Deep Entity Matching: An Entity Agnostic Data Harmonization Approach

Rajdeep Arora
raj007win@gmail.com

ABSTRACT
Deep Entity Matching is an entity agnostic approach to determine same/ similar entities. The problem has lots of relevance in customer identification across all POS data sources and hence know your customer better. The problem has also remained evident in ecommerce industry where there is a need to have a unique identifier for a product, so that it's easier to compare prices across various ecommerce websites. Similarly, the relevance of creating a single source of truth of an entity in the supply chain industries helps in route optimization and hence saves billions of dollars.

Traditionally creating a unique identifier has been a very labor-intensive or a heuristic work where humans need to look across the database to identify the duplicates. This process is time consuming and doesn't scale once the data goes beyond millions.

Deep Entity matching divided this task of entity deduplication / identity creation into 2 steps of blocking and matching. Since the problem has loads of opportunity across business and teams, the current implementation is entity agnostic. The same set of python modules works fine to create customer identification, seller identification and will also work for product identification across various ecommerce platforms.

Most Entity Matching solutions perform blocking first and then matching in order to reduce the pair wise comparisons O(n^2). However, the blocking traditionally has been either a rule-based approach or a hashing approach leveraging minLSH algorithms. Deepblocker framework made a significant advancement in the solution and apply recent advances in deep learning (e.g., sequence modeling, transformer, self-supervision). Then the results are compared across to empirically determine which solutions works best.

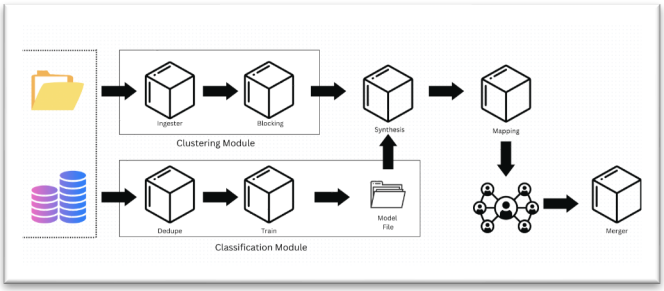
The data which has been considered for empirical evaluation is the Walmart and amazon publicly available product dataset. This is a structured dataset with textual columns of Title, Brand, Product Description, Price etc. Since its structured, the necessary NER can be avoided for the current research.

In the paper, the entire entity matching process is discussed with each component and the code is also shared for trying out the framework in user's data.

ARTIFACT AVAILABILITY
The source code, data, and/or other artifacts have been made available at https://github.com/rajdeep07/data-harmonization/pull/29

INTRODUCTION
Deep Entity Matching is a framework which performs 7 process to determine a unique identifier for set of entities which is a match. For entities, which the framework can't identify with others is called as singletons.

Majorly the framework can be divided into 4 components: Clustering (Ingester and Blocking), Classification (Dedupe and Train), Synthesis and Matching (Mapping & Merger).



Clustering:
Since this is an entity agnostic framework, I had to build another component called as Ingester to

identify attributes from the data. Currently the framework supports csv uploads and database like MySQL, however in future we can enhance the capability to read from other disparate data sources.

Once the entity is identified with its attributes, we run the blocking algorithm after feature cleansing. Currently the framework supports varied cleansing mechanisms of typos, synonymous, handles additional characters, and name swaps. After this we leverage various feature engineering vector representation modules to represents feature embedding of each entity which is leveraged to measure similarity-based distances among entities.

For the current minLSH implementation, Jaccard distances is used to measure the similarity between entities with an appropriate threshold of 0.65. One of the major reasons to have a high threshold is to ensure high recall so we don't miss of any potential matches in the downstream process.

Classification:
Since this is a supervised modeling approach, we need to build training dataset which has been a major area of concern. Considering this is an entity agnostic framework, it's important to create service which can create training data set with few examples provided from the user leveraging active learning framework.

The current implementation leverages dedupe python package to create positive example of training datasets and negative examples are creating by randomly selecting singletons pairs from clustering outputs.

Once the training data is created, we build a shallow and deep learning model and persist the model object. Persisting the model object is critical here since if we want to use this framework in production, the batch job would require blocking to run more frequently but there is no real need to retrain the classifier in the same frequency. Leveraging the same classifier would help in building more robust matching with reproducible and stable results. Stability is one of the most important success criteria for such data harmonization platforms since we want to track lineage of the IDs over time. With stability, we also ensure low merges to existing ids and splits which is required in most of the customer facing use cases.

Synthesis
Once we have the classifier model and the inferencing data which comes from blocking algorithm, we are good to run the synthesis module.

As part of this module, we first calculate various distance features like levenshtein distance, jarowrinkler distance, hamming distances and cosine distances after cleansing between the pairs of entities and then pass to shallow network of classifier. The results are then persisted in MySQL as output from the model.

