Proceedings of Machine Learning Research 129:689–704, 2020 ACML 2020

Randomness Efficient Feature Hashing for Sparse Binary Data

Rameshwar Pratap RAMESHWAR.PRATAP@GMAIL.COM
IIT-Mandi, H.P., India

Karthik Revanuru KARTHIK.REVANURU@CRED.CLUB
CRED, Bengaluru, India

Anirudh Ravi RANIRUDH1995@GMAIL.COM
University of Southern California, USA

Raghav Kulkarni KULRAGHAV@GMAIL.COM
Chennai Mathematical Institute (CMI), Chennai, India

Editors: Sinno Jialin Pan and Masashi Sugiyama.

Abstract

We present sketching algorithms for sparse binary datasets, which maintain binary version of the dataset after sketching, while simultaneously preserving multiple similarity measures such as Jaccard Similarity, Cosine Similarity, Inner Product, and Hamming Distance, on the same sketch. A major advantage of our algorithms is that they are randomness efficient, and require significantly less number of random bits for sketching – logarithmic in dimension, while other competitive algorithms require linear in dimension. Our proposed algorithms are efficient, offer a compact sketch of the dataset, and can be efficiently deployed in a distributive setting. We present a theoretical analysis of our approach and complement them with extensive experimentations on public datasets. For analysis purposes, our algorithms require a natural assumption on the dataset. We empirically verify the assumption and notice that it holds on several real-world datasets.
Keywords: Feature Hashing, Binary Data, Cosine Similarity, Jaccard Similarity.

1. Introduction

Due to the recent technological advancements in the areas of the WWW, IOT, cloud, etc., such applications generate a large volume of high dimensional data. For example, in many web applications, the dimension of the datasets are of the order of billions (Agarwal et al., 2014). Further, most of such high dimensional datasets are sparse and binary, owing to a wide adaption of "Bag-of-words" (BoW) representations. For example: in the case of document representation, word frequency within a document follows power law – most of the words occur rarely in a document, and higher order shingles occur only once. Sparse binary representation of datasets in large scale systems is also quite common in several industrial applications (Chandra et al.).

Computing the similarity score of data points under various similarity measures is a fundamental subroutine in several applications such as clustering, classification, identifying nearest neighbors, ranking. However, due to the "curse of dimensionality" a brute-force way of computing the similarity scores in the high dimensional dataset is infeasible, and at times impossible. In this work, we focus on deriving an efficient feature-hashing algorithm for

sparse binary datasets that can generate a succinct sketch of the dataset while preserving the similarity between data objects.

Important observation pertaining to feature hashing for sparse binary data. Given a pair of ψ-sparse (number of 1's in each vector is at most ψ) binary vectors, there are at most 2ψ *interesting* (indices having non-zero values) bit positions. If a hashing algorithm can isolate those indices into different buckets, then the similarities between original vectors are also preserved in the compressed version. This observation was originally noted in (Pratap et al., 2018a,b), that suggests a randomized bucketing algorithm, where each index of the input is randomly assigned to one of the $O(\psi^2)$ buckets. The sketch of an input vector is obtained by computing the parity of the bits fallen in each bucket.

Why saving on the randomness is important: We argue that randomness is a key resource needed for the computation along with running time and memory. Most of the feature hashing algorithms are randomized and require random bits for generating hash functions. Generation of random bits is a computationally expensive task. Our work focuses on saving the random bits required for the computation. We would like to emphasize that most state-of-the-art feature hashing algorithms require a large number of random bits, which is linear in the dimension of the dataset, making them impractical in the case of high dimensional datasets. The number of random bits required by our algorithms grows logarithmically in d, and offers a huge saving.

1.1. Our Contribution

We propose a randomness efficient feature hashing algorithm – PivotHash– for sparse binary datasets. We start by stating a couple of the following definitions.

Definition 1. *Consider a d-dimensional binary vector $\mathbf{u} \in \{0,1\}^d$. We call an index i of $\mathbf{u}$ a non-zero index if $\mathbf{u}[i] = 1$. We call a binary vector $\mathbf{u}$, a ψ-sparse vector, if the number of non-zero indices in $\mathbf{u}$ is at most ψ.*

Definition 2. *Consider a ψ-sparse, d-dimensional binary vector $\mathbf{u} \in \{0,1\}^d$. We call a pair of indices $\{i, j\}$ of $\mathbf{u}$ "well-spread" if $|i - j| = \Omega(d/\psi)$. We call a vector $\mathbf{u}$ well-spread if every pair of its non-zero indices is well-spread. Similarly, we call a binary dataset well-spread if each of its vectors is well-spread.*

Please note that for a ψ-sparse dataset, the number of candidate pairs (of non-zero indices) for well-spread is $(\psi - 1)$. We argue that the notion of well-spreadness is a natural assumption for most of the sparse, binary, high-dimensional, and real-world datasets. For example, in the case of text dataset while creating a BoW model, by choosing a hash function – which maps words to the indices from 1 to d uniformly at random – the notion of well-spreadness can be satisfied, with high probability. We statistically verify the assumption on several real-world datasets and observed that almost all pair of non-zero indices satisfies the assumption. We discuss this detail in Section 4. For a pair of sparse and well spread binary vectors, our proposed algorithm PivotHash (see Subsection 3.1 for the Algorithm) offers the following guarantee.

Theorem 3. *Let $\mathbf{u}, \mathbf{v} \in \{0,1\}^d$ be a pair of ψ-sparse and well-spread vectors. If we set the number of pivots $k = O(\psi \log \psi)$, and compress them into binary vectors $\mathbf{u}', \mathbf{v}' \in \{0,1\}^{\mathrm{N}}$ via*

Table 1: A comparison among the candidate algorithms, on the number of random bits and the compression time, to get a sketch of length N of a single data object. Compression time includes both (i) time required to generate hash function, which is of order the number of random bits, (ii) time required to generate the sketch using the hash functions. ψ is the sparsity of the dataset.

