Exploring the Design Space of Unsupervised Blocking with Pre-trained Language Models in Entity Resolution

Chenchen Sun[1](✉), Yuyuan Jin[1], Yang Xu[1], Derong Shen[2], Tiezheng Nie[2], and Xite Wang[3]

[1] School of Computer Science and Engineering, Tianjin University of Technology, Tianjin, China
suncc_db@163.com
[2] School of Computer Science and Engineering, Northeastern University, Shenyang, China
{shendr,nietiezheng}@mail.neu.edu.cn
[3] College of Information Science and Technology, Dalian Maritime University, Dalian, China
wangxite@dlmu.edu.cn

Abstract. Entity resolution (ER) finds records that refer to the same entities in the real world. Blocking is an important task in ER, filtering out unnecessary comparisons and speeding up ER. Blocking is usually an unsupervised task. In this paper, we develop an unsupervised blocking framework based on pre-trained language models (B-PLM). B-PLM exploits the powerful linguistic expressiveness of the pre-trained language models. A design space for B-PLM contains two steps. (1) The Record Embedding step generates record embeddings with pre-trained language models like BERT and Sentence-BERT. (2) The Block Generation step generates blocks with clustering algorithms and similarity search methods. We explore multiple combinations in above two dimensions of B-PLM. We evaluate B-PLM on six datasets (Structured + dirty, and Textual). The B-PLM is superior to previous deep learning methods in textual and dirty datasets. We perform sufficient experiments to compare and analyze different combinations of record embedding and block generation. Finally, we recommend some good combinations in B-PLM.

Keywords: entity resolution · unsupervised blocking · pre-trained language models · data integration · deep learning

1 Introduction

Entity resolution (ER) is a fundamental problem in data integration and data governance. Blocking [1] is a key issue in ER, reducing unnecessary comparisons and improving ER efficiency. Blocking methods place similar records in the same block and then perform intra-block comparisons. Traditional blocking methods are based on blocking keys, which require human selections for each dataset. How to reduce human involvement? That is, how to get high-quality blocks without block keys. Meanwhile, traditional blocking techniques are difficult to achieve good blocking results on textual and dirty datasets. Thus, how to perform blocking on text and dirty datasets is another issue.

Many deep learning (DL) works make contributions to reducing human efforts. As DL continues to evolve, DL has been widely used in ER [2, 3]. These works apply similarities between record embeddings to calculate the likelihood of matches. For efficiency, some DL-based ER methods [2, 3] use hash functions for blocking. DL-based blocking approaches are rare [4–6]. Meanwhile, blocking is still seen as a step in ER, rather than as a separate problem. Most current DL-based blocking approaches [2–4] use labeled record pairs to train representation models for blocking. Other DL-based blocking approaches [5] use the generated auxiliary labels for training. However, blocking essentially filters irrelevant record pairs, similar to indexing in databases. Therefore, blocking shouldn't use labeled data. Moreover, as described in [6], in most blocking scenarios there is no labeled data available. To this end, this paper explores the design space of unsupervised blocking solutions.

We propose an unsupervised blocking framework based on pre-trained language models (B-PLM). B-PLM is a simple but strong baseline for deep unsupervised blocking. Pre-trained language models (PLMs), like BERT [7], Sentence-BERT [8], Doc2Vec [9], and fastText [10], can provide rich semantic information for blocking. The potential of PLMs in unsupervised blocking has not been systematically explored. In addition, the PLMs are outstanding in handling long text data. At the same time, PLMs can handle other types of datasets (e.g., structured and dirty). Therefore, based on PLMs, B-PLM consists of two major steps: record embedding and block generation. Record embedding obtains the semantic features of records based on PLMs and aggregation methods. PLMs have strong capabilities of semantic representation and discrimination, which are useful for record representation. For block generation, the essence is to group records according to estimated similarities. Clustering algorithms [11–13] are often used for grouping tasks; top-k query and range query in databases also have similar capabilities. Therefore, we provide clustering algorithms and similarity search methods for block generation. Our experimental results demonstrate effectiveness of B-PLM. We design sufficient experiments to investigate impacts of different record embedding methods and different block generation methods. Finally, we recommend some blocking solutions in our framework.

The main contributions are as follows.

- We define a solution space for an unsupervised blocking framework based on pre-trained language models (B-PLM), consisting of five record embedding methods and seven block generation options.
- We demonstrate B-PLM's effectiveness with experimental evaluations on six datasets. Experimental results show that optimal methods in B-PLM greatly outperform the state-of-the-art work DeepBlocker [5]. This result demonstrates B-PLM on a textual dataset.
- We systematically compare and analyze characteristics and effectiveness of different record embedding methods and different block generation methods in B-PLM. Furthermore, we further studied the different layers of BERT and Sentence-BERT. Finally, we compared parameter sensitivity of block generation methods.

Organization. Section 2 introduces the B-PLM framework in detail. Section 3 conducts extensive experiments to compare different combinations and analyze each component in B-PLM. Section 4 presents related work. Section 5 concludes the paper.

2 The Design Space of Unsupervised Blocking with Pre-Trained Language Models

2.1 Framework Overview

Given a set of records from a single data source or multiple data sources, blocking filters out unnecessary comparisons by dividing the records that are likely to match into the same block as much as possible. To this end, we proposed an unsupervised blocking framework based on pre-trained language models (B-PLM), which contains two core steps, as shown in Fig. 1.

Step 1: **Record Embedding**. This step generates record embeddings for later block generation. First, each record transforms into a token sequence. Second, token sequences input a PLM to get record embeddings. Record embeddings are either directly generated or produced with aggregations of token embeddings. We explored the current dominant PLMS. This step provides five options: BERT + Average, Sentence-BERT, Doc2Vec, fastText + Average, and fastText + SIF, which are introduced in Sect. 2.2.

Step 2: **Block Generation**. With record embeddings, similar records should be grouped into the same blocks. In Fig. 1, block generation methods are divided into two categories: one is based on clustering and the other is based on similarity search. We explore the current common clustering algorithms and similarity search methods. Clustering algorithms include AHC, BIRCH, DBSCAN, AP and *k*-means; and similarity search methods include top-*k* query and range query, which are specified in Sect. 2.3.

