BlockBoost: Scalable and Efficient Blocking through Boosting

Thiago R. Ramos
USP

Rodrigo Schuller
IMPA

Alex A. Okuno
NYU

Lucas Nissenbaum
IMPA

Roberto I. Oliveira
IMPA

Paulo Orenstein
IMPA

Abstract

As datasets grow larger, matching and merging entries from different databases has become a costly task in modern data pipelines. To avoid expensive comparisons between entries, blocking similar items is a popular preprocessing step. In this paper, we introduce BlockBoost, a novel boosting-based method that generates compact binary hash codes for database entries, through which blocking can be performed efficiently. The algorithm is fast and scalable, resulting in computational costs that are orders of magnitude lower than current benchmarks. Unlike existing alternatives, BlockBoost comes with associated feature importance measures for interpretability, and possesses strong theoretical guarantees, including lower bounds on critical performance metrics like recall and reduction ratio. Finally, we show that BlockBoost delivers great empirical results, outperforming state-of-the-art blocking benchmarks in terms of both performance metrics and computational cost.

1 INTRODUCTION

With larger datasets and disparate data sources becoming more prevalent, properly identifying, integrating and linking entries across multiple datasets is now a crucial step in many data pipelines. This process of identifying and disambiguating entities within one or more datasets is known as entity matching, entity resolution or record linkage [Christen, 2012b]. Applications include merging health records [Clark, 2004, Kelman et al., 2002], aggregating census data [Winkler, 2006], identifying war casualties [Steorts and Shrivastava, 2018], detecting crimes [Jonas and Harper, 2006], cataloging bibliographic citations or business products, and matching genome sequences [Christen, 2012a]. By Rademacher Inequality [2], we have that with probability at least A fundamental issue in entity matching is the quadratic number of comparisons necessary between database items. Blocking [Steorts et al., 2014] is a popular technique to reduce the number of comparisons. It consists of blocking together items considered similar (in some metric) and only comparing entries within the same block. For example, if the goal is to match a list of customer purchase records to a list of customer accounts, blocks may be based on the customer's last name, or the ZIP code of their billing address, or all of these combined. If the blocks are well-crafted, this can significantly improve the speed and efficiency of the matching process.

A common way to create blocks is via hashing. A hash function yields a low-dimension binary representation of each entry, called the hash code. Unlike standard dimensionality-reduction methods, the fact that this representation is binary is important to ensure fast retrieval time, which is of the utmost importance for large databases [Andoni and Indyk, 2006, Charikar, 2002, Kulis and Darrell, 2009]. An efficient hash code maps similar candidates to the same hash code and dissimilar items to different hash codes, significantly reducing the associated number of comparisons and ensuring that similar items are indeed between these candidates.

However, devising good hash functions can be very hard. An important technique to create hash functions is locality-sensitive hashing (LSH). In locality-sensitive hashing [Andoni and Indyk, 2006, Har-Peled et al., 2012], hash codes are built in such a way that points

Proceedings of the 27th International Conference on Artificial Intelligence and Statistics (AISTATS) 2024, Valencia, Spain. PMLR: Volume 238.

that are close in some metric typically have similar hash codes. It has many interesting theoretical guarantees that are valid under broad circumstances. However, the fact that LSH is agnostic to the nature of the underlying data often leads to sub-optimal performance. To address this issue, there is a class of techniques known as learning to hash [Andoni and Beaglehole, 2021, Weiss et al., 2008a, Kulis and Darrell, 2009, Wang et al., 2018] that aims to improve the hashing efficiency by learning hash functions tailored for specific tasks, such as entity matching [Steorts et al., 2014]. While this approach can improve performance, it also introduces new challenges. For instance, methods like kernel LSH and TLSH maintain LSH's theoretical guarantees, but sacrifice scalability and efficiency [Kulis and Grauman, 2009, Jiang et al., 2014, Oliver et al., 2013]. A more recent alternative for blocking dispenses hashing altogether and is based on deep learning embeddings applied to entity matching [Thirumuruganathan et al., 2021a, Mudgal et al., 2018]. These state-of-the-art solutions use neural networks to learn feature representations that capture the underlying relationships between records, thus improving the accuracy and effectiveness of blocking.

In this paper, we propose a new blocking method called *BlockBoost*, that combines a boosting step that learns a pairwise similarity function and a hashing step on top of which blocking can be quickly performed (see Figure 1). Since boosting is a fast machine learning method with great out-of-the-box performance, BlockBoost works well for many different types of unstructured data. Furthemore, unlike many traditional blocking alternatives, it is possible to devise lower bounds on BlockBoost's performance in terms of relevant metrics such as recall and reduction ratio. This results in a data-driven technique that is efficient and scalable, with provable guarantees. Finally, BlockBoost achieves superior results to state-of-the-art deep learning solutions on multiple datasets.

Main Contributions. We introduce BlockBoost, a novel blocking algorithm that combines hashing and boosting with several features:

- Efficient data compression: by learning hash codes from data, BlockBoost is able to extract and the combine the most distinguishing features and automatically pick the right hash size; e.g., in one of the empirical examples considered, it obtains state-of-the-art results by compressing 9600 bits in the original features into a 150-bit hash. Also, hashing dimensions are ordered by importance, and can be trimmed for further compression;
- Speed: the training is quasi-linear in the entries, with linear prediction time; it is an order of magnitude faster than alternatives, it scales to millions of entries and runs well on CPUs;
- Simple tuning: BlockBoost has a single, easy-to-interpret hyperparameter;
- Theoretical results: unlike most blocking algorithms, BlockBoost has theoretical guarantees on its performance and lower bounds on popular metrics such as recall and reduction ratio;
- Interpretability: it is possible to interpret the contribution of each data feature to the final hashes by looking at importance measures derived from boosting; this is useful to identify the most distinguishing features available in the data;
- Empirical performance: BlockBoost outperforms state-of-the-art solutions on many canonical blocking datasets in terms of recall, reduction ratio and their harmonic average.

2 RELATED WORK

Due to its importance in data pipelines across many applications, entity matching [Winkler, 2004, Christophides et al., 2020, Elmagarmid et al., 2007] is a widely studied field. While blocking is an old idea [Fellegi and Sunter, 1969], it remains an active area of research [Papadakis et al., 2020].

Several blocking techniques are based on exact matches or certain blocking keys, such as attribute matching [Azzalini et al., 2020] and token blocking [O'Hare et al., 2019]. However, many real-world data possesses unnormalized or corrupted data, posing a serious challenge to such methods [Zhang et al., 2020]. Because BlockBoost learns attributes through boosting, it does not suffer from this problem.

There are also methods that do not require exact attribute matching for block creation. For example, the canopy clustering algorithm [McCallum et al., 2000] groups together items based on the similarity of certain fields using a clustering algorithm. However, it is slow and requires the tuning of several hyperparameters to obtain a competitive performance. BlockBoost, on the other hand, has a single hyperparameter.

Hashing-based blocking methods, such as LSH, are closer in spirit to our approach. They partition records to the same blocks if they share the same hash value, using a pre-specified random hash function. For example, in [Steorts et al., 2014, Steorts and Shrivastava, 2018], community detection techniques [Oliver et al., 2013]

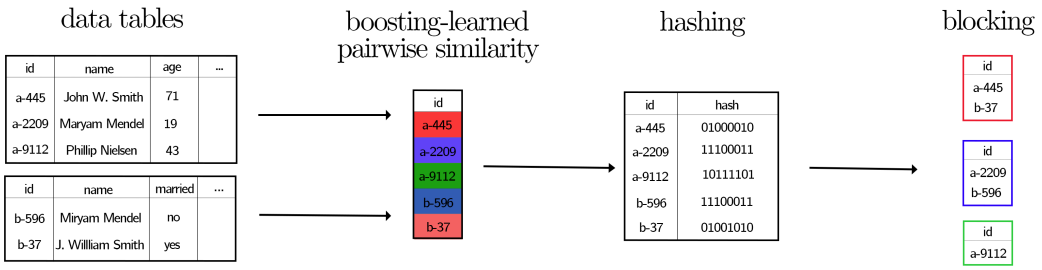


Figure 1: Overview of how BlockBoost performs blocking through (i) boosting and (ii) hashing.

and clustering algorithm [Paulevé et al., 2010] were used as a post-processing steps to LSH and lead to good results in entity matching. There are also learning to hash algorithms [Andoni and Beaglehole, 2021, Weiss et al., 2008a, Kulis and Darrell, 2009, Wang et al., 2018] that try to learn the hash functions via a training stage or refine already existing hash functions to lessen the correlations and redundancies between bits [Liu et al., 2024]. However, they are typically not employed for blocking due to scalability issues with learning overly complex algorithms [Kulis and Grauman, 2009, Jiang et al., 2014, Oliver et al., 2013]. BlockBoost, in contrast, can be orders of magnitude faster.

In recent years, there has been a growing interest in using deep learning embeddings to enhance the performance of blocking in entity matching [Thirumuruganathan et al., 2021a, Mudgal et al., 2018]. These approaches are oftentimes considered to be state-of-the-art, as they leverage neural networks to learn intricate feature representations that capture the inherent relationships between records.

Finally, other works, such as [Kim et al., 2020] and [Shakhnarovich, 2005], also employ boosting techniques for hashing. Still their setting significantly differs from ours as they do not consider blocking for entity matching. Indeed, [Kim et al., 2020] is concerned with distance functions in metric spaces and [Shakhnarovich, 2005] employ a gradient ascent algorithm with no generalization results.

3 BLOCKBOOST

Given datasets $\mathcal{A} := \{A_\ell\}_{\ell=1}^{N_\mathcal{A}}$ and $\mathcal{B} := \{B_r\}_{r=1}^{N_\mathcal{B}}$, both contained in a set $\mathcal{X}$, and a relationship between items $\sim_R$, we want to find pairs (A_ℓ, B_r) such that $A_\ell \sim_R B_r$, where we assume $\sim_R$ is unknown and must be learned. In our entity matching application, A_ℓ and B_r will correspond to entities in different datasets and we will say that $A_\ell \sim_R B_r$ if and only if A_ℓ and B_r are the same entity. For ease of presentation, our setup assumes two databases; however, the results hold for an arbitrary number of data collections (or one collection with many representations of the same item).

Our goal is to build a hash table for these items so that, given an item A_ℓ, one can find $B_r \sim_R A_\ell$ with as few table lookups as possible. To this end, we suppose we have access to a training sample,

$$\mathcal{S}_{\text{train},n} := \{((A_i, B_i), y_i) \in (\mathcal{A} \times \mathcal{B}) \times \{-1, 1\}, i \in [n]\},$$

such that, $y_i = 1$ if $A_i \sim_R B_i$ and -1 otherwise. This sample will be used in a training stage so our algorithm can learn a similarity classifier via a sample of similar/dissimilar items using boosting.

