased both on the test dataset and also on the complete dataset. This is understandable because after filtering out the non-products there are less number of false positives. So there is a slight increase in the f-measure. But when we compare the true-positives the number is almost the same.

Filtering	**F-measure on test data**	**F-measure on complete data**
Without filtering	0.889	0.202
With filtering	0.942	0.236

TABLE II: *Filtered vs non-filtered*

B. *Impact of different features combination on distinguishing matches and non-matches pairs:*

Table III shows that when we used the three features together there is a significant increase of F-measure both on the testing data and on the complete data. If two page-titles have good number of common words their vector representations are tend to be similar. This might increase the false positives. These false positives are decreased when we used brand and model along with the page-titles.

Feature Combination	**F-measure on test data**	**F-measure on complete data**
Page-title	0.8772	0.1410
Page-title and brand	0.898	0.1452
Page-title , brand and model	0.9421	0.236

TABLE III: *Impact of different features*

C. *Impact of different embeddings on f-measure:*

In the word embeddings section, we discussed that we trained embeddings using two different models. For the model trained on our data, we trained using both CBOW and skip-gram models. We evaluate these embeddings using cosine similarity values and also the final f-measure. In figure 5 we see the histogram plots of cosine similarity between the left-spec-id and the right-spec-id of the ground truth pairs. Even though the model trained with CBOW is giving low cosine values to non-matches and high cosine values to matches, it's f-measure on the complete data is less than the rest of the two models. Skipgram gave relatively high cosine values to non-matches, but it also differentiated matches and non-matches well compared to pre-trained model. Pre-trained embeddings cosine similarity values are almost similar to matches and non-matches. But it's performance is better than CBOW on test data and ground truth.

Embedding	**F-measure on test data**	**F-measure on complete data**
CBOW	0.92	0.179
SkipGram	0.942	0.236
Pretrained	0.921	0.2013

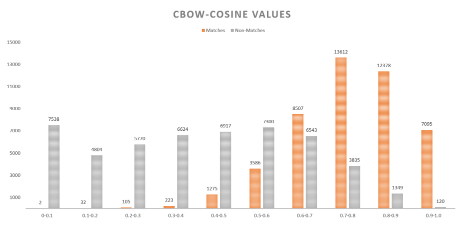
TABLE IV: *Impact of embeddings*

D. *Tuning of L2H and learned versus unlearned hashing:*

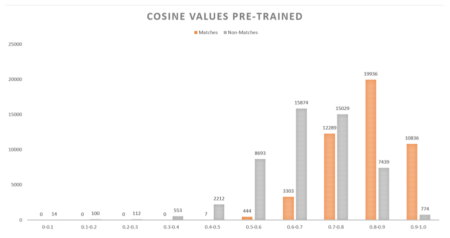
1) Computation: The task of establishing the pairs (a,b), where $(a,\, b \in D)$ and $a \neq b$ is computationally $O(N(N-1)/2$. Since we have 26,679 product unique spec ids, we would have 355 million possible pairs. This would result in very high computational effort.

$Computation = 100 * (\sum_{i=1}^{n}(b^2))/N^2$
$b = bucketsize, N = number\,of\,ids$

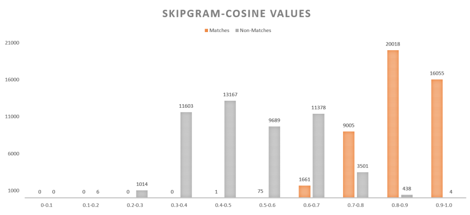
2) Coverage: To reduce the computation, similar entities are mapped in their respective buckets. This might sometimes lead to a bucket which might not contain all the similar entities depending upon the similarity threshold. This results in reduced number of pairs such that p'¡p where p' are number