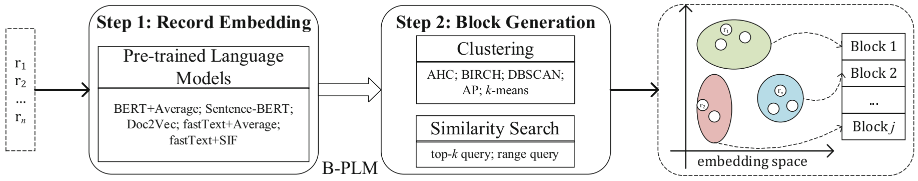

Fig. 1. Unsupervised blocking framework with pre-trained language models.

2.2 Record Embedding

This step provides five options for record embedding, as Fig. 1 shows. We convert records into sequences for input into PLMs to obtain record embedding. Formally, a record $r = \{attr_i, val_i\}_{1 \le i \le k}$ converts to a sequence S(r). This paper considers both the case with and without schema information. In the case with schema information, S(r) = [CLS] [ATT] $attr_1$ [VAL] val_1 …[ATT] $attr_k$ [VAL] val_k [SEP]. In the case without schema information, S(r) = [CLS] val_1 … val_k [SEP]. [ATT] and [VAL] indicate the beginning of the attribute name and attribute value, respectively.

BERT (Bidirectional Encoder Representations from Transformers) [7] adopts a multi-layer transformer encoder architecture, as Fig. 2 shows. BERT generates an embedding for each token, thus averaging all token embeddings in a sequence to get

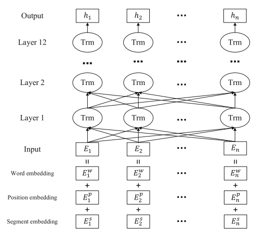

Fig. 2. Architecture of BERT.

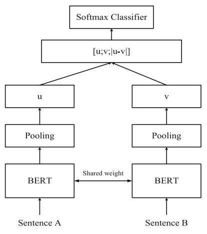

Fig. 3. Architecture of Sentence-BERT.

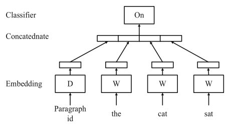

Fig. 4. Architecture of Doc2Vec.

record embeddings. BERT tends to encode all records into a smaller space region, which results in most record pairs having a high similarity score.

SBERT (Sentence-BERT) [8] is a modification of BERT, as shown in Fig. 3. Sentence A and Sentence B are put into shared weight BERT models, respectively. Going by pooling gets the embedding u and embedding v corresponding to sentence A and sentence B. In contrast to BERT, SBERT enables semantically similar sentences to be similar in the embedding space as well.

Doc2Vec [9] learns embeddings from text fragments with variable lengths. As shown in Fig. 4, a paragraph vector is introduced in Doc2Vec. In B-PLM, Doc2Vec uses paragraph vectors as record embeddings.

FastText [10] is a character-level pre-trained word embedding model. FastText can generate embeddings for tokens outside the vocabulary and is robust to some spelling errors. B-PLM provides two fastText methods, $\underline{\text{fastText} + \text{Average}}$ and $\underline{\text{fastText} + \text{SIF}}$. The average method does not consider the importance of each token in the record. In contrast, SIF [14] is a state-of-the-art solution for weighted averaging.

2.3 Block Generation

The block generation step contains two types of methods, clustering algorithms and similarity search methods.

Clustering Algorithms. There are five clustering algorithms: AHC, AP, DBSCAN, BIRCH and *k*-means.

AHC (Agglomerative Hierarchical Clustering) [11] is a bottom-up hierarchical clustering algorithm. In the beginning, each sample point is treated individually as a cluster. Clusters with high similarities are merged until all sample points form a cluster or a certain similarity threshold is reached.

BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) [12] is a hierarchical clustering algorithm. The main step in BIRCH is the creation of CF (clustering feature) tree. BIRCH does not work well if the clusters are not spherical, as it uses the concept of radius to control cluster boundaries.

DBSCAN (Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise) [11] is a density-based clustering algorithm that can discover clusters of arbitrary shape. It defines a cluster as the largest set of density connected points.

AP (Affinity Propagation) [13] is a graph-based clustering algorithm. The algorithm starts by treating all sample points as cluster centers, and clustering is run by passing messages between sample points. Automatically it identifies the number of clusters from sample points by maximizing the similarity sum of all sample points to their nearest cluster centers.

***k*-means** [11] is a partitioning-based clustering algorithm. First, k initial clustering centers are randomly selected. Next, each point is arranged into the cluster closest to that point. Subsequently, the cluster centers are recalculated for each cluster. It is repeated until the result is stable.

Similarity Search Methods. Similarity search methods include top-k query and range query.

Top-*k* query. For each record r, similarities between r and other records are first computed, and then records with the k highest similarities are selected to form a block with r. We use top-k for short in this paper.

Range Query. For each record r, similarities between r and other records are first computed, and then records with similarity scores above a given threshold are selected to form a block with r.

3 Comparative Experimental Evaluation and Analysis

3.1 Experiment Setup

Datasets. We evaluate B-PLM on six ER datasets, which are shown in Table 1. Most of the datasets in this paper are textual datasets, and previous blocking methods have struggled to achieve high-quality blocking on textual datasets. Cora is a common ER dataset. Notebook and Altosight are both from Blocking Contest in SIGMOD 2022[1]. Notebook is a dataset about laptops, with only one title attribute. Altosight is a dataset about electronic products, with attribute values that may be misplaced or missing. WDC_cameras is from the Web Data Commons project [15] and we use the cameras_small version. In WDC_cameras, we only use the title attribute. Both Abt-Buy and Amazon-Google2 are dual-source datasets about products.

