ACM SIGMOD Programming Contest 2022

SUSTech_DBGroup · Finalist

Jiarui Luo luojr2019@mail.sustech.edu.cn
Xinying Zheng zhengxy2019@mail.sustech.edu.cn
Renjie Liu liurj2019@mail.sustech.edu.cn

Advisor: Professor Bo Tang, Professor Xiao Yan

DBGroup@SUSTech: https://dbgroup.sustech.edu.cn/

Task Overview

Task: Entity Blocking, a filter for Entity Resolution, aims to filter out obvious non-matching pairs and obtain a much smaller candidate set for the following matching step.
Input/Output: Inputs are instances of products from different e-commerce websites and we are required to output a set of instance pairs that may refer to the same real-world product.
Performance Metric: Recall (the percentage of true instance pairs listed in our output) with a predetermined output size.

Solution Overview

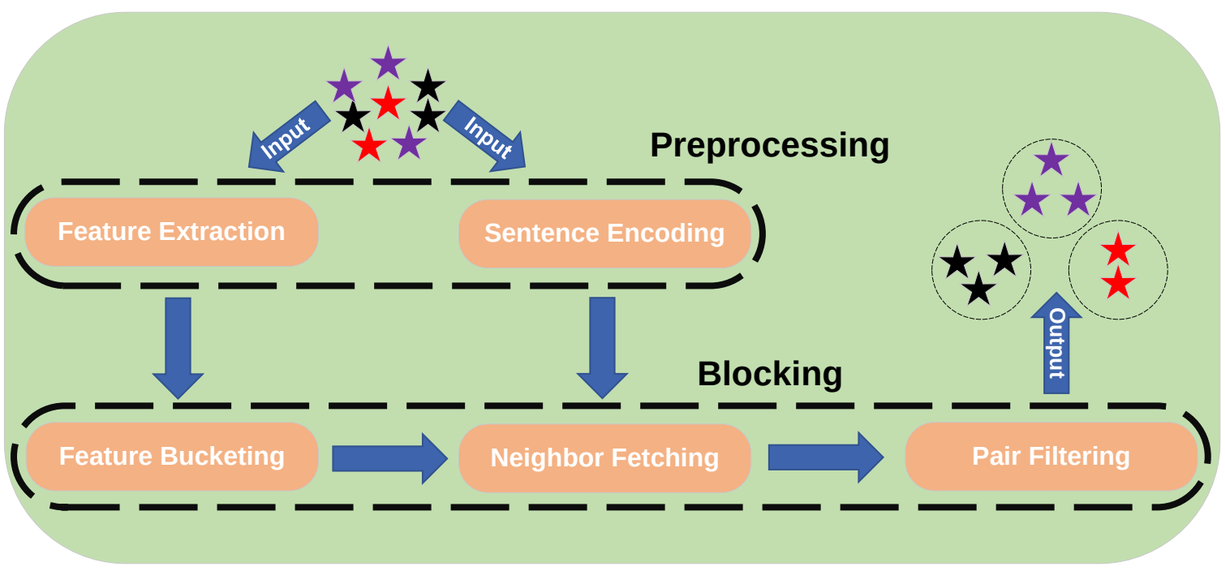

Figure 1: Solution framework

Preprocessing:
- For each instance, we extract key features from its description sentence.
- For each instance, we encode its description sentence to a vector.

Blocking: Put instances with similar features into a bucket, search neighbors for each instance in its bucket, and filter out impossible instance pairs.

Preprocessing

Feature Extraction

- Use regular rules to extract key features, for example:

Feature	Reugular Expression	Value
cpu_model	[AaEe][0-9][\-][0-9]{4}	A8-5545

- Result:

Instance_id	Feature 1	Feature 2	Feature 3	Feature 4
X	●	NULL	▲	◆
Y	●	⬟	▲	◆
Z	●	⬢	NULL	▰

Sentence Encoding

Model: Pre-trained BERT model, fine-tuned to make matching instance pairs more similar than mismatching instance pairs [1]
Output: A 128-dimension vector for each instance
Forward: Corpus + Tokenizer + Pooling
Backward: Minibatch + Triplet Loss

Triplet Loss $Minimize\ L = max(d(a,p) - d(a,n) + margin, 0)$
- a: Anchor description sentence
- p: Sentence having the same label as the anchor sentence
- n: Sentence having a different label from the anchor sentence
- margin: A positive hyperparameter
- d: Euclidean distance

Blocking

Feature Bucketing

- Divide the input instances into buckets based on the features of the products.
- For buckets of high confidence (e.g., instances in the bucket have some critical features in common), generate pairs in those buckets directly.

Neighbor Fetching

- For each instance, we find its top-k nearest neighbors to match with it and generate pairs.
- Since searching top-k neighbors in a bucket is time-consuming and costly, we build an HNSW index for each bucket to accelerate the search process [2].

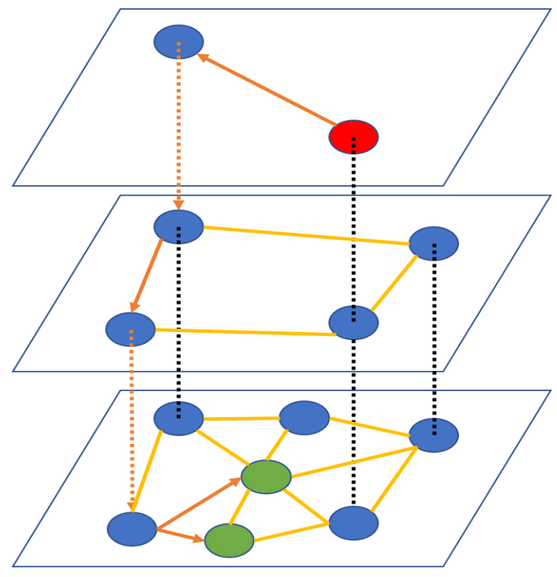

Figure 2: HNSW index

Pair Filtering

- Remove duplicated pairs and filter out pairs based on specific rules (e.g., key features like brand don't match).
- Sort the remaining pairs according to the **Euclidean distance** between the embedding vectors of a pair.
- Output the pairs in the sorted order until reaching the predetermined output size.

Result

Dataset	Recall	Time
X1	0.692	1671s
X2	0.323	

References

[1]J. Devlin, M. Chang, K. Lee, K. Toutanova, BERT: pre-training of deep bidirectional transformers for language understanding, in: J. Burstein, C. Doran, T. Solorio (Eds.), NAACL-HLT 2019.

[2]Y. A. Malkov, D. A. Yashunin, Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs, TPAMI, 2018.