(a) *CBOW*

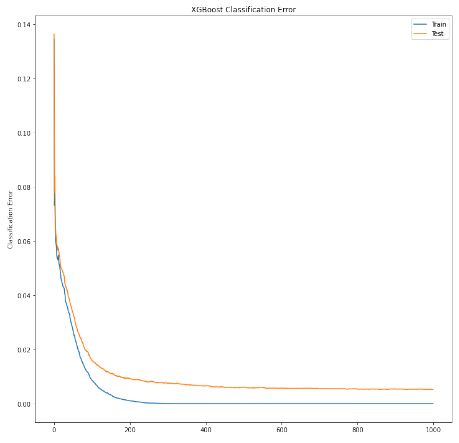


(b) *Pre-trained*

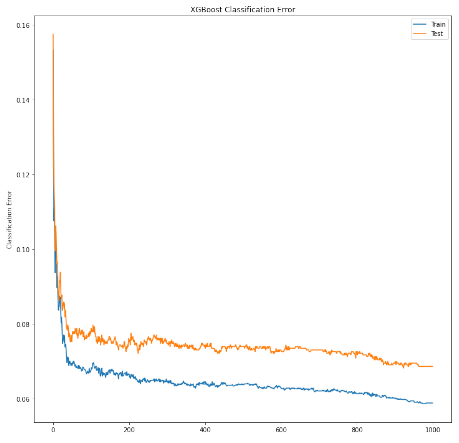


(c) *skipgram*

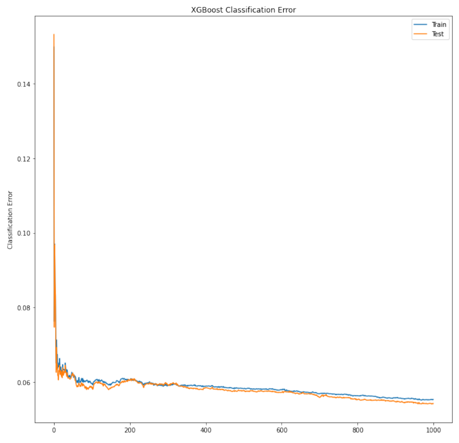
Fig. 5: *Cosine similarity of different embeddings*



(a) *learning rate=0.1,max depth=4*



(b) *learning rate=0.001,max depth=4*



(c) *learning rate=0.001,max depth=5*

Fig. 6: *XGBOOST classification errors*

of pairs after performing blocking and p is number of pairs without blocking

The cost of minimizing the computation can be evaluated by Coverage. It is the estimation of the ground truth that can be covered with the corresponding number of buckets and the number of entities in each bucket.

Algorithm 1: Coverage

```
1 $goodHash, counter = 0, pairs = [matchpairs]$;
2 for $(k1)$ in $groundTruth$ do
3     $counter+ = 1$;
4     if $k1$ in $pairs$ then
5         $goodHash+ = 1$;
6     end
7 end
8 $coverage = 100 * (goodHash/counter)$
```

E. Tuning of classifier:

We compare the performance of XGBOOST with the Decision Tree. In table VI , we can see that XGBOOST performed better than Decision Tree on both test data and on the complete data. On the complete data, Decision Tree predicted 82,080 pairs as matches and XGBOOST predicted 88,511 pairs as matches.

Tuning the XGBOOST is another biggest challenge. It is very prone to overfitting. In the figure 2 we can see the classification error for different configurations. A small change in max depth caused a significant change in the test and train error.

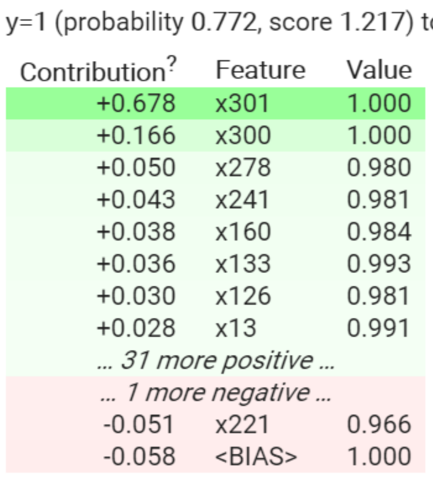
As we see in the figure 6, when we interpreted XGBOOST brand and the model features has the more contribution in predicting than the rest of the 300 dimensions of the page-title. This is mainly due to the values of the features are always 1 or 0.