We further analyze the characteristics of datasets. We obtain the entity redundancy figure for each dataset, which details the number of records corresponding to each entity, as shown in Fig. 5. In Cora, most entities have a large amount of redundancy. In Notebook, Altosight and WDC_cameras, few entities have more than 8 corresponding records. Almost all entities in Abt-Buy and Amazon-Google2 have only 1–3 corresponding records. Cora has the highest entity redundancy while Abt-Buy and Amazon-Google have the lowest.

Metrics. We use pair completeness (PC), reduction ratio (RR), Fα and PC-RR curve as evaluation metrics. $PC = |G \cap C|/|G|$, $RR = 1\text{-}(|C|/|D|)$, where G denotes all truly matched pairs, C denotes candidate pairs generated by blocking, and D denotes all record pairs

[1] http://sigmod2022contest.eastus.cloudapp.azure.com.

Table 1. Dataset details.

	Dataset	#records	#attributes	Type
single-source	Cora	1295	12	Structured + Dirty
	Notebook	1661	1	Textual
	Altosight	1993	5	Structured + Dirty
	WDC_cameras	1904	1	Textual
dual-source	Abt-Buy	1081 + 1092	3	Textual
	Amazon-Google2	1363 + 3226	2	Textual

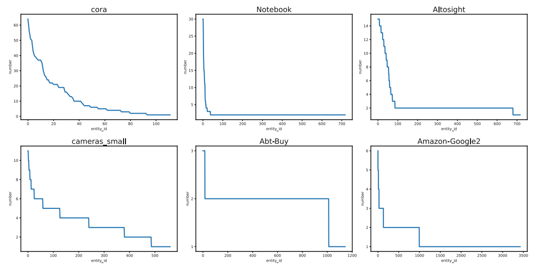

Fig. 5. Entity redundancy figures of datasets.

in a dataset. PC measures how many truly matched pairs are retained in blocks, while RR measures how many comparisons are reduced. We use Fα (the harmonic mean of PC and RR) to measure the overall performance, Fα = 2 · PC · RR/(PC + RR). In the PC-RR curve, the closer the curve is to the upper right, the better the performance of a blocking method is.

Baselines. DeepBlocker [5] is a state-of-the-art deep blocking work. DeepBlocker proposes eight blocking solutions. DeepBlocker first uses fastText to get the word embedding. The word embeddings are then aggregated into record embeddings and finally paired based on cosine similarity between vectors. We select the three best solutions: SIF, AE, and CTT as the baseline. AE uses full connection layers as encoder and decoder and the output of encoder as the record embedding. CTT uses a data generation procedure to automatically generate labeled record pairs. Then CTT learns record embeddings through classification tasks. In block generation, SIF, AE, and CTT all use top-*k* query.

Settings. Experiments are implemented with following Python libraries: Scikit-learn, transformer, sentence-transformer, fastText and gensim. All experiments are run on a server with an Inter Core I9-10900K CPU @ 3.70 GHz, 32 G RAM and an NVIDIA Quadro RTX 4000 GPU. Record embedding uses SBERT in default. All PLMs are used in this paper without fine-tuning. For BERT, we use the pre-trained version of “bert-base-uncased”. For SBERT we use the pre-trained version of “all-mpnet-base-v2”. For Doc2Vec we use the pre-trained version of “English Wikipedia DBOW”. For fastText, we use the pre-trained version of “Wiki.En”. Except for *k*-means the other block generation methods do not specify the number of clusters, which is more in line with real-world situations. The *k* of *k*-means is chosen from 10 to 50 and the step size is set to 2. Similarity search methods use cosine similarity.

3.2 Overall Performance

For each dataset, we report the best result from SIF, AE, and CTT as DeepBlocker. Overall comparison results are presented in Table 2 and Table 3. Overall B-PLM has advantages in textual and dirty datasets. The best performer in our framework is SBERT + top-*k*. SBERT + top-*k* has an average Fα of 93.65% on six datasets and a 5.55% improvement compared to the baseline. SBERT + range query has an average Fα of 92.85% on the six datasets, and a 4.75% improvement compared to the baseline. SBERT + AHC and SBERT + *k*-means also achieve better results than the baseline. In addition, except for Notebook, on the rest datasets, the best, second, and third solutions are all from our framework, demonstrating B-PLM's effectiveness.

The powerful semantic expressiveness of SBERT is reflected in the results. The results of DeepBlocker and SBERT + top-*k*, which both take the same block generation method, and we find that SBERT + top-*k* performs significantly better than DeepBlocker on most datasets.

For Cora, the clustering algorithms perform better overall. As illustrated in Fig. 5, many entities in Cora have high redundancy. In the remaining datasets, most entities have low redundancy (no more than five records). Clustering methods can generate blocks of different sizes according to the redundancy level of each entity, which is a self-adaption process. Therefore, clustering methods are more advantageous on datasets with various high entity redundancy.

The dual-source datasets Abt-Buy and Amazon-Google2 are more suitable for top-*k* query and range query. Because the dual-source dataset assumes that there are no duplicates within each data source. The executions of top-*k* query and range query naturally ensure that each record of data source A finds the most similar records in data source B to form a block. Top-*k* query and range query are not interfered with by similar records within data sources, making them easier to get high-quality blocks. Dual-source datasets are more difficult for clustering-based block generation methods. Because clustering-based methods run in the entire set of two data sources, but do not only conduct comparisons between two data sources.

3.3 Analysis of Record Embedding

Effectiveness of Record Embedding Methods. We compare record embeddings with different PLMs in blocking. The results are shown in Tables 4, 5, 6, 7, 8, 9 and 10. For both BERT and SBERT, we report the results at layer 12, and for *BERT and *SBERT, we report their results at optimal layers.

Overall, the best record embedding method is SBERT. The overall performance of SBERT is better than other PLMs for each block generation method. SBERT's strong linguistic power is demonstrated. The overall results of BERT are generally worse than those of SBERT, mainly because the focus of BERT in the pre-training step is to produce

Table 2. Overall Performance on six datasets (part 1 in 2). Bold, single underline and double underline represent the best, second and third, respectively.