Algorithm	No of random bits	Compression time
This work	$O(\psi \log \psi \log d)$	$O(\psi \log \psi \log d + \psi)$
BCS (Pratap et al., 2018a,b)	$O(d \log \mathrm{N})$	$O(d \log \mathrm{N} + \psi)$
DOPH (Shrivastava, 2017)	$O(d \log d)$	$O(d \log d + \psi + \mathrm{N})$
CBE (Yu et al., 2014)	$O(d)$	$O(d \log d)$
SimHash (Charikar, 2002)	$O(d\mathrm{N})$	$O((d + \psi)\mathrm{N})$
MinHash (Broder et al., 1998)	$O((d \log d)\mathrm{N})$	$O((d \log d + \psi)\mathrm{N})$

PivotHash*, where* $\mathrm{N} \leq 2k$*, then with a constant probability the values of Jaccard Similarity, Cosine Similarity, Inner Product, and Hamming Distance between* $\mathbf{u}', \mathbf{v}'$ *are same as of their corresponding values between* $\mathbf{u}, \mathbf{v}$.

The following theorem comments on the number of random bits required by PivotHash, and the subsequent remark quantifies this with respect to the other state-of-the-art algorithms, which provide a similar guarantee as of Theorem 3.

Theorem 4. *The number of random bits required by* PivotHash *in order to satisfy the guarantee offered by Theorem 3 is* $O(\psi \log \psi \log d)$.

For Jaccard Similarity, we compare the performance of our algorithms with MinHash (Broder et al., 1998), DOPH (Shrivastava, 2017), and BCS (Pratap et al., 2018b), and for Cosine Similarity, we compare it with SimHash (Charikar, 2002), CBE (Yu et al., 2014), and MinHash (Shrivastava and Li, 2015). In both the scenarios, for sparse binary datasets, our proposed algorithm PivotHash suggests a faster algorithm, requires significantly less number of random bits, while simultaneously offering almost a similar performance as compared to the baselines. We summarise an asymptotic comparison among the baseline algorithms on the number of random bits needed, and the compression time in Table 1. Further, an added advantage with PivotHash is that it can be efficiently distributed (see Subsection 3.1).

1.2. Applicability of our results

In the case of high dimensional sparse binary datasets, our algorithms can be used to improve the performance of numerous applications, which require Jaccard/Cosine/Hamming Distance/Inner Product similarity preserving sketch. That is, in applications, where their respective state-of-the-art sketching algorithms MinHash, SimHash, BCS (Pratap et al., 2018b,a) Asymmetric MinHash (Shrivastava and Li, 2015) are in use.

Scalable Ranking and deduplication of documents. Given a corpus of documents and a set of query documents, the task is to rank all documents in the corpus that are similar (under the given similarity measures like Jaccard, Cosine, Inner Product) to the

query documents. This problem is a fundamental sub-routine in many applications like near-duplicate data detection (Sood and Loguinov, 2011; Henzinger, 2006; Broder, 2000), efficient document similarity search (Jiang and Sun, 2011; Shrivastava and Li, 2015) plagiarism detection (Buyrukbilen and Bakiras, 2013; Broder, 2000). In Subsection 4.2, we provide empirical validations through a set of experiments that our proposed algorithms are efficient both in terms of space and time while maintaining almost a similar accuracy.

Scalable Clustering of documents. For high-dimensional and sparse binary datasets, our proposed algorithms can be used in scaling up the performance of several clustering algorithms by offering a succinct and accurate sketch of the original dataset. For example: in the case of Spherical k-means (Dhillon and Modha, 2001), which is a popular choice of clustering text documents using Cosine Similarity; and k-mode (Huang, 1998) clustering algorithm which is used for clustering binary dataset using Hamming Distance.

Other Applications. Apart from the above-mentioned applications, sketching techniques have been widely used in applications like compressing social networks (Chierichetti et al., 2009), all pair similarity (Bayardo et al., 2007). Our proposed methods can be used to scale up the performance of the respective algorithms by offering a randomness efficient, succinct, and accurate sketch.

2. Background

Notations	
N	dimension of the compressed data.
ψ	upper bound on the number of 1's in binary vectors.
k	number of sampled pivots for PivotHash.
$\mathbf{u}[i]$	i-th bit position of binary vector $\mathbf{u}$.
$\lvert\mathbf{u}\rvert$	number of 1's in the binary vector $\mathbf{u}$.
$\cos(\mathbf{u},\mathbf{v})$	Cosine Similarity between $\mathbf{u}$ and $\mathbf{v}$.
$\mathrm{JS}(\mathbf{u},\mathbf{v})$	Jaccard Similarity between $\mathbf{u}$ and $\mathbf{v}$.
$\mathrm{d_H}(\mathbf{u},\mathbf{v})$	Hamming Distance between $\mathbf{u}$ and $\mathbf{v}$.
$\langle\mathbf{u},\mathbf{v}\rangle$	Inner Product between $\mathbf{u}$ and $\mathbf{v}$.