Matching
Matching is a process of creating a cluster id for the entities which the framework decided to be the same. In order to determine the cluster id, the graph is created for all the entities as nodes and entities are joined if it's a match. Once the graph is created, we run depth first search (DFS) to identify isolated subgraphs and assign them the cluster ids. The advantage of creating cluster id like this is multi folds in terms of scale and reduced time complexity. Also, with this, we can also assign confidence to the matches in terms of strongly connected component being assigned the highest confidence score and confidence drops for weakly connected components.

Once the cluster id is generated from the DFS output, we run the merger service to determine the attribute values of the new cluster component. Currently, the selection is based on rules. By way of example, the product description is selected for merged cluster id based on highest length among the individual entities and emails is selected based on acceptable domains like gmail or yahoo over system generated bot's emails.

Evaluating the Deep Entity Matching Framework:
Since there are 2 major DL models applied here in clustering and classification. The evaluation also differ between them since both are trying to solve a different problem at hand.

Clustering algorithm is focused on High Recall and in view of that, we want to ensure every potential match is part of some cluster to have a classifier possible match pair and hence coverage. We measure the merge potential identified vs each singleton left to measure the performance of blocking algorithm.

Classifier algorithm is more focused on High Precision since having inappropriate matches would have a downsize effect on losing customers in case of customer data platforms. In the absence of ground truth, we aim to imitate dedupe algorithm with the hope to gain over its performance in few iterations.

There are other evaluation criteria like coverage, low latency, entity agnostic and stability which can be focused on selecting appropriate algorithms and architecture like leveraging graphs.

DESIGN SPACE OF DEEP LEARNING SOLUTIONS

The figure below shows the architecture template for the deep learning solution. The data source is provided and then fed with all identified attributed if not user provided else shortlisted based on parameter of attributes identified for blocking. The primary reason to use all attribute is that we don't really know without ground truth which label is more important and it supports the idea to stay entity agnostic since attributes would be inferred every time a new entity is introduced.

Template Module	Choices	
Word Embedding	Granularity (1) Word (2) Character	Training (1) Pre-trained (2) Learned
Tuple Embedding	(1) Aggregation based (a) Simple average (b) Weighted average (e.g., SIF) (2) Self-supervised *Many choices exist for each type of auxiliary tasks:* (a) Self-reproduction (b) Cross-tuple training (c) Triplet loss minimization (d) Hybrid	
Vector Pairing	(1) Hash (e.g., LSH) (2) Similarity based (a) Similarity measure: Cosine, Euclidean (b) Criteria: Threshold, KNN (3) Composite	

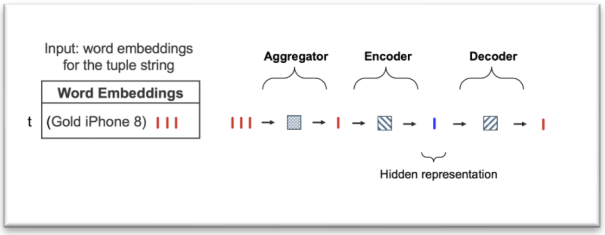
Once the attributes are identified and concatenated, the resulting strings are then fed into three main modules: Word Embedding, Tuple Embedding, and Vector Pairing. The Word Embedding module converts each word in the string into a high dimensional vector. The Tuple Embedding module combines these vectors into another vector representing the entire string. The Word Embedding module converts this string into four vectors (represented as four vertical red lines), then the Tuple Embedding module combines these four vectors into a single vector. Finally, the Vector Pairing module will find pairs of vectors that are similar. The corresponding pairs of the original tuples are then output as the set of candidate tuple pairs for the subsequent matching step.

Tuple Embedding Choices

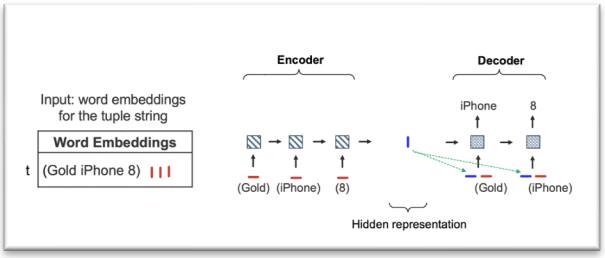
This module combines the embeddings of the words in a tuple into an embedding for the entire tuple. A key challenge is to ensure that similar tuples have similar embeddings without labeled data. To address this, we consider two broad categories of promising DL techniques: aggregation and self-supervision.

Aggregation Methods: These methods apply an aggregation function F : R de×· → R du to produce a tuple embedding ut . The most popular method is averaging, DeepER uses unweighted averaging where each word embedding has equal weight.

Self-Supervised Methods: These methods adapt the self-supervision idea popular in recent DL works. They work as follows:
(1) Define a supervised learning task, also called an auxiliary task, for which we can automatically derive labeled training data from the tuples.
(2) Solve the above task using a DL model trained on the labeled data.
(3) Use parts of the trained DL model to produce embeddings for the tuples.