Our method consists of two steps:

Boosting step. Using boosting, we learn binary classifiers $\{k_t^*\}_{t=1}^T$ over $\mathcal{X}$, as well as convex weights $\{\alpha_t^*\}_{t=1}^T$ for these classifiers. Then, given items $A \in \mathcal{A}$ and $B \in \mathcal{B}$, we construct a similarity function $f^*(A, B) = \sum_{t=1}^T \alpha_t^* k_t^*(A) k_t^*(B)$. In this step, weights α_t^* are expected to be large when the product $k_t^*(A)k_t^*(B)$ correlates strongly with the similarity relation $A \sim_R B$, and consequently $f^*(A, B)$ is close to $+1$ when $A \sim_R B$ and $f^*(A, B)$ is close to -1 when $A \not\sim_R B$.

Hashing step. From the $\{\alpha_t^*, k_t^*\}_{t=1}^T$ learned in the previous step, various techniques can be employed to create hash codes for blocking. In this paper, we focus on using the similarity function learned in the boosting to create blocks via a weighted hamming distance between pairs. The Supplementary Material discusses another plausible option based on LSH [Andoni and Indyk, 2006, Har-Peled et al., 2012].

We now consider each of the above steps in further detail.

3.1 Boosting Step

Fix a family $\mathcal{K}$ of binary classifiers $k : \mathcal{X} \to \{-1, +1\}$ and a max number of iterations $T_{\max} \in \{1, 2, \dots\}$. To

Algorithm 1 Boosting step

Input: $\mathcal{S}_{\text{train},n} = ((A_i, B_i), y_i)_{i=1}^n$, max number of iterations $T_{\max} \in \mathbb{N}$, binary family $\mathcal{K}$

for $i \leftarrow 1$ to n **do**
  $Q_1(i) \leftarrow \frac{1}{n}$
end for
$t \leftarrow 1$
$T \leftarrow T_{\max}$
while $t \leq T_{\max}$ **do**
  $k_t^* \leftarrow$ classifier in $\mathcal{K}$ with smallest error $\varepsilon_t = \sum_{i=1}^n Q_t(i)\mathbf{1}_{[y_i k_t^*(A_i)k_t^*(B_i)<0]}$
  if $\varepsilon_t \geq 1/2$ **then**
    $T \leftarrow t-1$
    break
  else
    $\alpha_t' \leftarrow \frac{1}{2}\log\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$
    $Z_t \leftarrow 2\left[\varepsilon_t(1-\varepsilon_t)\right]^{1/2}$
    for $i \leftarrow 1$ to n **do**
      $Q_{t+1}(i) \leftarrow \frac{Q_t(i)\exp\left(-\alpha_t' y_i k_t^*(A_i)k_t^*(B_i)\right)}{Z_t}$
    end for
  end if
  $t \leftarrow t+1$
end while
for $t \leftarrow 1$ to T **do**
  $\alpha_t^* \leftarrow \frac{\alpha_t'}{\sum_{s=1}^T \alpha_s'}$
end for

Output: $(\alpha_t^*)_{t=1}^T$, $(k_t^*)_{t=1}^T$

find the functions $\{k_t^*\}_{t=1}^T \in \mathcal{K}$ and the convex weights $\{\alpha_t^*\}_{t=1}^T$, with $T \leq T_{\max}$, we use a boosting algorithm over our training sample $\mathcal{S}_{\text{train},n} = ((A_i, B_i), y_i)_{i=1}^n$; see Algorithm 1. Note that T, rather than $T_{\max}$, is the key determinant of the total number of binary classifiers produced by our method. As we will demonstrate in forthcoming discussions, it also plays a pivotal role in determining the ultimate level of compression achieved by our method, as it directly influences the number of bits required for the hash.

While Algorithm 1 is reminiscent of AdaBoost [Freund and Schapire, 1997], note we are optimizing a function that is quadratic over the chosen classifier. This is crucial for the hashing step, described in Section 3.2. Also, there are classifier families for which this optimization problem is feasible. An example is the set of decision stumps $\mathcal{H}_{\text{stumps}}$: for $x \in \mathbb{R}^d$, if $x_{(j)}$ indicates the j-th coordinate of $x \in \mathbb{R}^d$, then

$$\mathcal{H}_{\text{stumps}} = \left\{\mathbf{1}_{[x_{(j)}<\xi]} \cup \mathbf{1}_{[x_{(j)}\geq\xi]} \ : \ \xi \in \mathbb{R}, j \in [p]\right\}. \tag{1}$$

Intuitively, since the model is trying to predict matches and non-matches, we expect that, given items $A \in \mathcal{A}$ and $B \in \mathcal{B}$, the learned function

$$f^*(A, B) = \sum_{t=1}^T \alpha_t^* k_t^*(A) k_t^*(B), \tag{2}$$

will be a good similarity measure between A and B. That is, $f^*(A, B)$ should be close to $+1$ when $A \sim_R B$, and close to -1 otherwise. Notice that, as usual in boosting, larger weights α_t^* are given to the classifiers k_t^* that achieve the smallest values of boosted errors ε_t. That is, our algorithm naturally gives more weight to functions $k_t^*(A)k_t^*(B)$ that correlate more strongly to the similarity relation.

3.2 Hashing Step

We now use the convex weights $(\alpha_t^*)_{t=1}^T$ and the functions $(k_t^*)_{t=1}^T$ to construct hash functions that will be used for blocking. Our solution involves a straightforward calculation of a weighted Hamming distance over hash codes to create the blocks.

For each element $A \in \mathcal{X}$, we create a T-bit hash function g that is given by

$$g(A) = (k_1^*(A), \ldots, k_T^*(A)),$$

where T is the number of iterations used in the boosting step. Items (A, B) will be part of the same block if for a given small $\delta \in [0, 1]$,

$$f^*(A, B) = \sum_{i=1}^T \alpha_i^* k_t^*(A) k_t^*(B) \geq 1 - \delta \tag{3}$$

After creating the blocks, we can reduce direct comparisons only to pairs (A, B) in each block, sparing a number of unnecessary comparisons when declaring a match.

An additional benefit of the proposed hashing approach is that it allows for the expression in Equation 3 to be written as a weighted Hamming distance between the hashes $g^*(A)$ and $g^*(B)$,

$$\begin{aligned}\sum_{i=1}^T \alpha_i^* k_t^*(A) k_t^*(B) &= \sum_{i=1}^T \alpha_i^*(1 - |k_t^*(A) - k_t^*(B)|) \\ &= 1 - \sum_{i=1}^T \alpha_i^* |k_t^*(A) - k_t^*(B)|,\end{aligned}$$

since $\sum_{t=1}^T \alpha_t^* = 1$. Therefore, items (A, B) will be part of the same block only if

$$\sum_{i=1}^T \alpha_i^* |k_t^*(A) - k_t^*(B)| \leq \delta, \tag{4}$$

and this can be computed efficiently, as we discuss in Section 5.6.

4 THEORETICAL GUARANTEES

For our theoretical analysis, we assume we have two datasets $\mathcal{A} := \{A_\ell\}_{\ell=1}^{N_\mathcal{A}}$ and $\mathcal{B} := \{B_r\}_{r=1}^{N_\mathcal{B}}$, both contained in a set $\mathcal{X}$ and a notion of similarity $\sim_R$ between

items (A_ℓ, B_r). For entity matching problems, we assume that $A_\ell \sim_R B_r$ if and only if A_ℓ and B_r are the same entity. All proofs for the results in this section can be found in the Supplementary Material.

4.1 Performance Metrics

We first define traditional performance metrics for entity matching. The goal of our method is to ensure that, for each $A \in \mathcal{A}$, one can find all similar $B \in \mathcal{B}$ while doing as few pairwise comparisons as possible. This is made precise by the Recall and the Reduction Ratio (RR) metrics [Christen, 2012b]:

$$\text{Recall} := \frac{1}{|\mathcal{M}|} \sum_{(\ell,r)\in\mathcal{M}} \mathbf{1}_{[A_\ell \text{ and } B_r \text{ share a block}]}; \quad (5)$$

$$\text{RR} := 1 - \frac{1}{|\mathcal{N}|} \sum_{(\ell,r)\in\mathcal{N}} \mathbf{1}_{[A_\ell \text{ and } B_r \text{ share a block}]}, \quad (6)$$

where $\mathcal{N} := [N_\mathcal{A}] \times [N_\mathcal{B}]$ denotes all possible pairs and $\mathcal{M}$ denotes the set of matching pairs:

$$\mathcal{M} := \{(\ell, r) \in \mathcal{N}, A_\ell \sim_R B_r, (A_\ell, B_r) \in \mathcal{A} \times \mathcal{B}\}. \quad (7)$$

Recall, also known as pair completeness, measures the proportion of similar pairs that end up in the same block, whereas RR, also known as efficiency, measures the proportion of the $N_\mathcal{A} \cdot N_\mathcal{B}$ potential pairwise comparisons that are avoided. Ideally, we would like to find as many as possible matching pairs (Recall ≈ 1), while avoiding as many comparisons as possible (RR ≈ 1).

To be able to compare performance through a single real value, a commonly used metric in the blocking literature [Azzalini et al., 2020] is the the harmonic mean between Recall and RR, H(Recall, RR), given by:

$$\text{H}(\text{Recall}, \text{RR}) := 2 \cdot \frac{\text{Recall} \cdot \text{RR}}{\text{Recall} + \text{RR}}. \quad (8)$$

4.2 Performance Guarantees

The notion of margin plays a central role in our analysis. It corresponds to our intuition that f^* is a good estimator of the similarity relation $\sim_R$ if it confidently identifies matches and non-matches.

Definition 4.1 (θ-margin condition)**.** For a fixed $\theta > 0$, given classifiers $\{k_t^*\}_{t=1}^T$ and convex weights $\{\alpha_t^*\}_{t=1}^T$ we say that the similarity function f^* defined in (2) has θ-margin if, with probability at least $1-\eta$ over the choice of (A, B, y), it holds that

$$f^*(A, B) > +\theta, \text{ if } y = +1, \quad \text{and}$$
$$f^*(A, B) < -\theta, \text{ if } y = -1.$$

Our first theorem shows that the function f^* obtained in the boosting step via Algorithm 1, satisfies the θ-margin condition with high probability. The value of η depends on the Rademacher complexity [Bartlett and Mendelson, 2002] of the base classifiers $\mathcal{K}$ and the samples $\mathcal{S}_{\mathcal{A},n} := \{A_i\}_{i=1}^n$ and $\mathcal{S}_{\mathcal{B},n} := \{B_i\}_{i=1}^n$ given by

$$\mathfrak{R}_{\mathcal{S}_{\mathcal{A},n}}(\mathcal{K}) = \frac{1}{n}\mathbb{E}_\sigma \left[\sup_{k\in\mathcal{K}} \sum_{i=1}^n \sigma_i y_i k(A_i)\right],$$
$$\mathfrak{R}_{\mathcal{S}_{\mathcal{B},n}}(\mathcal{K}) = \frac{1}{n}\mathbb{E}_\sigma \left[\sup_{k\in\mathcal{K}} \sum_{i=1}^n \sigma_i y_i k(B_i)\right],$$

where $\sigma_1, \ldots, \sigma_n$ are independent Rademacher random variables, i.e., uniformly chosen in $\{-1, 1\}$.

