Literature Review for XCS224u: D2EM Deep Blocking & Deep Matching for Entity Resolution

Rajdeep Arora
raj007win@gmail.com

Introduction

Deep Neural Network has made significant progress not just in sentiment analysis, question & answers or conversational AI but also in automating and bring efficiency to process of identifying and integrating records belonging to unique entities. Traditionally the standard method was either rule based or machine learning models to match entities utilizing supervised approaches. In order to reduce this second order complexity for pair wise comparison, blocking has been done to cluster similar entities together. However, unlike matching, not a lot of sophisticated deep neural network model has been implemented in blocking.

In this project, I want to implement various sophisticated NLU based models to implement a sophisticated deep blocking technique to find matches within product types across major ecommerce like Walmart, Amazon and Google. For this, we will leverage structured data (Amazon & Google) here and (Amazon & Walmart) here since we want to focus primarily on model performance and not on NER specific capabilities.

Summary of Articles

1. GetoorL. et al. Entity resolution: Theory, practice & open challenges; 2012 (here)

This research brings together perspectives on Entity Recognition from a variety of fields, including databases, machine learning, natural language processing and information retrieval. Ironically, different subdisciplines refer to it by a variety of names, including record linkage, deduplication, co-reference resolution, reference reconciliation, object consolidation, identity uncertainty and database hardening.

Despite the long history of work on ER there is still a surprising diversity of approaches – including rule-based methods, pair-wise classification, clustering approaches, and richer forms of probabilistic inference – and a lack of guiding theory.

Naive ER algorithms that compare every pair of mentions is O(n^2). The paper reviews efficient indexing, blocking, and message passing techniques, that can reduce the complexity to near linear time. In addition, distributed computation can also significantly improve scalability of ER algorithms, and the research review recent work on distributed ER.

2. R Baxter, P Christen, and T Churches. A Comparison of Fast Blocking Methods for Record Linkage; 2003 (here)

Conventionally, Record linkage has remained the problem of great importance and a basic design space has been leveraged ever since. The basic idea proposed in the paper is ideally it makes complete sense to run matching algorithm if the records are fewer. However, if the number of records grow up to millions, we can find potential records first which could be a match (cluster) and then do pairwise matching within those clusters.

The paper suggests bigram indexing and clustering with TF-IDF. Each blocking techniques, helps in create a specific blocking key which is used to create potential pairs. Multiple blocking keys can also be used which are generated from various blocking algorithm to improve the overall performance in matching.

Sorting of the document can be done using fixed window length. This helps to limit the pairwise comparison in the matching phase.

From the evaluation standpoint, three metrics are used namely reduction ratio i.e., RR = 1 – s/N where s is number of records pairs produced from blocking and N is number of possible record pair in the entire dataset. RR doesn't consider the time complexity. Second metric being paired completeness i.e., PC = s/M where s is number of true match records in the set of records pairs produced from blocking and M is number of true match record from entire dataset. The third metric is the F score, which combines RR and PC via a harmonic mean, F score = 2×PC×RR / (PC+RR). It captures the trade-off between the pair's completeness and reduction ratio metrics. The author focusses on RR and PC, rather than precision and recall of the overall linkage results, because that allows direct evaluation of the indexing methods without possible confounding of results with the comparison and decision methods.

3. V Verroios, H Garcia-Molina. Top-K Entity Resolution with Adaptive Locality-Sensitive Hashing; 2019 (here)

This paper is a bit advancement on the previous research as it leverages adaptive locality sensitive hashing for blocking the records. Unlike LSH which uses a set of hash tables and applies many hash functions on each record of the dataset. Every two records that LSH places in the same bucket, in one of the tables, are considered "similar", i.e., referring to the same entity for ER. Nevertheless, LSH requires many hash functions to be applied on each record. That is, while the cost of applying LSH is linear to the size of the input, hence the cost of processing each record is high.

The Adaptive LSH approach research proposes can process vast majority of records in the dataset with a very low cost per record, showing a speedup of up to 25x, compared to traditional-LSH blocking. Rest of the paper talks about the

Adaptive LSH starts by applying a small number of LSH functions on each record in the dataset. It then detects which areas are sparse and which are dense and continues by

applying more LSH functions only to the records in the dense areas, until it converges to the records referring to the top-k entities. Thus, Adaptive LSH can very efficiently filter out records that are highly unlikely to be records referring to top-k entities, and only requires a higher cost for records to the top-k entities.

4. LiY. et al. Deep entity matching with pre-trained language models; 2021 (here)

This paper suggests leveraging large language model instead of typical matching process and their experiments show that a straightforward application of language models such as BERT, DistilBERT, or RoBERTa pre-trained on large text corpora already significantly improves the matching quality and outperforms previous state-of-the-art (SOTA), by up to 29% of F1 score on benchmark datasets.

Ditto, a novel Entity Matching solution based on pre-trained Transformer-based language models. It is a sequence-pair classification problem to leverage such models, which have been shown to generate highly contextualized embeddings that capture better language understanding compared to traditional word embeddings. Apart from that, it leverages following optimization to improve matching algorithm:

- It allows domain knowledge to be added by highlighting important pieces of the input that may be useful for matching decisions.
- It summarizes long strings so that only the most essential information is retained.
- It augments training data with (difficult) examples, which challenges Ditto to learn more and reduces the amount of training data required.