SimHash – a sketching algorithm for Cosine Similarity (Charikar, 2002; Goemans and Williamson, 1995). The Cosine Similarity between a pair of vectors $\mathbf{u},\mathbf{v} \in \mathbb{R}^d$ is defined as $\langle\mathbf{u},\mathbf{v}\rangle/||\mathbf{u}||_2||\mathbf{v}||_2$. To compute a sketch of the vector $\mathbf{u}$, SimHash (Charikar, 2002) generates a random vector $\mathbf{r} \in \{-1,+1\}^d$, with each component generated from $\{-1,+1\}$ with probability 1/2, and only stores the sign of the projected data. That is,

$$\text{SimHash}^{(r)}(\mathbf{u}) = \begin{cases} 1, & \text{if } \langle\mathbf{u},\mathbf{r}\rangle \geq 0. \\ 0, & \text{otherwise.} \end{cases}$$

It was shown in (Goemans and Williamson, 1995) that SimHash offers the following guarantee

$$\Pr[\text{SimHash}^{(r)}(\mathbf{u}) = \text{SimHash}^{(r)}(\mathbf{v})] = 1 - \frac{\theta}{\pi}, \text{ where } \theta = \cos^{-1}\left(\langle\mathbf{u},\mathbf{v}\rangle/||\mathbf{u}||_2||\mathbf{v}||_2\right).$$

MinHash – a sketching algorithm for Jaccard Similarity (Broder et al., 1998). The Jaccard Similarity between a pair of set $\mathbf{u}, \mathbf{v} \subseteq \{1, 2, \dots d\}$ is defined as $\text{JS}(\mathbf{u}, \mathbf{v}) = \frac{|\mathbf{u} \cap \mathbf{v}|}{|\mathbf{u} \cup \mathbf{v}|}$. (Broder et al., 1998) suggested an algorithm – MinHash – to compress a collection of sets while preserving the Jaccard Similarity between any pair of sets.

Definition 5 (Minhash (Broder et al., 1998)). *Let π be a random permutation over $\{1, \dots, d\}$, then for a set $\mathbf{u} \subseteq \{1, \dots d\}$ $h_\pi(\mathbf{u}) = \arg\min_i \pi(i)$ for $i \in \mathbf{u}$. Then,* $\Pr[h_\pi(\mathbf{u}) = h_\pi(\mathbf{v})] = \frac{|\mathbf{u} \cap \mathbf{v}|}{|\mathbf{u} \cup \mathbf{v}|}$.

BCS – a sketching algorithm for sparse binary datasets (Pratap et al., 2018b,a). For a sparse binary dataset, BCS offers a sketching algorithm which simultaneously preserves Jaccard Similarity, Hamming Distance and Inner Product. We formally define BCS as follows:

Definition 6 (BCS). *Let* N *be the number of buckets, for $i = 1$ to d, we randomly assign the i-th position to a bucket number $b(i) \in \{1, \dots \text{N}\}$. Then a vector $\mathbf{u} \in \{0, 1\}^d$, compressed into a vector $\mathbf{u}' \in \{0, 1\}^{\text{N}}$ as follows:* $\mathbf{u}'[j] = \sum_{i:b(i)=j} \mathbf{u}[i] \pmod 2$.

3. Detailed Results

3.1. PivotHash Details

For a sparse and well-spread binary dataset, we present our algorithm PivotHash as follows. Given a d-dimensional binary dataset, our aim is to come up with a hash function $\mathcal{H}(.)$ that maps an index $i \in \{1 \dots d\}$ to the id (signature) of the bucket. We obtain it *via* a two-step procedure, and then use these signatures for sketching binary vectors in Step 3.

Step 1: We first sample k pivots, $p_1, \dots, p_k$, where each p_j is chosen *u.a.r.* from 1 to d, where the value of k is decided later on.

Step 2: Based on the sampled pivots $p_1, .., p_k$, we define k hash functions $h_{p_1}(.), .., h_{p_k}(.)$ as follows:

$$h_{p_j}(i) = \begin{cases} 0, & \text{if } d_{cyc}(i, p_j) < d/2. \\ 1, & \text{otherwise.} \end{cases}$$

Where,

$$d_{cyc}(a, b) = \begin{cases} a - b, & \text{if } b \leq a. \\ d + a - b - 1, & \text{otherwise.} \end{cases}$$

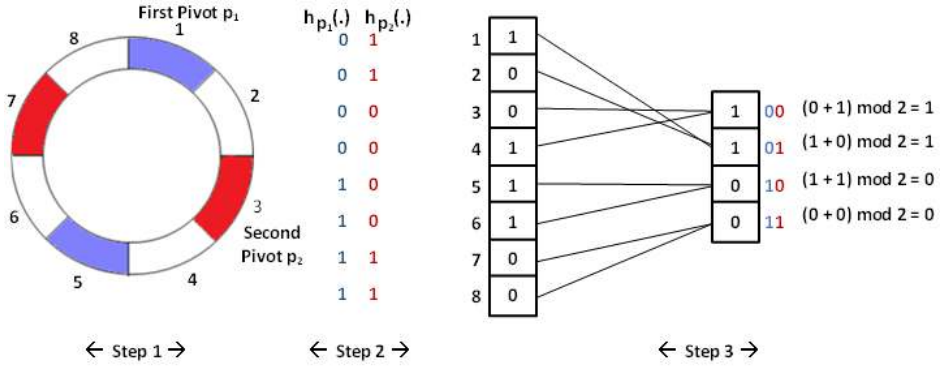
The mapping $\mathcal{H}(.)$ is obtained by concatenating $h_{p_1}(.), \dots, h_{p_k}(.)$. Further, we describe how the mapping $\mathcal{H}(.)$ is used for compressing a given binary vector $\mathbf{u} \in \{0, 1\}^d$ to a binary vector $\mathbf{u} \in \{0, 1\}^{\text{N}}$, where N is the number of buckets with distinct bucket ids.

Step 3: Let x denote an integer corresponding to the k bit binary vector obtained from $\mathcal{H}(i)$. Then

$$\mathbf{u}'[x] = \sum_{i:\mathcal{H}(i)=x} \mathbf{u}[i] \pmod 2.$$

We summarized all the steps of PivotHash in Figure 1. In PivotHash, each of the sampled pivots hash indices with respect to their given order, and due to that, it could generate at most $2k$ distinct k-bit binary strings. We summarize this in the following Observation.