Theorem 4.2 (Performance of the boosting step)**.** *With probability at least $1-\delta$, the function f^* corresponding to the output of Algorithm 1 satisfies the θ-margin condition with the value of $\eta := \eta_{train}(f^*, \mathcal{S}_{train,n}, \theta, \delta)$ given by:*

$$\eta := 2^T \prod_{t=1}^T \varepsilon_t^{1/2-\theta}(1-\varepsilon_t)^{\theta-1/2} + \frac{8}{\theta}\left(\mathfrak{R}_{\mathcal{S}_{\mathcal{A},n}}(\mathcal{K}) + \mathfrak{R}_{\mathcal{S}_{\mathcal{B},n}}(\mathcal{K})\right) + \sqrt{\frac{\log(1/\delta)}{2n}},$$

where $\varepsilon_t > 0$ are the errors defined in Algorithm 1.

Furthermore, if there exists $\gamma > 0$ such that for all $t \in [T]$, $\gamma \le (1/2 - \varepsilon_t)$ and $\theta \le 2\gamma$, then the first term in the right-hand side above decreases exponentially with T.

Intuitively, the product term in the definition of $\eta_{\text{train}}(f, \mathcal{S}_{\text{train},n}, \theta, \delta)$ is a margin bound over the training data. When $\varepsilon_t \le 1/2 - \gamma$ for all t, the term will decay exponentially fast in T for suitable choices of margin parameters θ. The other terms in $\eta_{\text{train}}(f, \mathcal{S}_{\text{train},n}, \theta, \delta)$ correspond to a generalization bound used for the test error.

Our next result shows that when Definition 4.1 holds, then the proposed hashing steps of BlockBoost produce high values of the Recall and RR metrics in expectation, with a suitable choice of hyperparameters.

Theorem 4.3 (Performance Weighted Hamming distance hashing)**.** *Consider databases $\mathcal{A}$ and $\mathcal{B}$ such that $|\mathcal{A}| = N_\mathcal{A}$, $|\mathcal{B}| = N_\mathcal{B}$ and let $\mathcal{M}$ be the set of matching pairs as in 7. For given $\theta > 0$ if the output f^* of Algorithm 1 satisfies the θ-margin condition and we set $1 - \theta > \delta$ in 3.2, then BlockBoost achieves*

$$\mathbb{E}\left[\text{RR}\right] \ge (1-\eta)\left(1 - \frac{|\mathcal{M}|}{N_\mathcal{A} \cdot N_\mathcal{B}}\right),$$
$$\mathbb{E}\left[\text{Recall}\right] \ge 1 - \eta$$

where expectations are with respect to the training step, and Recall and RR are defined in (5) and (6).

Note that, in entity matching problems, it is usually the case that $|\mathcal{M}| \ll N_{\mathcal{A}} \cdot N_{\mathcal{B}}$. Thus, the expected RR is close to 1, meaning that only a few comparisons have to be made.

4.3 Algorithmic Complexity and Speed

We now analyze the algoritmic complexity and speed of BlockBoost in each step.

Boosting. We use stump functions (1) as the family of base classifiers. As described in [Mohri et al., 2012], to determine the stump with the minimal weighted error at each round of boosting we can presort each component in $\mathcal{O}(n \log n)$ time with a total computational cost of $\mathcal{O}(nd \log n)$. For a given component, there are only $n+1$ possible distinct thresholds, since two thresholds between the same consecutive component values are equivalent. To find the best threshold at each round of boosting, all of these possible $n+1$ values can be compared, which can be done in $\mathcal{O}(n)$ time. Thus, the total computational complexity of the algorithm for T rounds of boosting is $\mathcal{O}(nd \log n + ndT)$.

Hashing and Blocking. Our binary hashing, with their low number of bits, allow for faster construction of candidate pairs compared to floating-point vectorizations. As a comparison, DeepBlocker's algorithm [Thirumuruganathan et al., 2021a], one of the fastest blocking alternatives, uses a NVIDIA V100 GPU and the well-optimized FAISS library, and still took 34 minutes for a dataset of 1 million entries [Thirumuruganathan et al., 2021b]. In contrast, BlockBoost achieved the same empirical performance in 2 minutes using a consumer-grade i7 CPU.

5 EXPERIMENTS

Our code is available at `https://github.com/thiagorr162/blockboost`. See the Supplementary Material for further details regarding the experiments.

5.1 Datasets

We make use of canonical blocking datasets that are sourced from a broad spectrum of domains and span a wide range of sizes as shown in Table 1. They are all publicly available and have been used in previous work on entity matching [Steorts et al., 2014, Steorts and Shrivastava, 2018, Thirumuruganathan et al., 2021a, Christen, 2012b, Christen, 2012a, Köpcke et al., 2010]. Further datasets are considered in the Supplementary Material.

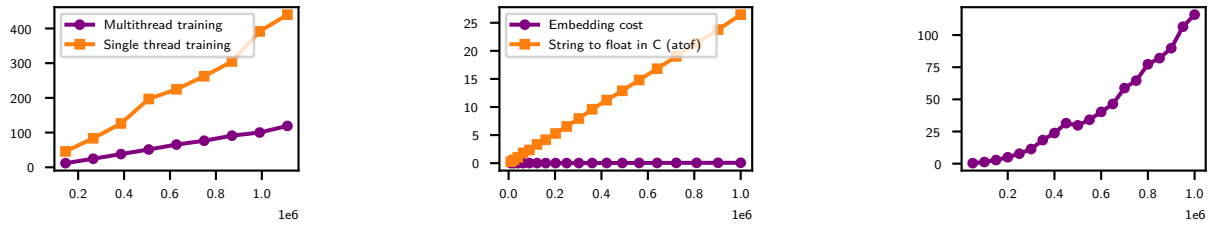
The datasets are divided into train, validation, and test sets, with 15%, 15%, and 70% of the total entries, respectively. Additionally, we ensure that the distribution of matches in each fold aligns with these proportions. The train set is used for models that require a training phase and the validation set is employed for adjusting hyperparameters, as needed. Note BlockBoost only has the single parameter δ in (4); in practice, T is chosen large enough that Algorithm 1 hits the stopping condition in line 8. Lastly, the test set is utilized to evaluate the performance metrics defined in Section 4.1.

We opted to use small training and validation proportions to more accurately represent a real-world scenario where a blocking model must be applied, but there are limited labeled examples available.

5.2 Benchmark Models

BlockBoost is compared against the following well-known methods for blocking in entity matching.

- *Canopy* [McCallum et al., 2000]: groups records into blocks based on the similarity of certain fields, using a clustering algorithm. For our experiments, we utilized the implementation in [CanopyByPython, 2018].
- *K-Means locality-sensitive hashing (KLSH)* [Paulevé et al., 2010]: uses k-means algorithm to construct a low-dimensional projection of the data. We use the code in [klsh, 2020] for the algorithm.
- *Transitive locality-sensitive hashing (TLSH)* [Oliver et al., 2013] uses a community detection technique to find similar entities. Our code follows [tlsh, 2020], which implements the work [Steorts et al., 2014].
- *Spectral hashing (Spect)* [Weiss et al., 2008b]: this learn to hash method uses a graph partitioning relaxation that is closely related to semantic hashing. We use the implementation in [LearnHash, 2018].
- *AGHasher (AG)* [Liu et al., 2011]: a learn to hash method automatically finds compact hash codes using graph-based neighborhood structure in the data. Our experiments follow the code in [AGHasher, 2022].
- *DeepBlocker* [Thirumuruganathan et al., 2021a]: a state-of-the-art blocking algorithm for entity matching using deep learning. We included the three distinct neural network architectures available as individual models: CTT, Autoencoder



(a) Training cost in second. (b) Embedding cost in seconds. (c) Blocking cost in seconds.

Figure 2: BlockBooster's cost in seconds on an i7 processor, as the size of the artificially created dataset based on `restaurant` gets larger. For $n = 10^6$, BlockBoost's blocking step takes at most 116 seconds, while DeepBlocker's CTT takes over 34 minutes on an NVIDIA V100 GPU.

Table 1: Baseline datasets for blocking, and compression (in bits) achieved by BlockBoost's hashes over the original set of features

DATASET	ENTITIES	MATCHES	TABLES	FEATURES	BLOCKBOOST BIT COMPRESSION
ABT_BUY	2,173	1,097	2	4	9×
AMZ_GG	4,589	1,300	2	5	23×
DBLP_ACM	4,908	2,224	2	4	8×
DBLP_SCH	66,879	5,347	2	4	5×
RESTAURANT	865	752	1	4	2×
RLDATA500	500	50	1	8	7×
RLDATA10K	10,000	1,000	1	8	6×
MUSICBRAINZ	19,375	10,000	5	7	3×
WM_AMZ	24,583	1,145	2	29	101×

(AE), and Hybrid, which integrates both the CTT and Autoencoder models. We use the official code repository provided by the authors [Thirumuruganathan et al., 2021b].

5.3 Vectorization

To ensure consistency with the original implementations of the baseline blocking models, we employed two distinct types of vectorization.

Shingling and MinHash. This vectorization is used for Canopy, TLSH and KSLH baseline models, as described in [Steorts et al., 2014, Steorts and Shrivastava, 2018] and implemented in [klsh, 2020, tlsh, 2020]. We first apply the shingling technique to construct a sparse numerical representation of our textual data. Then, we apply the MinHash algorithm [Broder et al., 2000, Steorts et al., 2014] to transform our sparse numerical information into a dense one.

SIF Embedding. We utilized this vectorization technique for both DeepBlocker and our own model, BlockBoost. SIF (Smooth Inverse Frequency) embedding [Arora et al., 2017] is a vector space model commonly used in NLP tasks, which assigns weights to each word vector based on their frequency in a reference corpus. This weighting scheme allows the resulting document vectors to capture the semantic meaning of the text while reducing the impact of frequent but less informative words.