Method	Dataset											
	Cora			Notebook			Altosight			WDC_cameras		
	PC(%)	RR(%)	Fα(%)	PC(%)	RR(%)	Fα(%)	PC(%)	RR(%)	Fα(%)	PC(%)	RR(%)	Fα(%)
DeepBlocker	88.08	91.73	90.38	96.80	90.68	93.64	68.76	79.49	73.74	83.65	91.83	87.55
SBERT + AHC	93.18	94.03	93.60	93.99	97.76	95.84	70.08	79.74	74.60	89.64	87.66	88.63
SBERT + BIRCH	93.47	96.08	94.76	89.52	97.84	93.49	70.77	82.57	76.21	72.97	91.93	81.36
SBERT + DBSCAN	92.10	96.94	94.46	97.30	48.54	64.77	73.63	55.17	63.08	81.22	81.28	81.25
SBERT + AP	93.51	96.77	95.11	88.88	87.69	88.28	64.57	83.05	72.65	81.22	88.12	84.53
SBERT + *k* -means	93.46	97.07	**95.23**	81.49	97.16	88.64	74.52	90.00	81.53	85.19	89.54	87.31
SBERT + top-*k*	93.18	94.80	93.98	95.42	88.72	91.95	86.95	88.00	87.47	92.13	91.83	**91.98**
SBERT + range query	91.74	95.44	93.55	94.95	97.58	**96.25**	79.49	82.36	**80.90**	91.36	90.79	91.07

Table 3. Overall Performance on six datasets (part 2 in 2).

Method	Dataset						
	Abt-Buy			Amazon-Google2			AVG of Six
	PC(%)	RR(%)	Fα(%)	PC(%)	RR(%)	Fα(%)	Fα(%)
DeepBlocker	95.04	95.24	95.14	82.86	94.17	88.16	88.10
SBERT + AHC	97.08	95.41	96.24	86.29	97.38	91.50	90.07
SBERT + BIRCH	91.44	93.55	92.48	67.10	95.88	78.95	86.21
SBERT + DBSCAN	91.83	84.34	87.92	86.46	69.71	77.19	78.11
SBERT + AP	88.23	93.87	90.96	61.95	96.35	75.42	84.49
SBERT + *k* -means	92.12	96.64	94.33	72.32	94.96	82.11	88.19
SBERT + top-*k*	99.22	97.25	**98.23**	98.46	98.14	**98.30**	**93.65**
SBERT + range query	98.64	96.77	97.70	97.00	98.27	97.63	92.85

better token embeddings rather than sentence embeddings. In contrast, SBERT learns embeddings of whole sentences directly, requiring that semantically similar sentences

Table 4. Results of AHC using different record embedding methods. Bold, single underline and double underline represent the best, second and third, respectively.

Embedding	Dataset						
	Cora Fα(%)	Notebook Fα(%)	Altosight Fα(%)	WDC_cameras Fα(%)	Abt-Buy Fα(%)	Amazon-Google2 Fα(%)	AVG Fα(%)
BERT	74.38	94.15	54.98	65.55	55.61	65.60	68.38
SBERT	93.60	95.84	**74.60**	**88.63**	**96.24**	**91.50**	90.07
Doc2Vec	90.99	94.86	65.54	80.90	66.40	64.60	77.22
fastText + Average	75.03	93.41	58.49	72.26	54.85	56.54	68.43
fastText + SIF	88.80	95.64	43.45	74.81	86.42	73.74	77.14
*BERT	87.56(l = 1)	94.71(l = 5)	63.82(l = 0)	78.45(l = 0)	69.92(l = 1)	66.62(l = 1)	76.85
*BERT + Schema	1.53(l = 12)	96.60(l = 6)	63.96(l = 1)	74.17(l = 0)	54.41(l = 2)	62.91(l = 11)	58.93
*SBERT	**96.17(l = 0)**	**96.65(l = 0)**	**74.60(l = 12)**	**88.63(l = 12)**	**96.24(l = 12)**	**91.50(l = 12)**	**90.63**
*SBERT + Schema	92.43(l = 12)	96.64(l = 0)	**76.40(l = 12)**	85.65(l = 12)	93.45(l = 12)	85.41(l = 12)	88.33

are also similar in the embedding space. The performance differences between BERT and *BERT are large. The reasons for this will be analyzed later.

Table 5. Results of BIRCH using different record embedding methods.

Embedding	Dataset						
	Cora Fα(%)	Notebook Fα(%)	Altosight Fα(%)	WDC_cameras Fα(%)	Abt-Buy Fα(%)	Amazon-Google2 Fα(%)	AVG Fα(%)
BERT	76.60	92.80	52.89	63.75	36.88	55.08	63.00
SBERT	94.76	93.49	**76.21**	**81.36**	**92.48**	**78.95**	86.21
Doc2Vec	26.39	49.95	24.95	6.80	0.19	0.51	18.13
fastText + Average	76.75	90.18	25.89	44.07	31.10	52.13	53.35
fastText + SIF	77.09	94.69	48.84	60.63	63.68	61.41	67.72
*BERT	88.15(l = 3)	94.70(l = 2)	64.65(l = 0)	64.48(l = 9)	50.60(l = 3)	57.19(l = 5)	69.96
*BERT + Schema	79.29(l = 2)	93.81(l = 4)	55.17(l = 8)	67.68(l = 12)	47.49(l = 4)	61.12(l = 11)	67.43
*SBERT	**96.39(l = 0)**	**94.82(l = 0)**	**76.21(l = 12)**	**81.36(l = 12)**	**92.48(l = 12)**	**78.95(l = 12)**	**86.70**
*SBERT + Schema	92.93(l = 12)	94.36(l = 5)	73.77(l = 12)	80.44(l = 12)	89.17(l = 12)	74.93(l = 12)	84.27

FastText + Average is less effective than fastText + SIF. This suggests that the core words in each record play dominant roles. Another aspect, Doc2Vec achieves good performances in AHC, top-*k*, and range query (top 3 or so). But it has poorer results in all other block generation methods, which are all Euclidean distance-based block generation

Table 6. Results of DBSCAN using different record embedding methods.