Figure 1: An illustration of PivotHash on an 8-dimensional dataset, with the sampled pivots $p_1 = 1$ and $p_2 = 3$. The clockwise cyclic distance between p_1 and the indices $1, 2, 3, 4$ is less than 4, therefore they are hashed to 0, and the remaining indices are hashed to 1. Similarly, p_2 hashes the indices $3, 4, 5, 6$ to 0 and the remaining to 1. After concatenation of these two hash values, four distinct bucket ids are generated. Each index is mapped to one of these buckets. Finally, a binary vector is compressed by taking the parity of bits fallen in each bucket.



Observation 7. *Consider a d-dimensional binary dataset. If we randomly sample k pivots from indices* $\in \{1, .., d\}$ *via* PivotHash*, then the number of buckets with distinct signatures is at most* $2k$*.*

For a sparse and well-spread dataset, PivotHash offers the following guarantee.

Theorem 8. *Let* $\mathbf{u}$ *be a* ψ*-sparse and well-spread vector. If we set* $k = 4\psi \log \psi$ *in* PivotHash*, then with probability at least* $1 - 1/e^4$ *hash values of every pair of the non-zero indices of* $\mathbf{u}$ *are different.*

Proof Consider a pair of non-zero indices $\{i, j\}$ of the vector $\mathbf{u}$. The probability that the hash values of these indices under the first sampled pivot are different is equal to $(2|i - j| - 2)/d$. The probability that the hash values of these indices under the first sampled pivot are same is equal to $1 - (2|i - j| - 2)/d$. Further, as the vector $\mathbf{u}$ is well-spread, $|i - j| = \Omega(d/\psi)$, then the above probability is at most $1 - 2c/\psi + 2/d$, where c is some constant with $c > 2$. As $d \geq \psi$, which implies $2/d \leq 2/\psi$, therefore the above probability is at most $1 - 2(c - 1)/\psi$, which is at most $1 - 2/\psi$. Further, probability that hash values of $\{i, j\}$ by all of the first k sampled pivots are same is at most $(1 - 2/\psi)^k$. If we set $k = 4\psi \log \psi$ the above probability is at most $1/(\psi^2 e^4)$. A proof of the theorem holds due to the probability union bound on the ψ^2 pairs of non-zero indices.

Theorem 8 along with Observation 7 suggest the following.

Corollary 9. *Let* $\mathbf{u}, \mathbf{v} \in \{0, 1\}^d$ *be a pair of* ψ*-sparse and well-spread vectors. If we set* $k = 8\psi \log \psi$*, and compress them into binary vectors* $\mathbf{u}', \mathbf{v}' \in \{0, 1\}^{\mathrm{N}}$ *via* PivotHash*, where* $\mathrm{N} \leq 2k$*. Then each of the following holds true with a constant probability* (a) $\mathrm{d_H}(\mathbf{u}, \mathbf{v}) = \mathrm{d_H}(\mathbf{u}', \mathbf{v}')$*,* (b) $\langle \mathbf{u}, \mathbf{v} \rangle = \langle \mathbf{u}', \mathbf{v}' \rangle$*.*

For a pair of binary vectors, the following proposition relates Jaccard Similarity with Hamming Distance and Inner Product.

Proposition 10. *Let* $\mathbf{u}, \mathbf{v} \in \{0,1\}^d$. *Then, we have* $\mathrm{JS}(\mathbf{u}, \mathbf{v}) = \frac{\langle \mathbf{u}, \mathbf{v} \rangle}{(\langle \mathbf{u}, \mathbf{v} \rangle + \mathrm{d_H}(\mathbf{u}, \mathbf{v}))}$.

As a result of Corollary 9 and Proposition 10, PivotHash compression preserves Jaccard Similarity.

Corollary 11. *Let* $\mathbf{u}, \mathbf{v} \in \{0,1\}^d$ *be a pair of* ψ*-sparse and well-spread vectors. If we set* $k = 8\psi \log \psi$, *and compress them into binary vectors* $\mathbf{u}', \mathbf{v}' \in \{0,1\}^{\mathrm{N}}$ *using* PivotHash, *where* $\mathrm{N} \leq 2k$. *Then with a constant probability we have,* $\mathrm{JS}(\mathbf{u}, \mathbf{v}) = \mathrm{JS}(\mathbf{u}', \mathbf{v}')$.

As a result of Theorem 8 and Corollary 9, PivotHash compression preserves Cosine Similarity.

Corollary 12. *Let* $\mathbf{u}, \mathbf{v} \in \{0,1\}^d$ *be a pair of* ψ*-sparse and well-spread vectors. If we set* $k = 8\psi \log \psi$, *and compress them into binary vectors* $\mathbf{u}', \mathbf{v}' \in \{0,1\}^{\mathrm{N}}$ *via* PivotHash, *where* $\mathrm{N} \leq 2k$. *Then with a constant probability we have,* $\cos(\mathbf{u}, \mathbf{v}) = \cos(\mathbf{u}', \mathbf{v}')$.

Proof of Theorem 3. Observation 7 and Corollaries 9,11,12 completes a proof of Theorem 3.

The following corollary extends the above results for a set of binary vectors.

Corollary 13. *Consider a set* U *of binary vectors* $\{\mathbf{u_i}\}_{i=1}^n \subseteq \{0,1\}^d$, *where each vector* $\{\mathbf{u_i}\}_{i=1}^n$ *is* ψ*-sparse and well-spread. If we set* $k' = O(\psi \log \psi \log n)$, *and compress them into a set* U′ *of binary vectors* $\{\mathbf{u'_i}\}_{i=1}^n \subseteq \{0,1\}^{\mathrm{N}}$ *via* PivotHash, *where* $\mathrm{N} \leq 2k'$. *Then for all* $\mathbf{u_i}, \mathbf{u_j} \in \mathrm{U}$, *each of the following is true with a constant probability* (a) $\mathrm{d_H}(\mathbf{u_i}, \mathbf{u_j}) = \mathrm{d_H}(\mathbf{u_i}', \mathbf{u_j}')$; (b) $\langle \mathbf{u_i}, \mathbf{u_j} \rangle = \langle \mathbf{u_i}', \mathbf{u_j}' \rangle$; (c) $\mathrm{JS}(\mathbf{u_i}, \mathbf{u_j}) = \mathrm{JS}(\mathbf{u_i}', \mathbf{u_j}')$; (d) $\cos(\mathbf{u_i}, \mathbf{u_j}) = \cos(\mathbf{u_i}', \mathbf{u_j}')$.