Method	**Buckets**	**Coverage**	**Computation**
Super-Bit LSH	1 Stage - 2 Buckets	96%	92%
Super-Bit LSH	5 Stage - 2 Buckets	80%	55%
Super-Bit LSH	2 Stage - 3 Buckets	73%	36%
Super-Bit LSH	15 Stage - 2 Buckets	38%	7%
Super-Bit LSH	5 Stage - 5 Buckets	12%	3%
DPSH	690	95.89%	4.09%
DPSH	737	70.3%	3%
DPSH	1185	91.7%	1.5%
DPSH	1380	88.5%	1.65%
DPSH	2496	97%	0.72%

TABLE V: *Learnt and Unlearnt Hashing Evaluation Results*

Classifier	**F-measure on test data**	**F-measure on complete data**
Decision Tree	0.916	0.179
XGBoost	0.942	0.236

TABLE VI: *Decision Tree vs XGBOOST*



y=1 (probability 0.772, score 1.217) top features

Contribution?	Feature	Value
+0.678	x301	1.000
+0.166	x300	1.000
+0.050	x278	0.980
+0.043	x241	0.981
+0.038	x160	0.984
+0.036	x133	0.993
+0.030	x126	0.981
+0.028	x13	0.991
... 31 more positive ...		
... 1 more negative ...		
-0.051	x221	0.966
-0.058	<BIAS>	1.000

Fig. 7: *Interpreting XGBOOST*

F. Consistency check

There is a significant increase of number of pairs after the consistency check. But in our case we have a lot of false positives so our consistency check algorithm increased the false positives also because it has no knowledge of true positives and false positives. This is the reason in table VII we can see the increase of true-positives from 88,511 to 89,973 and false-positives from 243,249 to 479,583 but a decrease in the f-measure from 0.23 to 0.17.

Consistency check	**F-measure**	**True-positives**	**False-positives**
Before	0.23	88,511	243,249
After	0.17	89,973	479,583

TABLE VII: *F-measure before and after consistency check*

V. Conclusion and Future work

In this work, we proposed and evaluated five methods for an end to end Entity Resolution problem. All the methods contribute to different extents, weak supervision and hashing contribute the most. Our experiments indicate that our methods performed well on the test data but they did not perform well on the complete data. The winning solutions which are submitted to the competition are not machine learning based and are dataset tuned methods which require a lot of cleaning and rules. Problem with these type of methods is we cannot generalize beyond this dataset.

We conclude that Entity Resolution on heterogeneous schema is still a big challenge. It would be an interesting area to work on machine learning models which can generalize beyond datasets and domains. The consistency propagation algorithm we proposed unfortunately didn't work on our solution it is also a challenging area to improve such algorithms.

Acknowledgement

We would like to express our gratitude to Gabriel Campero Durand for the numerous inputs and constant discussions. We would like to thank Vishnu Unnikrishnan for the initial brainstorms. We would also like to thank Dr. David Broneske, Bala Gurumurthy, Xiao Chen and University of Magdeburg.

References

[1] Ebraheem, Muhammad et al. "DeepER - Deep Entity Resolution." ArXiv abs/1710.00597 (2017): n. pag.

[2] Papadakis, G., Ioannou, E., Palpanas, T. (2020). Entity resolution: Past, present and yet-to-come: From structured to heterogeneous, to crowd-sourced, to deep learned. Paper presented at EDBT/ICDT 2020 Joint Conference, Copenhagen, Denmark.

[3] Enriching Word Vectors with Subword Information, Bojanowski, Piotr and Grave, Edouard and Joulin, Armand and Mikolov, Tomas, arXiv preprint arXiv:1607.04606, 2016

[4] Chen, Tianqi, and Carlos Guestrin. "XGBoost." Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2016): n. pag. Crossref. Web.

[5] Fabian Pedregosa and Gael Varoquaux and Alexandre Gramfort and Vincent Michel and Bertrand Thirion and Olivier Grisel and Mathieu Blondel and Peter Prettenhofer and Ron Weiss and Vincent Dubourg and Jake Vanderplas and Alexandre Passos and David Cournapeau and Matthieu Brucher and Matthieu Perrot and Edouard Duchesnay, Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research, 2011, 2825-2830

[6] Jianqiu Ji and Jianmin Li and Shuicheng Yan and Bo Zhang and Qi Tian,Super-Bit Locality-Sensitive Hashing,year=2012,cdate=1325376000000,pages=108-116,url=http://papers.nips.cc/paper/4847-super-bit-locality-sensitive-hashing,booktitle=NIPS,crossref=conf/nips/2012

[7] Rutuja Shivraj Pawar, ' An evaluation of Deep hashing for High Dimensional Similarity Search on Embedded Data'.