5.4 Training Setup

For most of our benchmark models, training is performed on the training data fold. The exception is DeepBlocker, which does not require training data to work and instead relies on generating artificial training data. In particular, entities in the test set are compared against other, distinct entities in the test set (these will be assumed to be non-matches) or against a slightly perturbed version of the original entity (these will be declared matches). This allows the method to train on a significantly larger number of examples. Further details regarding this methodology are explained in detail in [Thirumuruganathan et al., 2021a]. BlockBoost works with either of these training setups. Below, we mimic DeepBlocker's approach so our method also does not require training data to work. We employ a ratio of 16 negative examples to each positive example in all our experiments.

5.5 Blocking Performance

Table 2 displays the performance of BlockBoost against all the benchmarks on the datasets from Table 1. Note each of DeepBlocker's possible configurations (CTT, AE, Hybrid) are considered separately. Overall, BlockBoost (BB) displays the best average harmonic mean of Recall and RR, and even when it is not the top-performing algorithm, it is generally close to it. The Supplementary Material includes the individual values of Recall and RR out of which the harmonic mean was derived. Note that only DeepBlocker, in its best possible network configuration, was able to rival BlockBoost's performance.

5.6 Computational Performance

Even when BlockBoost is trained on millions of pairs, it only takes a couple of minutes on a modern CPU, and, as Figure 2a shows, it scales well as n gets larger. This is crucial for a blocking algorithm, since the number of pairs to be matched in real-world examples can often become unwieldy (and are, after all, the main motivation for blocking algorithms). After that, the process of mapping entries from the initial vectorization to the binary embedding is so fast that it's dominated by the low level C function `atof`, which converts text to floats from the csv file (see Figure 2b).

The binary nature of our embedding, coupled with the low number of bits that the boosting produces, means that the set of candidate pairs can be constructed much faster than it would be possible with floating-point based vectorizations (see Table 1 for the bit compression Blockboost's hashes achieve versus the original feature set on each dataset). As a consequence, BlockBoost on CPU achieves a 17× speedup over DeepBlocker using an NVIDIA V100 GPU. To make sense of this, note that our binary embeddings, evaluated at Table 2, have an average size of 158 bits, and a maximum size of 256 bits. This is particularly useful for any x86 CPU with SSE4, since they can compute the number of 1s in a word in a single cycle. Modern GPUs also have instructions dedicated to this operation.

Finally, since the dimensions of the embedding are ordered by importance, the least significant bits can be trimmed to make the operations even faster.

5.7 Scalability

The primary motivation for employing blocking methods is to effectively handle large datasets. Here, we present a comparative analysis of timing between BlockBoost and baseline models using the `musicbrainz` dataset across varying sizes: 20k, 200k, and 2m entries, with BlockBoost emerging as the most scalable option by a significant margin.

For the dataset comprising 2 million entries, only BlockBoost managed to generate predictions within the allocated time (11 hours) and memory constraints (32GB). Table 3 displays performance metrics across increasing dataset sizes (20k, 200k, 2m), underscoring a key advantage of BlockBoost: not only does it deliver high accuracy, but it also operates at orders of magnitude faster speeds compared to other benchmarks.

5.8 Interpretability

At each iteration $t = 1, \ldots, T$ of the boosting process, BlockBoost produces a stump function k_t^* and a weight α_t^*. The weight can be interpreted as an indication of how significant the stump is for matching entities. Additionally, as the stump is defined by a projected feature j_t^* and a threshold ξ_t^*, the weight also indicates how strongly such feature correlates with the similarity relation between items. Thus, the α_t^* can be understood as giving an automatic feature importance.

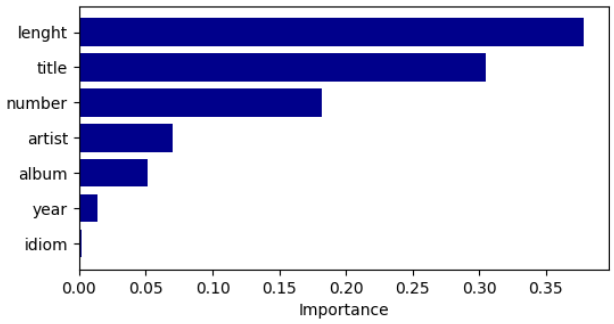


Figure 3: Feature relevance identified by BlockBoost during the boosting step for the `musicbrainz` dataset.

This can be very helpful when interpreting which features carry most of the signal for finding matches. This also allows for discarding irrelevant features for added speed, as well as simple attribute matching based on the selected ones. As an example, Figure 3 shows the importance of some features in the `musicbrainz` dataset when each one is vectorized separately. Here, importance is understood by the sum of weights associated to each feature.

Intuitively, if someone intends to correlate songs from various databases, BlockBoost considers the length and title as the most critical attributes. The track number on a CD emerges as a surprisingly reliable third option. This is because it's a numeric value and is typically automatically extracted, whereas other entries involve textual data and potential manual input. Notably, by generating hashes exclusively from the concatenation of the length and number (both numeric

Table 2: Comparison table of the evaluation metric H(Recall, RR), defined in 8, with the best model per dataset in bold. BlockBoost (BB) has the best performance overall.

Dataset	BB	Canopy	KLSH	TLSH	Spect	AG	CTT	AE	Hybrid
abt_buy	**0.911**	0.761	0.365	0.625	0.263	0.503	0.907	0.817	0.822
amz_gg	**0.877**	0.605	0.515	0.281	0.518	0.539	0.810	0.849	0.849
dblp_acm	0.993	0.850	0.895	0.861	0.662	0.696	0.993	0.996	**0.998**
dblp_sch	0.989	0.891	0.691	0.543	0.602	0.670	**0.991**	0.980	0.983
restaurant	0.988	0.785	0.937	0.838	0.519	0.728	**0.997**	0.997	0.997
rldata500	**0.992**	0.829	0.969	0.982	0.691	0.717	0.966	0.966	0.966
rldata10k	**0.999**	0.929	0.926	0.987	0.755	0.800	0.957	0.928	0.926
musicbrainz	0.991	0.101	0.944	0.950	0.662	0.737	**0.994**	0.992	0.992
wm_amz	**0.943**	0.017	0.495	0.005	0.577	0.558	**0.943**	0.917	0.942
average	**0.965**	0.641	0.749	0.675	0.583	0.660	0.951	0.938	0.942

Table 3: Benchmark against baseline models in the `musicbrainz` dataset, with 20k, 200k and 2m entries. Instances exceeding 11 hours are categorized as Out of Time (OOT), while those surpassing 32gb are designated as Out of Memory (OOM). BlockBoost-1bi includes computational optimizations, as detailed in Section D of the Supplementary Material.

Model	Time 20k	Time 200k	Time 2m	H(Recall, RR) 2m
blockboost-1bi	0.55 sec	5.61 sec	1 min 55 sec	0.9887
blockboost	4.3 sec	45.4 sec	14 min 76 sec	0.9895
deepblocker	12 min 35 sec	2 hrs 17 min 57 sec	OOM	OOM
tlsh	2 min 18 sec	52 min 23 sec	OOT	OOT
klsh	14 min 41 sec	OOT	OOT	OOT
canopy	14 min 10 sec	OOM	OOM	OOM

fields, although not an immediately obvious choice), one can achieve a high blocking performance with a H(Recall, RR) close to 0.91.

6 CONCLUSION

This paper introduces BlockBoost, a novel blocking method for entity resolution that is data-driven, efficient, and scalable. The results show that combining boosting techniques with hashing leads to a powerful blocking method that empirically outperforms state-of-the-art hashing and deep learning models in terms of scalability and performance over several canonical blocking datasets.

Beyond speed and efficiency, the algorithm is also interpretable and theoretically sound. It outputs weights that reflect variable importance measures, providing users with insight into the most relevant data features for the blocking process. BlockBoost also comes with guaranteed lower bounds on each performance through an extension of margin theory results.

Finally, BlockBoost is able to generate hashes whose size are automatically determined in a data-dependent fashion, leading to great compression and fast retrieval time. Since the dimensions of the hash are ordered by importance, one can trim the hashes for added time and better compression.

As a possible future avenue for this line of research, we believe BlockBoost can be adapted to many other applications beyond entity matching, such as image retrieval and similarity search, while carrying the same advantages as presented above.

References

[AGHasher, 2022] AGHasher (2022). Repository: Aghasher. https://github.com/dstein64/aghasher.

[Andoni and Beaglehole, 2021] Andoni, A. and Beaglehole, D. (2021). Learning to hash robustly, guaranteed.

[Andoni and Indyk, 2006] Andoni, A. and Indyk, P. (2006). Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 459–468.

[Arora et al., 2017] Arora, S., Liang, Y., and Ma, T. (2017). A simple but tough-to-beat baseline for sentence embeddings. In *5th International Conference*

on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. OpenReview.net.

[Azzalini et al., 2020] Azzalini, F., Jin, S., Renzi, M., and Tanca, L. (2020). Blocking techniques for entity linkage: A semantics-based approach. *Data Sci. Eng.*, 6:20–38.

[Bartlett and Mendelson, 2002] Bartlett, P. and Mendelson, S. (2002). Rademacher and Gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3(Nov):463–482.

[Broder et al., 2000] Broder, A. Z., Charikar, M., Frieze, A. M., and Mitzenmacher, M. (2000). Min-wise independent permutations. *Journal of Computer and System Sciences*, 60(3):630–659.

[CanopyByPython, 2018] CanopyByPython (2018). Repository: Canopy. https://github.com/AlanConstantine/CanopyByPython.

[Charikar, 2002] Charikar, M. S. (2002). Similarity estimation techniques from rounding algorithms. In *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*, STOC '02, page 380–388, New York, NY, USA. Association for Computing Machinery.

[Christen, 2012a] Christen, P. (2012a). *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Springer Publishing Company, Incorporated.

[Christen, 2012b] Christen, P. (2012b). A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Transactions on Knowledge and Data Engineering*, 24(9):1537–1555.

[Christophides et al., 2020] Christophides, V., Efthymiou, V., Palpanas, T., Papadakis, G., and Stefanidis, K. (2020). An overview of end-to-end entity resolution for big data. *ACM Comput. Surv.*, 53(6).

[Clark, 2004] Clark, D. E. (2004). Practical introduction to record linkage for injury research. *Injury Prevention*, 10(3):186–191.

[Elmagarmid et al., 2007] Elmagarmid, A. K., Ipeirotis, P. G., and Verykios, V. S. (2007). Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16.

[Fellegi and Sunter, 1969] Fellegi, I. P. and Sunter, A. B. (1969). A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210.

[Freund and Schapire, 1997] Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139.

[Har-Peled et al., 2012] Har-Peled, S., Indyk, P., and Motwani, R. (2012). Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory of Computing*, 8(14):321–350.

[Jiang et al., 2014] Jiang, K., Que, Q., and Kulis, B. (2014). Revisiting kernelized locality-sensitive hashing for improved large-scale image retrieval.