PivotHash can be efficiently distributed. PivotHash can be easily deployed in a distributed environment. To distribute the mapping, it is enough to share the sampled pivots, which can be kept in the *master-node*, and it requires only $O(\psi \log \psi \log d)$ bits. All the *client-nodes* can access these pivots from the master node, and can recover the mapping at their end for compression. Therefore PivotHash suggests a *succinct* hash function which can be shared efficiently, especially in a distributed setting.

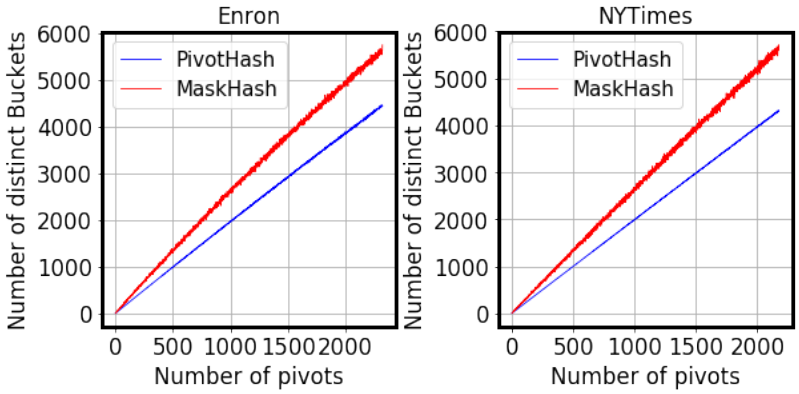
Each sampled pivot in PivotHash performs the hashing on the given fixed ordering of indices. We notice that if it is possible to generate an independent random permutation of indices *w.r.t.* each sampled pivot, then a larger number of buckets can be generated, which further helps in offering a more randomness efficient hashing. We made a step in this direction by suggesting a heuristic – MaskHash – which generates an almost random permutation of indices corresponding to each sampled pivot. We discuss it below.

3.2. Improving the performance of PivotHash.

PivotHash hashes the indices from 1 to d based on their fixed given ordering, and because of that, it could generate at most $2k$ distinct buckets (Observation 7). We argue that for a given value of k, generating a large number of buckets helps in saving on the randomness. We discuss this as follows. If it is possible to generate a random and independent ordering of indices for each pivot, then 2^k distinct buckets can be generated, and indices from 1 to d will be mapped uniformly across all 2^k buckets. Therefore, due to (Pratap et al., 2018b),(Pratap

et al., 2018a) if we set $N = O(\psi^2)$, which implies to choose $k = O(\log \psi)$, then the guarantees of similarity preserving compression similar to Corollaries 9,11,12 will hold. This would result in a significant saving in terms of randomness which is $k \log d = \log \psi \times \log d$, whereas in the case of PivotHash it is $O(\psi \log \psi \log d)$. The above-suggested approach in principle holds true. However, it is computationally expensive, as generating one independent random permutation corresponding to each pivot requires $O(d \log d)$ random bits, which is infeasible for large values of d.

Figure 2: Comparison of the number of distinct buckets *vs* the number of pivots between PivotHash and MaskHash.



MaskHash. We attempt to replace the method discussed above with an efficient proxy method – MaskHash. For each randomly sampled pivot p_j, it generates a $\log d$ bit random string – *mask*. Further, it generates an approximate random permutation of the indices, by taking bitwise XOR between every index from 1 to d with the sampled mask.

With one random mask corresponding to each pivot, we can potentially hope to generate more than $2k$ distinct buckets which requires $O(k \log d)$ random bits. We could not prove any mathematical bound on the number of distinct buckets generated *via* MaskHash. However, we experimentally validate that it indeed generates a larger number of buckets as compared to PivotHash (see Figure 2), and as a consequence, offers somewhat better performance compared to the PivotHash. We argue that MaskHash can be of independent interest because it helps in generating *proxies* of random permutations. Therefore, it can help in scaling up the algorithms/applications that require generating random permutations, *e.g.* MinHash (Broder et al., 1998).

4. Experiments

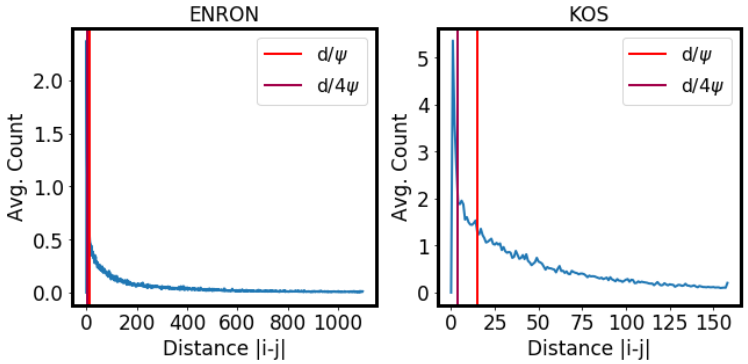
Hardware description. We performed our experiments on a machine having the following configuration: CPU: Intel(R) Core(TM) i5-3320M CPU @ 2.60GHz x 4; Memory: 7.5 GB; OS: Ubuntu 18.04; Model: Lenovo Thinkpad T430.