Embedding	Dataset						
	Cora Fα(%)	Notebook Fα(%)	Altosight Fα(%)	WDC_cameras Fα(%)	Abt-Buy Fα(%)	Amazon-Google2 Fα(%)	AVG Fα(%)
BERT	71.07	62.84	55.73	66.58	66.43	58.51	63.53
SBERT	**94.46**	64.77	**63.08**	**81.25**	**87.92**	**77.19**	78.11
Doc2Vec	48.12	17.07	11.75	1.66	0.91	6.71	14.37
fastText + Average	76.86	66.63	48.81	53.79	55.67	53.07	59.14
fastText + SIF	64.97	65.52	56.08	61.75	61.24	58.04	61.27
*BERT	87.15(l = 1)	**67.95(l = 8)**	59.61(l = 0)	71.46(l = 4)	68.63(l = 0)	59.30(l = 1)	69.02
*BERT + Schema	83.43(l = 3)	67.55(l = 2)	58.77(l = 3)	74.34(l = 2)	64.95(l = 4)	57.88(l = 3)	67.82
*SBERT	**94.46(l = 12)**	67.43(l = 8)	**63.08(l = 12)**	**81.25(l = 12)**	**87.92(l = 12)**	**77.19(l = 12)**	**78.56**
*SBERT + Schema	92.48(l = 12)	67.72(l = 0)	62.67(l = 12)	77.92(l = 3)	85.33(l = 12)	73.47(l = 12)	76.60

Table 7. Results of AP using different record embedding methods.

Embedding	Dataset						
	Cora Fα(%)	Notebook Fα(%)	Altosight Fα(%)	WDC_cameras Fα(%)	Abt-Buy Fα(%)	Amazon-Google2 Fα(%)	AVG Fα(%)
BERT	82.05	90.28	58.54	72.43	71.51	54.93	71.62
SBERT	**95.11**	88.28	72.65	**84.53**	**90.96**	**75.42**	84.49
Doc2Vec	53.43	68.73	44.26	54.20	4.93	11.01	39.43
fastText + Average	81.58	89.36	40.28	62.43	60.32	52.46	64.41
fastText + SIF	70.19	77.11	54.20	62.21	70.17	68.50	67.06
*BERT	89.96(l = 2)	92.10(l = 1)	69.97(l = 0)	74.30(l = 1)	71.51(l = 12)	54.93(l = 12)	75.46
*BERT + Schema	72.35(l = 12)	91.97(l = 3)	64.87(l = 1)	73.03(l = 12)	52.71(l = 11)	50.95(l = 11)	67.65
*SBERT	**95.11(l = 12)**	93.18(l = 0)	72.65(l = 12)	**84.53(l = 12)**	**90.96(l = 12)**	**75.42(l = 12)**	**85.31**
*SBERT + Schema	94.51(l = 12)	**93.34(l = 0)**	**75.88(l = 12)**	81.21(l = 12)	87.54(l = 12)	68.89(l = 12)	83.56

methods, so we infer that Doc2Vec is not suitable for block generation methods using Euclidean distance. All in all, SBERT shows its great advantage in record embedding, compared to other PLMs. In addition, this paper conducts experiments on BERT and SBERT which consider schema information. In most cases, schema information does not improve blocking results. Therefore, the schema information is not considered in this paper.

Table 8. Results of *k*-means using different record embedding methods.

Embedding	Dataset						
	Cora Fα(%)	Notebook Fα(%)	Altosight Fα(%)	WDC_cameras Fα(%)	Abt-Buy Fα(%)	Amazon-Google2 Fα(%)	AVG Fα(%)
BERT	82.80	92.62	60.52	69.64	53.26	58.49	69.56
SBERT	95.23	88.64	**81.53**	**87.31**	**94.33**	**82.11**	88.19
Doc2Vec	88.20	92.04	64.43	79.18	50.49	47.06	70.23
fastText + Average	80.60	90.43	51.64	73.37	65.54	54.49	69.35
fastText + SIF	82.01	89.93	48.44	66.68	82.33	66.89	72.71
*BERT	89.74(l = 1)	94.57(l = 5)	69.42(l = 1)	76.09(l = 0)	58.29(l = 1)	58.49(l = 12)	74.43
*BERT + Schema	79.74(l = 1)	94.22(l = 1)	57.56(l = 12)	74.99(l = 0)	32.78(l = 12)	40.35(l = 12)	63.27
*SBERT	**96.07(l = 0)**	**94.67(l = 4)**	**81.53(l = 12)**	**87.31(l = 12)**	**94.33(l = 12)**	**82.11(l = 12)**	**89.34**
*SBERT + Schema	93.40(l = 12)	94.52(l = 0)	78.81(l = 12)	84.43(l = 12)	92.33(l = 12)	79.12(l = 12)	87.10

Table 9. Results of top-*k* using different record embedding methods.

Embedding	Dataset						
	Cora Fα(%)	Notebook Fα(%)	Altosight Fα(%)	WDC_cameras Fα(%)	Abt-Buy Fα(%)	Amazon-Google2 Fα(%)	AVG Fα(%)
SBERT	93.98	91.95	**87.47**	**91.98**	**98.23**	**98.30**	93.65
Doc2Vec	91.32	91.21	82.32	89.64	95.28	90.72	90.08
fastText + Average	88.77	92.89	68.66	87.26	84.68	76.58	83.14
fastText + SIF	90.36	**93.29**	70.52	87.37	95.08	88.13	87.46
*BERT	92.17(l = 1)	92.48(l = 0)	82.29(l = 1)	90.09(l = 0)	90.68(l = 0)	80.57(l = 1)	88.05
*BERT + Schema	87.98(l = 1)	92.36(l = 0)	77.54(l = 1)	88.47(l = 0)	86.24(l = 12)	76.27(l = 12)	84.81
*SBERT	**94.98(l = 0)**	92.51(l = 0)	**87.47(l = 12)**	**91.98(l = 12)**	**98.23(l = 12)**	**98.30(l = 12)**	**93.91**
*SBERT + Schema	94.10(l = 0)	92.95(l = 0)	87.39(l = 12)	90.92(l = 12)	98.21(l = 12)	95.12(l = 12)	93.12

Layer Analysis of Sentence-BERT and BERT. We investigate effectiveness of each layer in SBERT and BERT for blocking on Cora, Altosight, and Amazon-Google2. We get output token embeddings from each layer of BERT and SBERT and apply an average operation to obtain record embeddings.