[Jonas and Harper, 2006] Jonas, J. and Harper, J. C. (2006). Effective counterterrorism and the limited role of predictive data mining.

[Kelman et al., 2002] Kelman, C., Bass, J., and Holman, C. (2002). Research use of linked health data - a best practice protocol. *Australian and New Zealand journal of public health*, 26:251–5.

[Kim et al., 2020] Kim, S., Yang, H., and Kim, M. (2020). Boosted locality sensitive hashing: Discriminative binary codes for source separation. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 106–110.

[klsh, 2020] klsh (2020). Repository: Klsh. https://github.com/cleanzr/klsh.

[Köpcke et al., 2010] Köpcke, H., Thor, A., and Rahm, E. (2010). Evaluation of entity resolution approaches on real-world match problems. *Proc. VLDB Endow.*, 3(1–2):484–493.

[Kulis and Darrell, 2009] Kulis, B. and Darrell, T. (2009). Learning to hash with binary reconstructive embeddings. In Bengio, Y., Schuurmans, D., Lafferty, J., Williams, C., and Culotta, A., editors, *Advances in Neural Information Processing Systems*, volume 22. Curran Associates, Inc.

[Kulis and Grauman, 2009] Kulis, B. and Grauman, K. (2009). Kernelized locality-sensitive hashing for scalable image search. In *2009 IEEE 12th International Conference on Computer Vision*, pages 2130–2137.

[LearnHash, 2018] LearnHash (2018). Repository: Learnhash. https://github.com/galad-loth/LearnHash.

[Liu et al., 2024] Liu, H., Zhou, W., Wu, Z., Zhang, S., Li, G., and Li, X. (2024). Refining codes for locality sensitive hashing. *IEEE Transactions on Knowledge and Data Engineering*, 36(3):1274–1284.

[Liu et al., 2011] Liu, W., Wang, J., Kumar, S., Chang, S.-F., and Scheffer, T. (2011). Hashing with graphs proceedings of the 28th international conference on machine learning. *ICML 2011*, pages 1–8.

[McCallum et al., 2000] McCallum, A., Nigam, K., and Ungar, L. H. (2000). Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, page 169–178, New York, NY, USA. Association for Computing Machinery.

[Mohri et al., 2012] Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2012). *Foundations of Machine Learning*. The MIT Press.

[Mudgal et al., 2018] Mudgal, S., Li, H., Rekatsinas, T., Doan, A., Park, Y., Krishnan, G., Deep, R., Arcaute, E., and Raghavendra, V. (2018). Deep learning for entity matching: A design space exploration. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, page 19–34, New York, NY, USA. Association for Computing Machinery.

[O'Hare et al., 2019] O'Hare, K., Jurek-Loughrey, A., and Campos, C. d. (2019). *A Review of Unsupervised and Semi-supervised Blocking Methods for Record Linkage*, pages 79–105. Springer International Publishing, Cham.

[Oliver et al., 2013] Oliver, J., Cheng, C., and Chen, Y. (2013). Tlsh – a locality sensitive hash. In *2013 Fourth Cybercrime and Trustworthy Computing Workshop*, pages 7–13.

[Papadakis et al., 2020] Papadakis, G., Skoutas, D., Thanos, E., and Palpanas, T. (2020). Blocking and filtering techniques for entity resolution: A survey. *ACM Comput. Surv.*, 53(2).

[Paulevé et al., 2010] Paulevé, L., Jégou, H., and Amsaleg, L. (2010). Locality sensitive hashing: a comparison of hash function types and querying mechanisms. *Pattern Recognition Letters*, 31(11):1348–1358.

[Shakhnarovich, 2005] Shakhnarovich, G. (2005). Learning task-specific similarity.

[Steorts and Shrivastava, 2018] Steorts, R. C. and Shrivastava, A. (2018). Probabilistic blocking with an application to the syrian conflict.

[Steorts et al., 2014] Steorts, R. C., Ventura, S. L., Sadinle, M., and Fienberg, S. E. (2014). A comparison of blocking methods for record linkage.

[Thirumuruganathan et al., 2021a] Thirumuruganathan, S., Li, H., Tang, N., Ouzzani, M., Govind, Y., Paulsen, D., Fung, G., and Doan, A. (2021a). Deep learning for blocking in entity matching: A design space exploration. *Proc. VLDB Endow.*, 14(11):2459–2472.

[Thirumuruganathan et al., 2021b] Thirumuruganathan, S., Li, H., Tang, N., Ouzzani, M., Govind, Y., Paulsen, D., Fung, G., and Doan, A. (2021b). Repository: Deepblocker. https://github.com/qcri/DeepBlocker.

[tlsh, 2020] tlsh (2020). Repository: Tlsh. https://github.com/cleanzr/tlsh.

[Wang et al., 2018] Wang, J., Zhang, T., song, j., Sebe, N., and Shen, H. T. (2018). A survey on learning to hash. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):769–790.

[Weiss et al., 2008a] Weiss, Y., Torralba, A., and Fergus, R. (2008a). Spectral hashing. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems*, volume 21. Curran Associates, Inc.

[Weiss et al., 2008b] Weiss, Y., Torralba, A., and Fergus, R. (2008b). Spectral hashing. *Advances in neural information processing systems*, 21.

[Winkler, 2004] Winkler, W. E. (2004). Methods for evaluating and creating data quality. *Information Systems*, 29(7):531–550. Data Quality in Cooperative Information Systems.

[Winkler, 2006] Winkler, W. E. (2006). Overview of record linkage and current research directions. Technical report, BUREAU OF THE CENSUS.

[Zhang et al., 2020] Zhang, W., Wei, H., Sisman, B., Dong, X. L., Faloutsos, C., and Page, D. (2020). AutoBlock. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. ACM.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [✓**Yes**/No/Not Applicable]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [✓**Yes**/No/Not Applicable]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [✓**Yes**/No/Not Applicable]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [✓**Yes**/No/Not Applicable]
   (b) Complete proofs of all theoretical results. [✓**Yes**/No/Not Applicable]
   (c) Clear explanations of any assumptions. [✓**Yes**/No/Not Applicable]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [✓**Yes**/No/Not Applicable]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [✓**Yes**/No/Not Applicable]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [✓**Yes**/No/Not Applicable]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [✓**Yes**/No/Not Applicable]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [✓**Yes**/No/Not Applicable]
   (b) The license information of the assets, if applicable. [✓**Yes**/No/Not Applicable]
   (c) New assets either in the supplemental material or as a URL, if applicable. [✓**Yes**/No/Not Applicable]
   (d) Information about consent from data providers/curators. [Yes/No/✓**Not Applicable**]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Yes/No/✓**Not Applicable**]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Yes/No/✓**Not Applicable**]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Yes/No/✓**Not Applicable**]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Yes/No/✓**Not Applicable**]

Supplementary Material

A PROOFS FOR SECTION 4

Proof of Theorem 4.2. We need to define the following sets:

$$\mathcal{K}_2 := \{f_k : (A, B) \mapsto k(A)k(B), k \in \mathcal{K}\},$$

and the convex hull $\text{conv}(\mathcal{K}_2)$ of $\mathcal{K}_2$ given by,

$$\begin{aligned}\text{conv}(\mathcal{K}_2) &:= \left\{ f : (A, B) \mapsto \sum_{t=1}^{T} \alpha_t f_{k_t}(A, B) : T \geq 1, \alpha_t \geq 0, f_{k_t} \in \mathcal{K}_2, \sum_{t=1}^{T} \alpha_t = 1 \right\} \\ &= \left\{ f : (A, B) \mapsto \sum_{t=1}^{T} \alpha_t k_t(A)k_t(B) : T \geq 1, \alpha_t \geq 0, k_t \in \mathcal{K}, \sum_{t=1}^{T} \alpha_t = 1 \right\}.\end{aligned}$$

To show that the output of Algorithm 1 indeed satisfies Condition 4.1 we prove the following stronger result.

Theorem A.1. *Consider an iid sample $\mathcal{S}_{train,n} = ((a_i, b_i), y_i)_{i=1}^n$ with $((a_i, b_i), y_i)$ drawn from P. Then, given $\theta \in (0, 1)$ and $\delta \in (0, 1)$, with probability at least $1 - \delta$, for any $f \in conv(\mathcal{K}_2)$*

$$P[yf(A, B) \leq \theta] \leq \eta_{train}(f, \mathcal{S}_{train,n}, \theta, \delta),$$

where

$$\eta_{train}(f, \mathcal{S}_{train,n}, \theta, \delta) := \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}_{[y_i f(a_i,b_i) \leq 2\theta]} + \frac{8}{\theta} \mathfrak{R}_{\mathcal{S}_{train,n}}(\mathcal{K}) + \sqrt{\frac{\log(1/\delta)}{2n}}.$$

Proof. First, consider the surrogate margin loss function given by:

$$\varphi_\theta(x) = \min\left(1, \max\left(1 - \frac{x}{\theta}, 0\right)\right).$$

and the following set:

$$\Phi_\theta := \{\varphi_{\theta,f} : ((a,b),y) \mapsto \varphi_\theta(yf(a,b) - \theta) : f \in \text{conv}(\mathcal{K}_2)\}.$$

By Rademacher Inequality [3], we have that with probability at least $1-\delta$, for all $f \in \text{conv}(\mathcal{K}_2)$:

$$\mathbb{E}\left[\varphi_\theta(yf(A,B) - \theta)\right] \leq \frac{1}{n}\sum_{i=1}^{n} \varphi_\theta(y_i f(a_i,b_i) - \theta) + 2\mathfrak{R}_{\mathcal{S}_{\text{train},n}}(\Phi_\theta) + \sqrt{\frac{\log(1/\delta)}{2n}}.$$

Using the fact that $\mathbf{1}_{[x\leq\theta]} \leq \varphi_\theta(x-\theta)$, we have that with probability at least $1-\delta$, for all $f \in \text{conv}(\mathcal{K}_2)$

$$P[yf(a,b) \leq \theta] \leq \frac{1}{n}\sum_{i=1}^{n} \varphi_\theta(y_i f(a_i,b_i) - \theta) + 2\mathfrak{R}_{\mathcal{S}_{\text{train},n}}(\Phi_\theta) + \sqrt{\frac{\log(1/\delta)}{2n}}.$$

Since φ_θ is $1/\theta$-Lipschitz, by Talagrand's Lemma and the fact that $\mathfrak{R}_{\mathcal{S}_{\text{train},n}}(\text{conv}(\mathcal{K}_2)) = \mathfrak{R}_{\mathcal{S}_{\text{train},n}}(\mathcal{K}_2)$ [1], we have with probability at least $1-\delta$, for all $f \in \text{conv}(\mathcal{K}_2)$:

$$P[yf(A,B) \leq \theta] \leq \frac{1}{n}\sum_{i=1}^{n} \varphi_\theta(y_i f(a_i,b_i) - \theta) + \frac{2}{\theta}\mathfrak{R}_{\mathcal{S}_{\text{train},n}}(\mathcal{K}_2) + \sqrt{\frac{\log(1/\delta)}{2n}}.$$