Datasets. The experiments were performed on publicly available datasets - namely, NYTimes news articles (number of points = 300000, dimension = 102660), Enron Emails (number of points = 39861, dimension = 28102), and KOS blog entries (number of points = 3430,

dimension = 6906) from the UCI machine learning repository (Lichman, 2013); and BBC News Datasets (number of points = 2225, dimension = 9635) (Greene and Cunningham, 2006). We considered the entire corpus of KOS and BBC News datasets, while for NYTimes, ENRON datasets we took a sample of 5000 data points.

Competing Algorithms: For our experiments, we have used two similarity measures: Jaccard Similarity and Cosine Similarity. For the Jaccard Similarity, MinHash (Broder et al., 1998), Densified One Permutation Hashing (DOPH) – a faster variant of MinHash – (Shrivastava, 2017), and BCS (Pratap et al., 2018b) were the competing algorithms. For the Cosine Similarity, SimHash (Charikar, 2002), Circulant Binary Embedding (CBE) – a faster variant of SimHash – (Yu et al., 2014), MinHash (Shrivastava and Li, 2014), and DOPH (Shrivastava, 2017) on the sketch obtained by MinHash (Shrivastava and Li, 2014), were the competing algorithms.

Figure 3: Empirical verification of well-spreadness assumption.



Empirically verifying well-spreadness assumption. In this experiment, we aim to show that a large fraction of pairs of non-zero indices satisfy our well-spreadness assumption. We do this by generating plots between distance between non-zero indices with their respective count. To this purpose, for every data point, we compute the distance between consecutive pair of non-zero indices and their respective count. This step is repeated for all the data points, and then we calculate the average. We further noted the number of pair of indices whose distance is at least $d/4\psi$ and d/ψ. We summarise our result in Figure 3. We observed that for Enron dataset 92.54% and 82.84% pairs of non-zero indices are at distance at least $d/4\psi$ and d/ψ, respectively. For KOS datasets there numbers were 88.54% and 69.44%, respectively. We obtained a similar results on the other datasets as well. This validates that a large number of pair of indices satisfies the well-spreadness assumption.

4.1. Experiment 1: Accuracy of Estimation

Evaluation Metric. In order to understand the behavior of our proposed algorithms PivotHash and MaskHash on various similarity thresholds, we need to extract samples of similar pairs (on various thresholds) from our datasets. For this purpose, we enumerated over all the pairs, and extracted those whose Jaccard/Cosine Similarities were higher than the given thresholds $\in \{0.95, 0.9, 0.85, 0.8, 0.6, 0.5, 0.2, 0.1\}$. For example: for the threshold value 0.95, we considered only those pairs whose Jaccard/Cosine Similarities are higher than 0.95. We used mean square error (MSE) as our evaluation criteria. Using PivotHash

and MaskHash, we compressed the datasets to various values of compression length N. We compressed the pruned datasets using the competitive algorithms to the same values of N. We then calculated the MSE for all the competing algorithms, for different values of N. We illustrate this as follows. For example, in order to calculate the MSE of PivotHash with respect to the ground truth result, for every pair of data points, we calculated the square of the difference between their estimated Jaccard/Cosine Similarity after the result of PivotHash, and the corresponding ground truth Jaccard/Cosine Similarity. We added these values for all such pairs and calculated its mean. The value of this quantity can be at most 1, and we computed its negative logarithm base e. A higher value $-\log(\text{MSE})$ is an indication of better performance. Similarly, we calculated $-\log(\text{MSE})$ of other algorithms *w.r.t.* their respective ground truth similarities.

Insights. We summarize our results in Figure 5. For Jaccard Similarity, on higher and intermediate thresholds, our algorithms tend to be performing almost similar *w.r.t.* other candidates. While on lower thresholds, the performance of our algorithms is somewhat better than DOPH and worse than BCS and MinHash. We obtained similar results for Cosine Similarity and for other datasets as well.

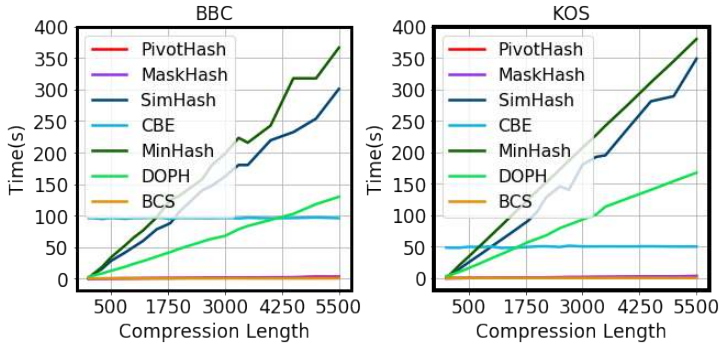
4.2. Experiment 2: Ranking

Evaluation Metric. In the ranking experiment, given a dataset and a set of query points, the task is to find all data points in the datasets that are similar to the query points, under the desired similarity measure. To do so, we split the dataset in to two parts 90% and 10% – the bigger partition is used to compress the dataset, and is referred as the *training partition*, while the second one is used to evaluate the quality of compression, and is referred as *querying partition*. We call each vector of the querying partition as a query vector. For each query vector, we compute the points in the training partition whose Jaccard/Cosine similarity is higher than a certain threshold. We used the threshold values from the set $\{0.95, 0.9, 0.85, 0.8, 0.6, 0.5, 0.2, 0.1\}$. We first did this on the uncompressed data in order to find the underlying ground truth result – for every query point compute all points that are similar to them. Then we compressed the original data, on various values of compression lengths, using the competing algorithms. To evaluate the performance of the competing algorithms, we used the *accuracy-precision-recall* ratio as our standard measure. We define it as follows. If the set $\mathcal{O}$ denotes the ground truth result (result on the uncompressed dataset), and the set $\mathcal{O}'$ denotes the results on the compressed datasets, then accuracy $= |\mathcal{O} \cap \mathcal{O}'|/|\mathcal{O} \cup \mathcal{O}'|$, precision $= |\mathcal{O} \cap \mathcal{O}'|/|\mathcal{O}'|$ and recall $= |\mathcal{O} \cap \mathcal{O}'|/|\mathcal{O}|$.