In Fig. 6, 7 and 8, for BERT, the highest Fα is often found in layers 0–3. Therefore, shallow layers are superior to other layers in most cases. A study [16] has shown that surface information features are in the shallow network, syntactic information features and semantic information features are in the middle and higher layer network. Surface

Table 10. Results of range query using different record embedding methods.

Embedding	Dataset						
	Cora Fα(%)	Notebook Fα(%)	Altosight Fα(%)	WDC_cameras Fα(%)	Abt-Buy Fα(%)	Amazon-Google2 Fα(%)	AVG Fα(%)
BERT	63.52	80.66	62.78	**74.54**	81.72	73.33	72.76
SBERT	93.55	96.25	80.90	**91.07**	**97.70**	**97.63**	92.85
Doc2Vec	89.13	95.46	78.33	87.66	92.37	85.86	88.14
fastText + Average	73.94	95.85	62.44	81.79	83.22	70.00	77.87
fastText + SIF	90.27	96.15	72.12	87.07	94.92	87.83	88.06
*BERT	87.62(l = 1)	96.50(l = 0)	75.61(l = 0)	85.91(l = 0)	88.58(l = 0)	74.82(l = 1)	84.84
*BERT + Schema	14.77(l = 12)	96.85(l = 3)	67.13(l = 12)	85.21(l = 0)	76.96(l = 2)	76.01(l = 12)	69.49
*SBERT	**93.64(l = 0)**	**97.43(l = 0)**	80.90(l = 12)	**91.07(l = 12)**	**97.70(l = 12)**	**97.63(l = 12)**	**93.06**
*SBERT + Schema	92.29(l = 12)	**97.43(l = 0)**	**83.45(l = 12)**	88.92(l = 0)	97.05(l = 12)	96.40(l = 12)	92.59

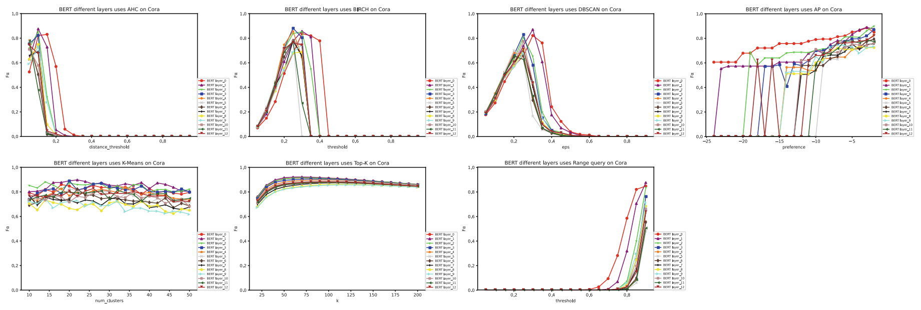

Fig. 6. Results of different BERT layers on Cora, layer = 0 indicates the input of BERT (token embedding + segment embedding + position embedding).

information is some intuitive information including sentence length etc. Syntactic information and semantic information include a great deal of grammar. In blocking, if two records are similar in surface information, then the two records should be placed in the same block. The syntactic and semantic details are more meaningful in matching of ER. For most experimental results, our intuition is correct. On Amazon-Google2, the optimal layers of BIRCH, AP, and *k*-means fall into other layers because Amazon-Google2's records are longer than other datasets. Longer records can result in common tokens being repeated in many records, and this is where syntax information is needed to help get better record embeddings. In addition, the shallower the layer is, the wider the focus of each token is [17]. Thus, each token in the shallow layers aggregates with each other, leading to drops in performance in the shallow layers.

In Fig. 9, 10 and 11, we observe that for SBERT, good blocking results can be seen at the 0-th layer and the 11-th layer, but the optimal layer is usually the 12-th layer. We

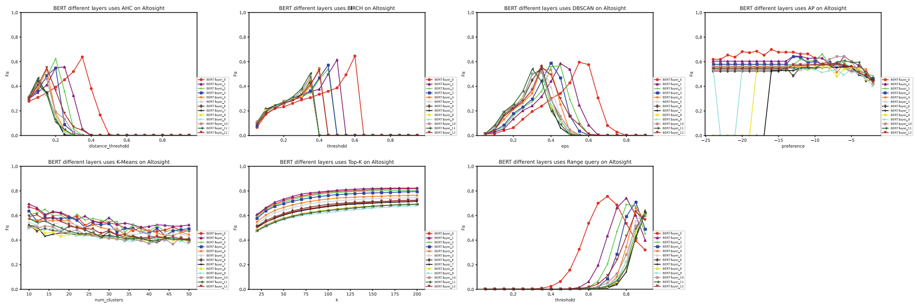

Fig. 7. Results of different BERT layers on Altosight.

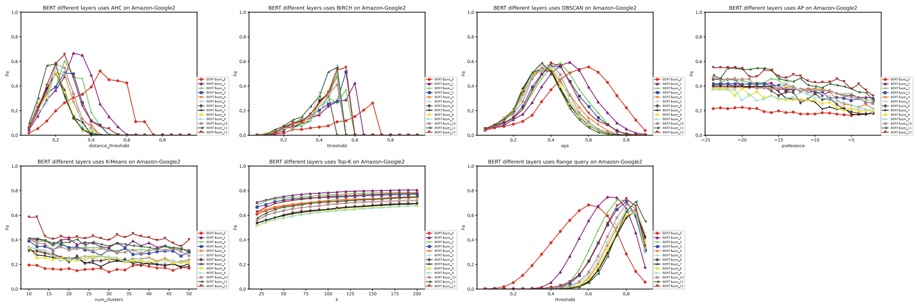

Fig. 8. Results of different BERT layers on Amazon-Google2.