Using the fact that $\varphi_\theta(x-\theta) \leq \mathbf{1}_{[x\leq 2\theta]}$, with probability at least $1-\delta$, for all $f \in \text{conv}(\mathcal{K}_2)$:

$$P[yf(A,B) \leq \theta] \leq \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{[y_i f(a_i,b_i)\leq 2\theta]} + \frac{2}{\theta}\mathfrak{R}_{\mathcal{S}_{\text{train},n}}(\mathcal{K}_2) + \sqrt{\frac{\log(1/\delta)}{2n}}.$$

Now we just need to bound $\mathfrak{R}_{\mathcal{S}_{\text{train},n}}(\mathcal{K}_2)$ in terms of $\mathfrak{R}_{\mathcal{S}_{\text{train},n}}(\mathcal{K})$ so our final result depends only on the Rademacher complexity of the family $\mathcal{K}$ which is usually known. But note that

$$\begin{aligned}
\mathfrak{R}_{\mathcal{S}_{\text{train},n}}(\mathcal{K}_2) &= \frac{1}{n}\mathbb{E}_\sigma\left[\sup_{k\in\mathcal{K}_2}\sum_{i=1}^{n}\sigma_i y_i k(a_i)k(b_i)\right] \\
&= \frac{1}{n}\mathbb{E}_\sigma\left[\sup_{k\in\mathcal{K}}\sum_{i=1}^{n}\sigma_i y_i k(a_i)k(b_i)\right] \\
&\leq \frac{1}{n}\mathbb{E}_\sigma\left[\sup_{k_1,k_2\in\mathcal{K}}\sum_{i=1}^{n}\sigma_i y_i k_1(a_i)k_2(b_i)\right] \\
&= \frac{1}{n}\mathbb{E}_\sigma\left[\sup_{k_1,k_2\in\mathcal{K}}\sum_{i=1}^{n}\sigma_i k_1(a_i)k_2(b_i)\right] \\
&= \frac{1}{n}\mathbb{E}_\sigma\left[\sup_{k_1,k_2\in\mathcal{K}}\sum_{i=1}^{n}\sigma_i\left(1 - \frac{(k_1(a_i)-k_2(b_i))^2}{2}\right)\right] \\
&= 0 + \frac{1}{n}\mathbb{E}_\sigma\left[\sup_{k_1,k_2\in\mathcal{K}}\sum_{i=1}^{n}\sigma_i\frac{(k_1(a_i)-k_2(b_i))^2}{2}\right] \\
&= \frac{1}{2n}\mathbb{E}_\sigma\left[\sup_{k_1,k_2\in\mathcal{K}}\sum_{i=1}^{n}\sigma_i L\left(k_1(a_i)-k_2(b_i)\right)\right]
\end{aligned}$$

where,

$$L(x) = \begin{cases} x^2, \text{ if } x \in [-2,2] \\ 4, \text{ otherwise.} \end{cases}$$

Since L is 4-Lipschitz, by Talagrand's Lemma, we have that, with probability at least $1-\delta$, for all $f \in \text{conv}(\mathcal{K}_2)$

$$P[yf(A,B) \leq \theta] \leq \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{[y_i f^*(a_i,b_i) \leq 2\theta]} + \frac{8}{\theta}\left(\mathfrak{R}_{\mathcal{S}_{\mathcal{A},n}}(\mathcal{K}) + \mathfrak{R}_{\mathcal{S}_{\mathcal{B},n}}(\mathcal{K})\right) + \sqrt{\frac{\log(1/\delta)}{2n}}.$$

□

B DATASET DETAILS

Below, we provide an overview of each database utilized in our research paper. In all the datasets, every entry includes both record information and a distinct record ID, which serves as a unique identifier for each entry across the entire database, encompassing all potential tables. Additionally, each entry is associated with an entity ID, allowing us to identify and group together entities that are identical, regardless of their location in different tables.

The code necessary to download and process each dataset can be accessed from our GitHub repository `https://github.com/thiagorr162/blockboost`.

`abt_buy`

This dataset contains name, description, manufacturer and price of product data from `abt.com` and `buy.com`. It is available at `https://dbs.uni-leipzig.de/research/projects/object_matching/benchmark_datasets_for_entity_resolution`.

`amz_gg`

This dataset contains name, description, manufacturer and price product data from Amazon and Google. It can be downloaded via `https://dbs.uni-leipzig.de/research/projects/object_matching/benchmark_datasets_for_entity_resolution`.

`dblp_acm`

This dataset contains title, authors, venue and year information of bibliographic data from DBLP and ACM. We downloaded the dataset from `https://dbs.uni-leipzig.de/research/projects/object_matching/benchmark_datasets_for_entity_resolution`.

`dblp_sch`

This dataset contains title, authors, venue and year information of bibliographic data from DBLP and Google Scholar. It is available at `https://dbs.uni-leipzig.de/research/projects/object_matching/benchmark_datasets_for_entity_resolution`.

`musicbrainz`

This dataset contains number, title, length, artist, album, year and language information of music data from MusicBrainz. This dataset can be found at `https://dbs.uni-leipzig.de/research/projects/object_matching/benchmark_datasets_for_entity_resolution`.

`restaurant`

This dataset contains name, address, location and cuisine type of restaurants data. It can be downloaded from `https://github.com/cleanzr/restaurant`.

`rldata`

These datasets contain individuals' first and last names, as well as their birth year, birth month, and birth day. It is available at `https://github.com/cran/RecordLinkage/`.

`wm_amz`

This dataset comprises a wide range of product information from both Walmart and Amazon. It includes details such as brand, title, shelf description, short and long descriptions, model number, weight, and various other relevant attributes. It can be found at `https://github.com/anhaidgroup/deepmatcher/blob/master/Datasets.md`.

C REDUCTION RATIO AND RECALL VALUES

In this section, we show the Recall, RR and H(Recall, RR) values for all models across all datasets. The results are sorted in descending order based on the H(Recall, RR) value.

C.1 Dataset `abt_buy`

Model	Recall	RR	H(Recall, RR)
blockboost	0.896 7	0.926 5	0.911 3
ctt	0.840 1	0.986 8	0.907 6
hybrid	0.704 6	0.986 8	0.822 2
ae	0.697 8	0.986 8	0.817 5
canopy	0.719 9	0.807 2	0.761 0
tlsh	0.505 8	0.820 9	0.625 9
aghasher	0.477 0	0.532 5	0.503 2
klsh	0.228 3	0.917 5	0.365 6
spectral	0.155 8	0.843 4	0.263 1

C.2 Dataset `amz_gg`

Model	Recall	RR	H(Recall, RR)
blockboost	0.831 2	0.929 3	0.877 5
hybrid	0.751 5	0.977 9	0.849 9
ae	0.751 5	0.977 9	0.849 9
ctt	0.685 6	0.991 2	0.810 6
canopy	0.628 4	0.583 6	0.605 2
aghasher	0.512 0	0.570 6	0.539 7
spectral	0.516 5	0.519 9	0.518 2
klsh	0.393 3	0.746 5	0.515 2
tlsh	0.166 9	0.894 0	0.281 3

C.3 Dataset `dblp_acm`

Model	Recall	RR	H(Recall, RR)
hybrid	0.999 4	0.996 9	0.998 1
ae	0.996 8	0.996 9	0.996 8
ctt	0.991 0	0.996 9	0.993 9
blockboost	0.988 3	0.996 9	0.992 6
klsh	0.816 0	0.992 3	0.895 6
tlsh	0.769 1	0.978 4	0.861 2
canopy	1.000 0	0.739 7	0.850 4
aghasher	0.723 0	0.672 3	0.696 7
spectral	0.770 6	0.581 3	0.662 7

C.4 Dataset `dblp_sch`

Model	Recall	RR	H(Recall, RR)
ctt	0.983 6	0.998 9	0.991 2
blockboost	0.984 0	0.994 4	0.989 1
hybrid	0.967 3	0.999 9	0.983 3
ae	0.962 3	0.999 9	0.980 7
canopy	0.867 0	0.917 9	0.891 7
klsh	0.529 2	0.998 9	0.691 9
aghasher	0.703 1	0.640 4	0.670 3
spectral	0.474 2	0.826 4	0.602 6
tlsh	0.373 0	0.999 0	0.543 1

C.5 Dataset `musicbrainz`

Model	Recall	RR	H(Recall, RR)
ctt	0.990 8	0.997 9	0.994 3
hybrid	0.986 9	0.997 9	0.992 4
ae	0.986 5	0.997 9	0.992 2
blockboost	0.986 6	0.995 2	0.990 9
tlsh	0.905 3	1.000 0	0.950 3
klsh	0.894 6	1.000 0	0.944 4
aghasher	0.728 0	0.747 7	0.737 7
spectral	0.616 2	0.716 1	0.662 4
canopy	0.053 2	1.000 0	0.101 0

C.6 Dataset `restaurant`

Model	Recall	RR	H(Recall, RR)
ctt	1.000 0	0.995 2	0.997 6
hybrid	1.000 0	0.994 4	0.997 2
ae	1.000 0	0.994 4	0.997 2
blockboost	1.000 0	0.977 8	0.988 8
klsh	0.903 8	0.974 4	0.937 8
tlsh	0.730 8	0.982 2	0.838 0
canopy	0.766 0	0.805 7	0.785 4
aghasher	0.641 0	0.842 8	0.728 2
spectral	0.359 0	0.941 6	0.519 8

C.7 Dataset `rldata500`

Model	Recall	RR	H(Recall, RR)
blockboost	0.985 7	0.999 3	0.992 5
tlsh	0.985 7	0.979 0	0.982 3
klsh	0.971 4	0.968 1	0.969 7
ctt	0.942 9	0.992 1	0.966 9
ae	0.942 9	0.990 6	0.966 2
hybrid	0.942 9	0.990 6	0.966 1
canopy	0.842 9	0.816 8	0.829 6
aghasher	0.657 1	0.826 7	0.732 2
spectral	0.657 1	0.729 8	0.691 6

C.8 Dataset `rldata10000`

Model	Recall	RR	H(Recall, RR)
blockboost	0.998 2	0.999 5	0.998 8
tlsh	0.981 1	0.994 0	0.987 5
ctt	0.921 4	0.995 6	0.957 1
canopy	0.896 4	0.964 2	0.929 1
ae	0.870 0	0.996 2	0.928 8
klsh	0.876 8	0.982 0	0.926 4
hybrid	0.864 3	0.998 1	0.926 4
aghasher	0.748 6	0.860 1	0.800 5
spectral	0.682 9	0.846 2	0.755 8

C.9 Dataset `wm_amz`

Model	Recall	RR	H(Recall, RR)
ctt	0.901 2	0.990 3	0.943 6
blockboost	0.930 1	0.957 1	0.943 4
hybrid	0.899 5	0.990 3	0.942 7
ae	0.849 2	0.996 8	0.917 1
spectral	0.499 2	0.684 5	0.577 3
aghasher	0.512 6	0.613 8	0.558 6
klsh	0.347 6	0.863 4	0.495 6
canopy	0.008 8	0.990 3	0.017 4
tlsh	0.002 8	0.998 7	0.005 6

D BLOCKING AT SCALE

In this section, we explore how BlockBoost performs for very large datasets. In this case, several of the benchmarking blocking algorithms either become out of time (over 11 hours) or out of memory (over 32GB). For this reason, these experiments were not included in the main paper, but we include them in the Supplementary Material as they give further evidence of BlockBoost's scalability.