Autoencoder Architecture

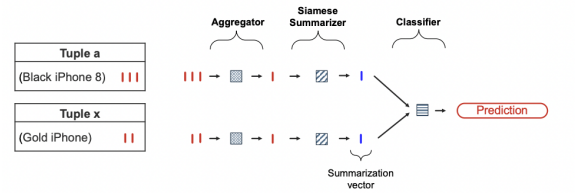


Seq2Seq Architecture

There are autoencoder and seq2seq architecture leveraged for encoding and more can be read in the paper published in Deep Blocker.

Cross Tuple Training Method

The self-reproduction approach exploits information within a single tuple to generate tuple embeddings. The key idea is to perturb the tuples of Tables A and B to generate synthetic labeled data, which is a set of tuple pairs (ti , tj) with match/no-match labels, then use this data to train a DL model to produce tuple embeddings, such that the embeddings of a matching tuple pair are closer to each other while those of non-matching tuple pairs are farther from each other.



Cross Tuple Training Architecture

The CTT architecture can be read in the paper published in Deep Blocker.

EMPIRICAL VALUATION

Dataset: We use datasets from an ecommerce products category of Walmart and Amazon. For Deep Entity Matching, we use dataset with 5 attributes; some of them are textual like brand, category, and title and some of them are numeric like price. For dirty EM, we focus on one type of dirtiness, which is widespread in practice, mainly due to information extraction glitches, where attribute values are 'injected' into other attributes (e.g., the value of 'brand' is missing and appears in attribute 'title'). There are loads of missing values and sometimes the field can hold both textual and numeric values like model no.

Blocking

We evaluated three techniques for blocking and below are show some of the statistics for model comparisons:

Approaches			
	Singletons	Merges	Time Taken
minLSH	10678	13950	70 seconds
DeepBlocker (Tuple Embedding)	12298	12330	150 seconds
DeepBlocker (CTT)	12264	12364	180 seconds

Singletons are the entities which couldn't find a match, merges are all potential matches found using a respective algorithm and time taken is the latency. Since there are 24,628 total entities available pre matching; latency is the function of number of records. Min Locality hashing architectures are pretty known for applying large scale clustering approaches on textual data.

Both the Deep blocker architecture of tuple embedding, and cross tuple training perform similarly but outperform LSH in finding fewer potential matches. minLSH although fast but lacks the hidden nuances of the feature interaction which other deep blocking algorithm takes into consideration as sophistication in their architecture.

One of the next steps would be to combine both tuple based and cross tuple training approaches as hybrid and apply to our dataset.

Classification

We evaluated the classifier by building engineering features of pairs like levenshtein distance, jaro wrinkler distance, hamming distances and cosine distances after cleansing the textual data because of typos, synonymous, additional unnecessary characters, and swaps.

Then we build a basis 2 hidden layer Shallow Neural Network model to predict the merges. This is built on top of the positive training data generated from python package dedupe which leverages active learning as few shots' architecture. The precision is bit low at 0.50.

In order to further improve on the simple shallow neural network model, we fine-tuned a pre-trained large language model DeBerta-v3-base(https://huggingface.co/microsoft/deberta-v3-base). We provided concatenated information of both the products together as a single string for encoding the information leveraging the LLM. Since this is an entity agnostic approach, we have not injected any domain specific information to support the low code entity deduplication platform. The

precision improved from a simple shallow neural network model to 0.72. We discuss various opportunity to enhance the performance of the classifier in the next section.

CONCLUSION & FUTURE WORK

In the paper, we specifically applied an entity agnostic deep learning-based approach to block an entity efficiently. There is various performance based and more accurate advancement possible which provides opportunity to enhance the capability of this low code platform.

Accuracy:

- Leverage DeBERTa for each individual product information as an encoder with custom layers and find a novel approach to fuse the embedding of the two products.
- The research for hybrid clustering solution which can apply the best of both worlds of tuple embedding and training examples of cross tuple learning would also improve the accuracy.
- There is an opportunity to create more features which could be specific to a use case and can be embedded in configuration as user input for this low code platform.

Performance:

- Improving Latency:
  There is opportunity to use event-based architecture (message queue) instead of services / python modules interacting as microservices with data contract lying in a database.

Traceability:

- Since stability and coverage being the important feature for this framework success, it's important to have a data lineage capability to understand the life span of a cluster id and changes like merge, split of clusters over time.
- Providing confidence score to individual sub-graphs using DFS on the based on strongly connected components and weakly connected component would provide belief in the architecture to various stakeholder. Idea being they can select the threshold of confidence based on use case which will drive higher adoption.

REFERENCES

1. GetoorL. et al. Entity resolution: Theory, practice & open challenges; 2012 (here)
2. R Baxter, P Christen, and T Churches. A Comparison of Fast Blocking Methods for Record Linkage
3. V Verroios, H Garcia-Molina. Top-K Entity Resolution with Adaptive Locality-Sensitive Hashing; 2019
4. LiY. et al. Deep entity matching with pre-trained language models; 2021
5. Deep Learning for Blocking in Entity Matching: A Design Space Exploration
6. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In NeurIPS. 5998–6008
7. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers). 4171–4186.
8. https://www.databricks.com/blog/2017/05/09/detecting-abuse-scale-locality-sensitive-hashing-uber-engineering.html