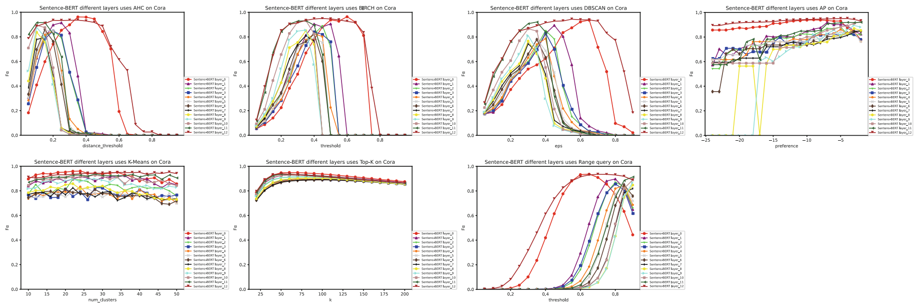

Fig. 9. Results of different SBERT layers on Cora, layer = 0 indicates the input of SBERT (token embedding + position embedding).

can draw a general conclusion: the optimal layer comes most from the last layer, and the first two layers generally outperform the middle layers. SBERT benefits from its whole-sentence learning, with the 12-th layer output showing a clear advantage on the blocking task. Of course, due to the nature of the blocking task itself, layers that focus on surface information can also work well.

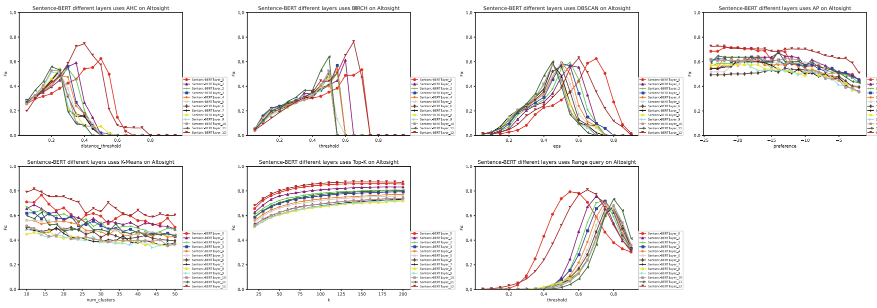

Fig. 10. Results of different SBERT layers on Altosight.

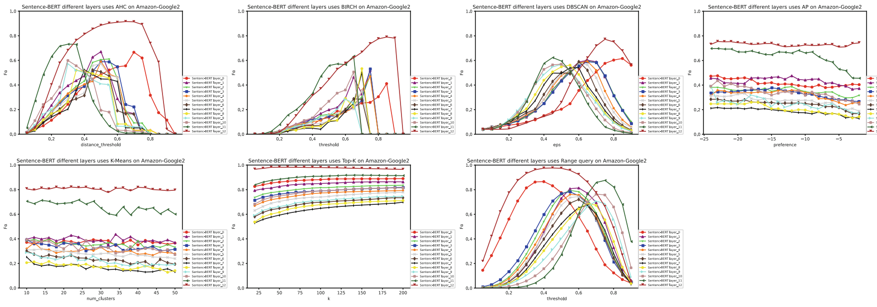

Fig. 11. Results of different SBERT layers on Amazon-Google2.

3.4 Analysis of Block Generation

Effectiveness of Block Generation Methods. In Table 2, we find that the best block generation methods in B-PLM are top-*k* and range query. The simple similarity search method achieves optimal performance. AHC performs better than other clustering algorithms. Firstly, AHC is a bottom-up clustering algorithm. Thus naturally more similar records are preferentially grouped into the same blocks. Also, AHC and *k*-means use cosine distances, whereas other clustering algorithms use Euclidean distances. In higher dimensional spaces, the Euclidean distance loses efficacy, whereas the cosine distance does not have this problem. In addition, *k*-means also achieves good results, especially on Cora, a more redundant dataset, and on Altosight, a dirty dataset, so we believe that *k*-means may be suitable for more difficult settings. BIRCH, DBSCAN, and AP perform poorly overall. To explore the characteristics of each block generation method more fully, we present the PC-RR curves for different block generation variants on six datasets, as shown in Fig. 12. The optimal block generation method is top-*k*, as the top-*k* method has a more slow decrease in RR as PC increases. The performance of range query is second only to that of top-*k*. For AHC, RR declines sharply if PC exceeds a certain value. In general, the top four block generation methods in terms of RC-RR curve are top-*k*, range query, AHC and *k*-means. Furthermore, we believe that DBSCAN is less suitable for blocking. Firstly, density measurement is inaccurate in high dimensional spaces where Euclidean distance's accuracy declines. Secondly, we cannot guarantee that cluster densities in a dataset are similar, but DBSCAN is only applicable to data with similar densities across different clusters. Finally, DBSCAN has several parameters that need to be adjusted, which decreases its usability for blocking.

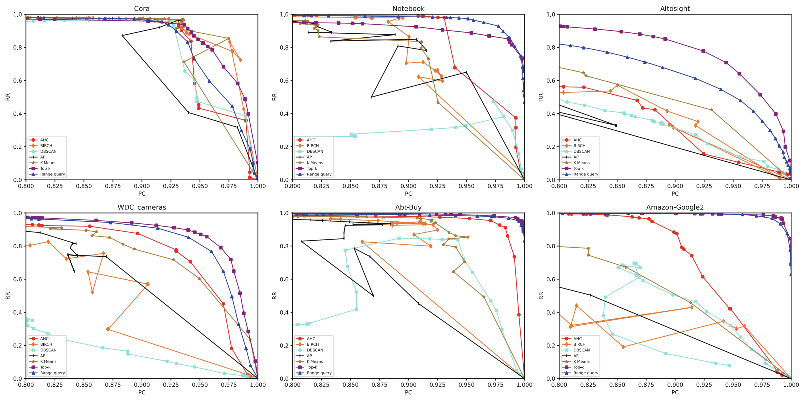

Fig. 12. PC-RR curves of different block generation methods on different datasets.