In small datasets, increasing the size of the training set by choosing random pairs as non-matches is a good unsupervised strategy to improve the predictive performance, since the chance of picking matches is very low. However, using a training dataset larger than a couple of million entries yields diminishing returns and can hinder scalability. Moreover, very large sets of candidate sets might not be desirable in some practical applications. To represent the two aforementioned scenarios, i.e. small and large datasets respectively, two experiments were included:

- **BlockBoost**: For each matching pair in the training dataset, select 16 non-matches. The best maximum hamming distance is selected using a validation dataset.
- **BlockBoost-1bi**: For each matching pair in the training dataset, select 1 non-match. The maximum hamming distance is also selected using a validation dataset, but the possibilities are restricted to values that produce a set of candidate pairs with less than 1 billion entries.

As shown in Figure 1, the price in recall of limiting the size of the set of candidate pairs is low in the `musicbrainz_2m` dataset, and this restriction prevents IO bottlenecks. All of the benchmarks ran on an Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz, with 32 GB of DDR4 – 2666 MT/s.

D.1 Predictive Performance – `musicbrainz_2m`

In this subsection, we analyze the predictive performance of blocking benchmarks, as well as BlockBoost, over the data set `musicbrainz_2m`, with 2 million entries, which can be downloaded in `https://dbs.uni-leipzig.de/research/projects/object_matching/benchmark_datasets_for_entity_resolution`. As claimed in the

main text, we find that BlockBoost can scale to this size and still maintain a competitive performance in terms of recall and reduction ratio.

D.1.1 Recall and Reduction Ratio

Model	Recall	RR	H(Recall, RR)
ctt	OOM	OOM	OOM
hybrid	OOM	OOM	OOM
ae	OOM	OOM	OOM
blockboost	0.9848	0.9941	0.9895
blockboost-1bi	0.9791	0.9984	0.9887
tlsh	OOT	OOT	OOT
klsh	OOT	OOT	OOT
canopy	OOM	OOM	OOM

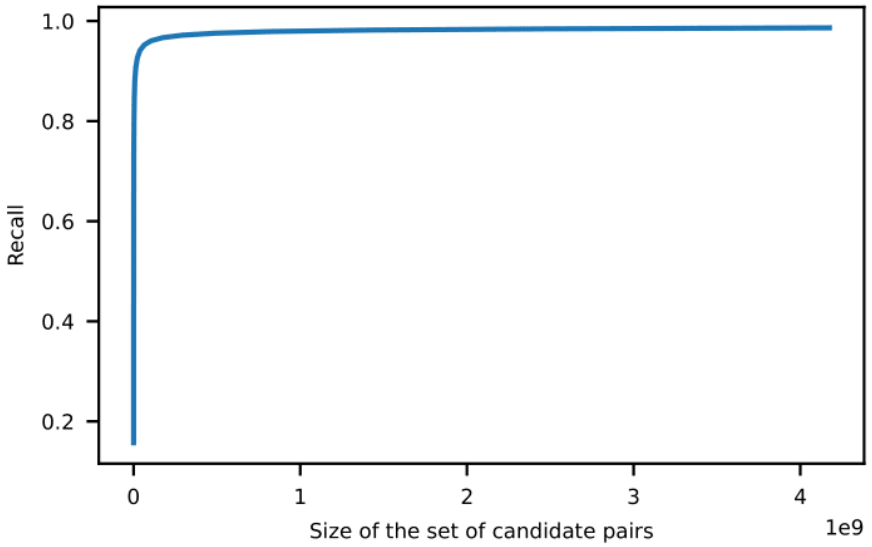


Figure 1: Recall as a function of the size of the set of candidate pairs in the `musicbrainz_2m` dataset, with proportion of non-matches of 1. Note the the maximum value of x is 1/100th of the total number of possible pairs. Computing the reduction ratio and recall for all of the possible maximum hamming distances takes 113.3 seconds on this dataset.

D.1.2 Bit Compression

Here we show the compression (in bits) achieved by BlockBoost's hashes over the original set of features.

- **BlockBoost**: 11x
- **BlockBoost-1bi**: 12x

D.2 Time

In this section, we compare BlockBoost against other benchmark models on the `musicbrainz_20` dataset with 20,000 entries, `musicbrainz_200` dataset with 200,000 entries, and `musicbrainz_2m` dataset with 2 million entries, showcasing the execution time in each case. Each of these datasets can be downloaded from `https://dbs.uni-leipzig.de/research/projects/object_matching/benchmark_datasets_for_entity_resolution`.

- `musicbrainz_20k`

klsh: 14 min 41 sec

tlsh: 2 min 18 sec

canopy: 14 min 10 sec

DeepBlocker: 12 min 35 sec

blockboost: 4.3 sec (size of train = 112234)

blockboost-1bi: .55 sec (size of train = 13204)

- `musicbrainz_200k`

  klsh: OOT

  tlsh: 52 min 23 sec

  canopy: OOM

  DeepBlocker: 2 hrs 17 min 57 sec

  blockboost: 45.4 sec (size of train = 1112820)

  blockboost-1bi: 5.61 sec (size of train = 130920)

- `musicbrainz_2m`

  klsh: OOT

  tlsh: OOT

  canopy: OOM

  DeepBlocker: OOM

  blockboost: 14 min 76 sec (size of train = 11032354)

  blockboost-1bi: 1 min 55 sec (size of train = 1297924)

E BLOCKING VS LEARNING-TO-HASH

In this section, we discuss important differences between the fields of blocking and learning-to-hash (as well as some similarities).

Blocking involves grouping together items that are considered similar based on a specific metric so that it is possible to forgo a quadratic number of comparisons and only focus on comparing entries within the same block,as we detail in Sections 1 and 2 in the main paper. For instance, if the goals it to match customer purchase records to customer accounts, blocks can be created using attributes such as last name or ZIP code of the billing address, or a combination of these. Well-designed blocks can greatly enhance the speed and efficiency of the matching process. One way to create such blocks is via hashing (although there are important alternatives). An effective hash code ensures that similar items are grouped together by mapping them to the same hash code, while dissimilar items are assigned different hash codes.

While the field of learning-to-hash also deals with developing effective hash functions, it is typically concerned with the nearest neighbors problem, rather than blocking. The nearest neighbor problem focuses on finding the most similar data points to a given query point within a dataset regardless of their specific identities or relationships. In contrast, blocking is a technique used to group together similar items based on shared attributes or criteria.

This difference gives more structure to blocking problems, which are typically exploited by benchmark algorithms. For instance, in entity matching, which is a typical application for blocking, there exists a specific notion of similarity, i.e., duplicated records that represent the same entity. In this context, the occurrence of duplicate entries often follows specific patterns, such as typos or minor textual variations. Understanding these patterns is crucial as it allows us to develop tailored techniques that improve the blocking process. Examples of such techniques include the use of shingling and minhash vectorization [5, 4], as well as the construction of artificial training sets, which is the case of DeepBlocker [6]. These approaches depend significantly on these specific textual heuristics to achieve successful outcomes, and it is not immediately obvious how to apply them to other learning-to-hash problems.

It is also important to highlight that the research tools employed in the literature vary between blocking and learning-to-hash. These two approaches have different focuses and evaluation criteria, leading to the use of distinct benchmark models and databases for assessing their performance. For instance, in blocking, in addition to recall, we also consider the reduction ratio metric. If we apply this measure to a widely used datasets in learning-to-hash literature like MNIST or CIFAR-10, we observe that since there are only 10 classes of objects, the best possible reduction ratio for each class would be 90%. This would represent a perfect blocking scenario, but even then, we would still need to perform numerous comparisons within each block (e.g. $(10\% \times 60,000)^2 = 36,000,000$ in CIFAR-10). This is not the case for most blocking problems, where we usually have just a few duplicated entries, easily leading to reduction ratio metrics close to 99.9% (as is the case for BlockBoost in some of the datasets in the paper).

In spite of these differences, one can use still apply learning-to-hash for blocking. As argued above, we expect that they result in suboptimal performance; that is indeed what happens in the paper with two traditional learning-to-hash algorithms (Aghasher and Spectral Hashing). In all of the datasets considered, BlockBoost enjoyed better performance and speed. Conversely, as alluded to in the paper's conclusion, one could adapt BlockBoost to solve typical learning-to-hash problems; that is an avenue for future work.

F BLOCKBOOST WITH LSH-INSPIRED BLOCKING

In this section, we present an alternative to the weighted hamming distance hashing step in the original paper based on the theory of Locality-sensitive hashing (LSH). We believe that this alternative solution can be used to solve classic problems in the learning-to-hash field.

The algorithm: The algorithm produces k-bit hash functions $g_1, \dots, g_L$ such that

$$g_{1,j} = k_j^* \text{ with probability } \alpha_j^*. \tag{1}$$

Items (A, B) will be part of the same block if there exists at least one hash function g_i with $g_i(A) = g_i(B)$.