Insights. For Recall measure, we summarize our results in Figure 6. For Accuracy measure, we summarize our results in Figure 7. For Recall, on high and intermediate thresholds, our methods perform similar to the baselines but for lower thresholds, our methods have a significant advantage over them. For Accuracy and Precision parameters, our observations are similar to the $-\log(\text{MSE})$ experiments i.e. on higher and intermediate thresholds performance of our algorithms is as good as baselines, while on lower thresholds it is slightly worse than MinHash, SimHash, and BCS, but mostly better than DOPH and CBE. These results are observed for both Cosine and Jaccard Similarity. We defer extended experimental results to the appendix due to the space limitations.

Efficiency of PivotHash and MaskHash. We comment on the efficiency of our proposed algorithms with the other competing algorithms, and summarize our results on BBC and KOS datasets in Figure 4. We noted the time required to compress the original dataset using all the competing algorithms. We notice that the time required by PivotHash, MaskHash, and BCS are comparable and is negligible for all values of N and on both the datasets. Compression time of CBE is higher, however, it is independent of compression length N. For the remaining algorithms, their respective compression time grows linearly with N.

Figure 4: Comparison of compression times between BBC and KOS datasets.



To summarize, both PivotHash and MaskHash offer an efficient dimensionality reduction/sketching algorithm, which compresses a given d-dimensional binary dataset to a relatively smaller N-dimensional binary sketch, while simultaneously preserving multiple similarity measures on the same sketch. Simultaneously, they perform better, or comparable than the other competitive algorithms on both the evaluation criteria – accuracy-precision-recall and $-\log(\text{MSE})$.

Open Questions: Our work leaves the possibility of a couple of major open questions in regard to MaskHash: (i) can we suggest a *masking* heuristic which offers a more randomness efficient and succinct feature hashing, (ii) can we suggest a masking heuristic which could eliminate the assumption of well-spreadness of indices (see Definition 2), (iii) In a recent work Pratap et al. (2019) propose an improved dimensionality reduction algorithm for sparse binary data. It is interesting to see the applications of PivotHash and MaskHash in their algorithm to get a randomness efficient feature hashing for sparse binary data.

References

Alekh Agarwal, Oliveier Chapelle, Miroslav Dudík, and John Langford. A reliable effective terascale linear learning system. volume 15, pages 1111–1133, 2014. URL http://jmlr.org/papers/v15/agarwal14a.html.

Roberto J. Bayardo, Yiming Ma, and Ramakrishnan Srikant. Scaling up all pairs similarity search. In *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, pages 131–140, 2007. doi: 10.1145/1242572.1242591. URL https://doi.org/10.1145/1242572.1242591.

Andrei Z. Broder. Identifying and filtering near-duplicate documents. In *Combinatorial Pattern Matching, 11th Annual Symposium, CPM 2000, Montreal, Canada, June 21-*

23, 2000, Proceedings, pages 1–10, 2000. doi: 10.1007/3-540-45123-4_1. URL http://dx.doi.org/10.1007/3-540-45123-4_1.

Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-wise independent permutations (extended abstract). In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 327–336, 1998. doi: 10.1145/276698.276781. URL http://doi.acm.org/10.1145/276698.276781.

Sahin Buyrukbilen and Spiridon Bakiras. Secure similar document detection with simhash. In *Secure Data Management - 10th VLDB Workshop, SDM 2013, Trento, Italy, August 30, 2013, Proceedings*, pages 61–75, 2013.

Tushar Chandra, Eugene Ie, Kenneth Goldman, Tomas Lloret Llinares, Jim McFadden, Fernando Pereira, Joshua Redstone, Tal Shaked, and Yoram Singer. Sibyl: a system for large scale machine learning. *Technical report*.

Moses Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 380–388, 2002. doi: 10.1145/509907.509965. URL http://doi.acm.org/10.1145/509907.509965.

Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, Michael Mitzenmacher, Alessandro Panconesi, and Prabhakar Raghavan. On compressing social networks. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*, pages 219–228, 2009. doi: 10.1145/1557019.1557049. URL http://doi.acm.org/10.1145/1557019.1557049.

Inderjit S. Dhillon and Dharmendra S. Modha. Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42(1/2):143–175, 2001. doi: 10.1023/A:1007612920971. URL https://doi.org/10.1023/A:1007612920971.

Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, 1995.

Derek Greene and Pádraig Cunningham. Practical solutions to the problem of diagonal dominance in kernel document clustering. In *Proc. 23rd International Conference on Machine learning (ICML'06)*, pages 377–384. ACM Press, 2006.

Monika Rauch Henzinger. Finding near-duplicate web pages: a large-scale evaluation of algorithms. In *SIGIR 2006: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Seattle, Washington, USA, August 6-11, 2006*, pages 284–291, 2006. URL http://doi.acm.org/10.1145/1148170.1148222.

Zhexue Huang. Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data Mining and Knowledge Discovery*, 2(3):283–304, Sep 1998.

ISSN 1573-756X. doi: 10.1023/A:1009769707641. URL https://doi.org/10.1023/A:1009769707641.

Qixia Jiang and Maosong Sun. Semi-supervised simhash for efficient document similarity search. In *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 19-24 June, 2011, Portland, Oregon, USA*, pages 93–101, 2011. URL http://www.aclweb.org/anthology/P11-1010.

M. Lichman. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml.

Rameshwar Pratap, Raghav Kulkarni, and Ishan Sohony. Efficient dimensionality reduction for sparse binary data. In *IEEE International Conference on Big Data, Big Data 2018, Seattle, WA, USA, December 10-13, 2018*, pages 152–157, 2018a.