Parameter Sensitivity in Block Generation. We investigate parameters of different block generation methods. With SBERT, how parameter adjustments affect blocking qualities in different block generation methods are presented in Fig. 9, 10 and 11. AP, *k*-means and top-*k* are relatively insensitive to parameters, where blocking qualities on most layers change mildly with parameter adjustments. BIRCH is more sensitive to parameters. In general, top-*k*, *k*-means, and AP are robust to parameter changes, and their appropriate parameters are easy to catch.

4 Related Work

Non-deep learning blocking has been extensively studied [1]. There are six types of blocking techniques. 1. The Traditional Blocking method inserts all records with the same blocking key value (BKV) into the same block. 2. The basic idea of Sorted Neighborhood Blocking is to sort the database according to BKVs and then generate blocks by moving windows. 3. The basic idea of Q-Gram Based Blocking creation is to use q-grams for each BKV variation. 4. Suffix Array-Based Blocking is to insert the BKVs and their suffixes into a suffix array-based inverted index. 5. Canopy Clustering is based on the idea of using a computationally cheap clustering approach to create high-dimensional overlapping clusters. 6. String-Map-Based Blocking is based on a mapping of strings to objects in a multi-dimensional Euclidean space, where the distances between strings are preserved. Recently, DL-based blocking methods have emerged, which obtain record embedding by applying DL models, and then perform embedding-based block generation. DeepER [2] uses an RNN or LSTM to learn record embeddings for facilitating ER, and LSH is employed for blocking. BERT-ER [3] is based on BERT learn record embeddings and uses learnable hash functions for blocking. AutoBlock [4] uses fastText [10] to obtain token embeddings and learns how to combine token embeddings into record embeddings with labeled data. The above methods all use labeled data. Fabio Azzalini et al. [6] use fastText to obtain token embeddings and generate record embeddings by averaging or recurrent neural network (RNN). Then block generation is performed by LSH or clustering-based methods. DeepBlocker [5] utilizes fastText for token embeddings. To obtain record embeddings, DeepBlocker proposes four self-supervised tasks to accomplish the aggregation of token embeddings. Please refer to Sect. 3.1 for details of DeepBlocker. Most DL-based approaches still see blocking as part of ER, whereas B-PLM sees blocking as a separate task. B-PLM leverages the powerful linguistic expressiveness of PLM compared to the above methods, providing a wealth of external information for unsupervised blocking tasks.

5 Conclusion

This paper explores a design space of unsupervised blocking solutions based on PLMs. B-PLM is available as a baseline for unsupervised blocking. We compare seven representative solutions. The proposed solutions are experimentally proven to outperform the state-of-the-art DL-based blocking framework. The experimental results proved the advantages of B-PLM on textual and dirty datasets. Based on thorough experiments, we suggest recommended methods for record embedding (SBERT) and block generation (top-k). In the next step, we will consider some more advanced PLM models, and consider fine-tuning PLMs and dimension reduction.

Acknowledgments. This work is supported by the National Natural Science Foundation of China (Grant Nos. 62002262, 62172082, 62072086, 62072084).

References

1. Christen, P.: A survey of indexing techniques for scalable record linkage and deduplication. IEEE Trans. Knowl. Data Eng. **24**(9), 1537–1555 (2011)
2. Ebraheem, M., Thirumuruganathan, S., Joty, S., Ouzzani, M., Tang, N.: Distributed representations of tuples for entity resolution. Proc. VLDB Endowment **11**(11), 1454–1467 (2018)
3. Li, B., Miao, Y., Wang, Y., Sun, Y., Wang, W.: Improving the efficiency and effectiveness for BERT-based entity resolution. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, no. 15, pp. 13226–13233 (2021)
4. Zhang, W., Wei, H., Sisman, B., Dong, X. L., Faloutsos, C., Page, D.: AutoBlock: a hands-off blocking framework for entity matching. WSDM, pp. 744–752 (2020)
5. Thirumuruganathan, S., Li, H., Tang, N., Ouzzani, M., Govind, Y., Paulsen, D., Fung, G., Doan, A.: Deep learning for blocking in entity matching: a design space exploration. Proc. VLDB Endow. **14**(11), 2459–2472 (2021)
6. Azzalini, F., Jin, S., Renzi, M., Tanca, L.: Blocking techniques for entity linkage: a semantics-based approach. Data Sci. Eng. **6**(1), 20–38 (2020)
7. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)

8. Reimers, N., Gurevych, I.: Sentence-BERT: sentence embeddings using siamese BERT-networks. In: EMNLP-IJCNLP, pp. 3980–3990 (2019)
9. Le, Q., Mikolov, T.: Distributed representations of sentences and documents. In: ICML, pp. 1188–1196 (2014)
10. Bojanowski, P., Grave, E., Joulin, A., Mikolov, T.: Enriching word vectors with subword information. Trans. Assoc. Comput. Linguistics, **5**, 135–146 (2017)
11. Han, J., Pei, J., Tong, H.: Data Mining: Concepts and Techniques. Morgan Kaufmann (2022)
12. Zhang, T., Ramakrishnan, R., Livny, M.: BIRCH: an efficient data clustering method for very large databases. ACM SIGMOD Rec. **25**(2), 103–114 (1996). https://doi.org/10.1145/235968.233324
13. Frey, B.J., Dueck, D.: Clustering by passing messages between data points. Science **315**(5814), 972–976 (2007)
14. Arora, S., Liang, Y., Ma, T.: A simple but tough-to-beat baseline for sentence embeddings. In: ICLR (Poster) (2017)
15. Primpeli, A., Peeters, R., Bizer, C.: The WDC training dataset and gold standard for large-scale product matching. In: Companion Proceedings of the 2019 World Wide Web Conference, pp. 381–386 (2019)
16. Jawahar, G., Sagot, B., Seddah, D.: What does BERT learn about the structure of language?. ACL **1**, 3651–3657 (2019)
17. Clark, K., Khandelwal, U., Levy, O., Manning, C.D.: What does BERT look at? An analysis of bert's attention: BlackboxNLP@ACL, 276–286 (2019)