Algorithm 1 LSH inspired hashing

Input: $k, L \in \mathbb{N}$, convex weights $(\alpha_t^*)_{t=1}^T$, classifiers $(k_t^*)_{t=1}^T$
for $i \leftarrow 1$ to L **do**
  for $j \leftarrow 1$ to k **do**
    $g_{i,j} \leftarrow k_t^*$ with probability α_t^*
  end for
  $g_i \leftarrow (g_{i,1}, \dots, g_{i,k})$
end for
Output: $(g_1, \dots, g_L)$

Why it works: To understand why this hashing technique works, note that, given f^* as in (2), for any $(A, B) \in \mathcal{A} \times \mathcal{B}$, and any function $g_{i,j}$ constructed as before, since $g_{i,j}$ is $\{-1, +1\}$-valued and $g_{i,j} = k_t^*$ with probability α_t for each $t \in [T]$, it is easy to show that

$$\mathbb{P}_{g_{i,j}}\left[g_{i,j}(A) = g_{i,j}(B)\right] = \mathbb{E}_{g_{i,j}}\left[\frac{1 + g_{i,j}(A)g_{i,j}(B)}{2}\right] = \frac{1}{2} + \frac{f^*(A,B)}{2}.$$

This implies that, if $f^*(A, B)$ properly approximates the similarity notion $A \sim_R B$, it is expected that $1 \approx \mathbb{P}_{g_{i,j}}\left[g_{i,j}(A) = g_{i,j}(B)\right] \geq p_1 > 1/2$ for most similar pairs, whereas $0 \approx \mathbb{P}\left[g_{i,j}(A) = g_{i,j}(B)\right] \leq p_2 < 1/2$ for most dissimilar pairs. This intuition can be made precise (see Supplementary Material, where values p_1 and p_2 will be derived from a margin property of the function f^*). Here, k and L are hyperparameters used to amplify the gap between the values p_1 and p_2.

Lemma F.1. *Let f^* be as in (2). Then for any $(A, B) \in \mathcal{A} \times \mathcal{B}$, and any function $g_{i,j}$ as in Algorithm 1,*

$$\mathbb{P}_{g_{i,j}}\left[g_{i,j}(A) = g_{i,j}(B)\right] = \frac{1 + f^*(A,B)}{2},$$

where the probability is over the choice of $g_{i,j}$.

Proof. Since $g_{i,j}$ is $\{-1,+1\}$-valued,

$$\mathbb{P}_{g_{i,j}}\left[g_{i,j}(A)=g_{i,j}(B)\right]=\mathbb{E}_{g_{i,j}}\left[\frac{1+g_{i,j}(A)g_{i,j}(B)}{2}\right].$$

Now recall that $g_{i,j}=k_t^*$ with probability α_t for each $t\in[T]$. □

Theorem F.2 (Performance of the LSH inspired hashing)**.** *Consider databases $\mathcal{A}$ and $\mathcal{B}$ such that $|\mathcal{A}|=N_{\mathcal{A}}$ and $|\mathcal{B}|=N_{\mathcal{B}}$ $\mathcal{M}$ be the set of matching pairs. For given $\theta>0$ and $\gamma\in(0,1)$, if the output f^* of Algorithm 1 satisfies the θ-margin condition and we set:*

$$\begin{aligned}\rho &:= \frac{\log\left(\frac{2}{1+\theta}\right)}{\log\left(\frac{2}{1-\theta}\right)}\in[0,1),\\ k &:= \lceil\log_{\frac{2}{1+\theta}} N_{\mathcal{A}}\cdot N_{\mathcal{B}}\rceil,\\ L &:= \left\lceil\frac{2(N_{\mathcal{A}}\cdot N_{\mathcal{B}})^{\rho}\,\log(1/\gamma)}{1+\theta}\right\rceil,\end{aligned}$$

then the LSH inspired hashing method achieves

$$\begin{aligned}\mathbb{E}\left[\text{Recall}\right] &\geq (1-\gamma)(1-\eta)\\ \mathbb{E}\left[\text{RR}\right] &\geq \left(1-\frac{|\mathcal{M}|+L}{N_{\mathcal{A}}\cdot N_{\mathcal{B}}}\right)(1-\eta),\end{aligned}$$

where expectations are with respect to the randomness in the hash code, and Recall and RR are defined in (5) and (6).

Proof. This proof is an adaptation of [2]. Since f^* satisfies Condition 4.1 for $\theta>0$, we know by Lemma F.1 that for all A,B in a set $\mathcal{E}$ of P-measure $\geq 1-\eta$

$$\begin{aligned}&\text{if } A\sim_R B, \text{then } \mathbb{P}_{g_{i,j}}\left[g_{i,j}(A)=g_{i,j}(B)\right]\geq\frac{1+\theta}{2}=p_1\\ &\text{if } A\not\sim_R B, \text{then } \mathbb{P}_{g_{i,j}}\left[g_{i,j}(A)=g_{i,j}(B)\right]\leq\frac{1-\theta}{2}=p_2,\end{aligned}$$

where P is as described in Section 4. For our next calculations assume we are conditioned to this event. Fix $k=\lceil\log_{1/p_2} N_{\mathcal{A}}\cdot N_{\mathcal{B}}\rceil$ and let $\mathcal{M}$ be the set of matching pairs as defined in (7). We split the proof in the following steps:

- **Probability of finding correct matches.** Suppose that $A\sim_R B$ and $(A,B)\in\mathcal{E}$. By independence, for $i\in[L]$

$$\begin{aligned}\mathbb{P}_{g_i}\left[g_i(A)=g_i(B)\right] &= \mathbb{P}_{g_{i,j}}\left[g_{i,j}(A)=g_{i,j}(B)\right]^k\\ &\geq p_1^k\\ &\geq p_1^{\log_{1/p_2}(N_{\mathcal{A}}\cdot N_{\mathcal{B}})+1}\\ &= p_1 p_1^{\log_{1/p_2}(N_{\mathcal{A}}\cdot N_{\mathcal{B}})}\\ &= p_1(N_{\mathcal{A}}\cdot N_{\mathcal{B}})^{-\rho},\end{aligned}$$

  where in the last equality we used a simple logarithm change of basis. That is,

$$\mathbb{P}_{g_i}\left[g_i(A)\neq g_i(B)\right]\leq 1-p_1(N_{\mathcal{A}}\cdot N_{\mathcal{B}})^{-\rho},$$

  Thus, the probability of finding the correct match is

$$\begin{aligned}\mathbb{P}\left[\exists i\in\{1,\dots,L\},\ g_i(A)=g_i(B)\right] &= 1-\mathbb{P}\left[\forall i\in\{1,\dots,L\},\ g_i(A)\neq g_i(B)\right]\\ &= 1-\mathbb{P}_{g_i}\left[g_i(A)\neq g_i(B)\right]^L\\ &\geq 1-\left(1-p_1(N_{\mathcal{A}}\cdot N_{\mathcal{B}})^{-\rho}\right)^L\end{aligned}$$

hence, by setting $L = \frac{\log(1/\gamma)(N_{\mathcal{A}} \cdot N_{\mathcal{B}})^{\rho}}{p_1}$ for $\gamma \in (0,1)$, we have that

$$\begin{aligned}\mathbb{P}\left[\exists i \in \{1,\dots,L\},\ g_i(A) = g_i(B)\right] &\geq 1 - \left(1 - p_1(N_{\mathcal{A}} \cdot N_{\mathcal{B}})^{-\rho}\right)^L \\ &\geq 1 - e^{-\log(1/\gamma)} \\ &= 1 - \gamma.\end{aligned}$$

- **Expected Recall.** By the previous item, we have

$$\begin{aligned}\mathbb{E}\left[\text{Recall}\right] &\geq \frac{1}{|\mathcal{M}|} \sum_{(\ell,r)\in\mathcal{M}} \mathbb{P}\left[\exists i \in \{1,\dots,L\}, g_i(A_\ell) = g_i(B_r) | (A,B) \in \mathcal{E}\right] P[\mathcal{E}] \\ &\geq (1-\gamma)(1-\eta).\end{aligned}$$

- **Probability of finding wrong matches.** Suppose that $A \not\sim_R B$ and $(A,B) \in \mathcal{E}$. Then, for $i \in [L]$:

$$\begin{aligned}\mathbb{P}_{g_i}\left[g_i(A) = g_i(B)\right] &= \mathbb{P}_{g_{i,j}}\left[g_{i,j}(A) = g_{i,j}(B)\right]^k \\ &\leq p_2^k \\ &\leq \frac{1}{N_{\mathcal{A}} \cdot N_{\mathcal{B}}},\end{aligned}$$

by our choice of k.

- **Expected number of wrong matches.** By the previous item, conditioned to $(A,B) \in \mathcal{E}$, the random variable that counts the number wrong matches found by g_i

$$C(g_i) = \sum_{(\ell,r)\notin\mathcal{M}} \mathbf{1}_{[g_i(A_\ell)=g_i(B_r)]}$$

follows a binomial distribution with parameter $\left(N_{\mathcal{A}} \cdot N_{\mathcal{B}} - |\mathcal{M}|, \frac{1}{N_{\mathcal{A}} \cdot N_{\mathcal{B}}}\right)$, hence

$$\mathbb{E}_{g_i}\left[C(g_i)\right] \leq 1,$$

therefore the number of total wrong collisions for g_i is at most 1 and the number of total wrong collisions for all g_i for $i \in \{1,\dots,L\}$ is at most L.

- **Expected RR.** By the previous item and the fact that Condition 4.1 holds with probability $\geq 1-\eta$, the expected number of comparisons is

$$\begin{aligned}\mathbb{E}\left[\#\text{ comparisons}\right] \leq &\sum_{(\ell,r)\in[N_{\mathcal{A}}]\times[N_{\mathcal{B}}]} \mathbb{P}\left[\exists i \in \{1,\dots,L\}, g_i(A_\ell) = g_i(B_r) | (A,B) \in \mathcal{E}\right] P[(A,B) \in \mathcal{E}] \\ &+ P[(A,B) \notin \mathcal{E}] \\ &\leq \sum_{(\ell,r)\in\mathcal{M}} \mathbb{P}\left[\exists i \in \{1,\dots,L\}, g_i(A_\ell) = g_i(B_r) | (A,B) \in \mathcal{E}\right](1-\eta) \\ &+ \sum_{(\ell,r)\notin\mathcal{M}} \mathbb{P}\left[\exists i \in \{1,\dots,L\}, g_i(A_\ell) = g_i(B_r) | (A,B) \in \mathcal{E}\right](1-\eta) + \eta \\ &\leq (|\mathcal{M}| + L)(1-\eta) + \eta.\end{aligned}$$

Therefore, the expected RR satisfies

$$\mathbb{E}\left[\text{RR}\right] \geq 1 - \eta - \left(\frac{|\mathcal{M}| + L}{N_{\mathcal{A}} \cdot N_{\mathcal{B}}}\right)(1-\eta).$$

The number of number of operations and the size of our data structure can be easily estimated using our previous calculations, but can also be found in [2], Theorem 3.4.

□

Algorithmic complexity. The number of operations can be derived from the proof of Theorem F.2. It is possible to show that if $\xi := (N_{\mathcal{A}} \cdot N_{\mathcal{B}})^{\rho} / \log(2/(1+\theta))$, then the algorithm requires at most $\mathcal{O}(\xi)$ distance computations/evaluations of hash functions and the data structure uses at most $\mathcal{O}(\xi)$ words of space, in addition to the space needed to store the dataset.