Largely, a pre-trained LMs learns the semantics of words better than conventional word embedding techniques such as word2vec, GloVe, or FastText. This is largely because the Transformer architecture calculates token embeddings from all the tokens in the input sequence and thus, the embeddings it generates are highly contextualized and captures the semantic and contextual understanding of the words.

From the training purpose, fine-tune of a pre-trained Language Model for the Entity Matching task with a labeled training dataset in following ways:

- Add task-specific layers after the final layer of the Language Model. For Entity Matching, we add a simple fully connected layer and a SoftMax output layer for binary classification.
- Initialize the modified network with parameters from the pretrained LM.
- Train the modified network on the training set until it converges.

For the optimizations, the research digs deeper to estimate various ways for providing domain knowledge via span typing and span normalization. A span of tokens is provided by inserting special tokens to reflect the information annotators normally read together.

Instead of truncating long text to 512 words token, In Ditto's current implementation, we use a TF-IDF-based summarization technique that retains non-stop word tokens with the high TF-IDF scores.

For Data Augmentation, MixDA computes a convex interpolation of the original example with the augmented examples. Hence, the interpolated example is somewhere in between, i.e., it is a "partial" augmentation of the original example, and this interpolated example is expected to be less distorted than the augmented one.

5. Deep Learning for Blocking in Entity Matching: A Design Space Exploration (here)

This research proposes Deep Blocker framework that significantly advances the state of the art in applying Deep Learning to blocking for Entity Matching. These solutions do not require labeled training data and exploit recent advances in DL (e.g., sequence modeling, transformer, self-supervision). Also, it empirically determines which solutions perform best on what kind of datasets (structured, textual, or dirty).

There were primarily 2 limitation this research aims to solve:

- First, they consider only a relatively simple way to convert each tuple into a vector, namely by combining the vectors of the words in the tuple using unweighted or weighted averaging. This method is called aggregation. Such methods suffer from low recall accuracy.
- Another limitation of the existing works is that some of them require labeled data, Autoblock [here] uses this data to learn the weights for combining the word vectors based on semantics, position, and the surrounding context. Such labeled data is difficult to generate for blocking in many real-world Entity Matching scenarios.

Some of the important pointers from research was:

- It develops self-supervised solutions for blocking. The training data is generated as part of the process.
- Defining a Rich Space of DL Solutions as an architecture or framework where various blocking algorithm can be tested and verified.

The research digs deeper in various blocking techniques:

- Attribute equivalence, hash, and sorted neighborhood:
  Attribute outputs a pair of tuples if they share the same values of a set of attributes. Hash blocking (also called key-based blocking) is a generalization of AE, which outputs a pair of tuples if they share the same hash value, using a pre-specified hash function. Sorted neighborhood outputs a pair of tuples if their hash values (also called key values) are within a pre-defined distance.
- Other blockers include similarity, rule-based, and composite blocking:
  Similarity blocking is like AE, except that it accounts for dirty values, misspellings, abbreviations, and natural variations by using a predicate

involving string similarity measures, such as edit distance and Jaccard distances. Rule-based blocking employs multiple blocking rules, where each rule can employ multiple predicates.
Composite blocking (e.g., canopy blocking) generalizes rule-based blocking and can combine arbitrary blocking methods.

- Schema agnostic tokenization a.k.a. token-based blocking
- Meta-blocking techniques improves upon this process by constructing a blocking graph and using it to discard redundant comparisons.
- DeepER: This computes tuple vector via unweighted aggregation of the vectors of the individual words.
- AutoBlock: It improves upon DeepER as by using a set of tuple pairs labeled as match/no-match to learn the weights for the aggregation.

Architecture Template:

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	

The template provides various options like word vs character level embeddings or leveraging pre-trained vs learned embeddings. Also, when the embedding is generated for each word or characters, the tuple embedding choice is either aggregation or self-supervised methods. The aggregator then is feed to encoder-decoder architecture.

Auxiliary Tasks	Solution Architectures	Instantiation Examples
Self-Reproduction	Aggregator + Encoder + Decoder	• Aggregator: SIF, LSTM, . . . • Encoder/Decoder: Feed-Forward NN, LSTM, Transformer
Cross-Tuple Training	Aggregator + Summarizer + Classifier	• Aggregator: SIF, LSTM, . . . • Classifier: Feed-Forward NN, LSTM, Cosine
Triplet Loss Minimization	Aggregator + Loss Minimizer	• Aggregator: BERT, . . .
Hybrid	Many possible architectures exist, e.g.: • encoder + summarizer + classifier • SBERT aggregator + summarizer + classifier	• Each architecture has many components • Each component has many instantiation choices as listed above

There are primarily two tuple embedding choices and these are aggregation methods and self-supervised methods. The paper delves in detail to discuss weighted and unweighted aggregation strategies. Also, it considers four types of auxiliary tasks in self supervised methods like self-reproduction, cross-tuple training, triplet loss minimization, and hybrid.

Conclusion: *General idea being {Entity matching} = {Blocking} + {Matching}*

1. The first paper discusses about entity matching in general and what problem space in NLU it represents.
2. The second paper here discuss about slightly better than pairwise comparison leveraging TF-IDF and block hash key base comparisons.
3. The third paper gone beyond and leverages non key base hash key (Locality Sensitive Hashing).
4. The fourth paper discuss better matching algorithm leveraging pre trained large language models.
5. The fifth paper gone beyond and applied deep learning pre trained large language models in blocking itself.