Rameshwar Pratap, Ishan Sohony, and Raghav Kulkarni. Efficient compression technique for sparse sets. In *Advances in Knowledge Discovery and Data Mining - 22nd Pacific-Asia Conference, PAKDD 2018, Melbourne, VIC, Australia, June 3-6, 2018, Proceedings, Part III*, pages 164–176, 2018b.

Rameshwar Pratap, Debajyoti Bera, and Karthik Revanuru. Efficient sketching algorithm for sparse binary data. In *2019 IEEE International Conference on Data Mining, ICDM 2019, Beijing, China, November 8-11, 2019*, pages 508–517, 2019. URL https://doi.org/10.1109/ICDM.2019.00061.

Anshumali Shrivastava. Optimal densification for fast and accurate minwise hashing. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 3154–3163, 2017. URL http://proceedings.mlr.press/v70/shrivastava17a.html.

Anshumali Shrivastava and Ping Li. In defense of minhash over simhash. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics, AISTATS 2014, Reykjavik, Iceland, April 22-25, 2014*, pages 886–894, 2014. URL http://jmlr.org/proceedings/papers/v33/shrivastava14.html.

Anshumali Shrivastava and Ping Li. Asymmetric minwise hashing for indexing binary inner products and set containment. In *Proceedings of the 24th International Conference on World Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015*, pages 981–991, 2015. doi: 10.1145/2736277.2741285. URL http://doi.acm.org/10.1145/2736277.2741285.

Sadhan Sood and Dmitri Loguinov. Probabilistic near-duplicate detection using simhash. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, pages 1117–1126, 2011. ISBN 978-1-4503-0717-8.

Felix X. Yu, Sanjiv Kumar, Yunchao Gong, and Shih-Fu Chang. Circulant binary embedding. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages II–946–II–954. JMLR.org, 2014. URL http://dl.acm.org/citation.cfm?id=3044805.3044998.

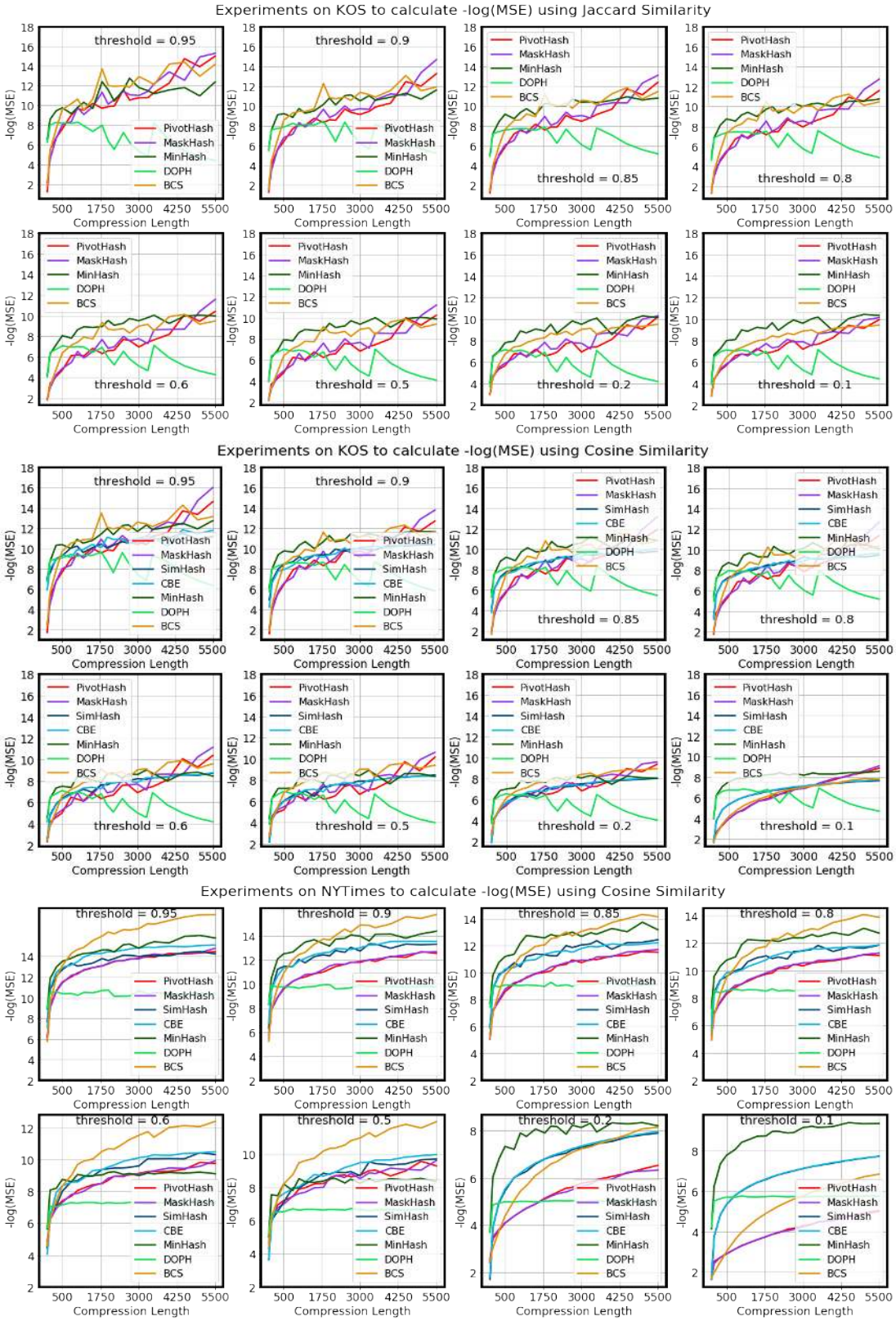


Figure 5: Comparison of $-\log(\text{MSE})$ measure on KOS and NYTimes datasets.

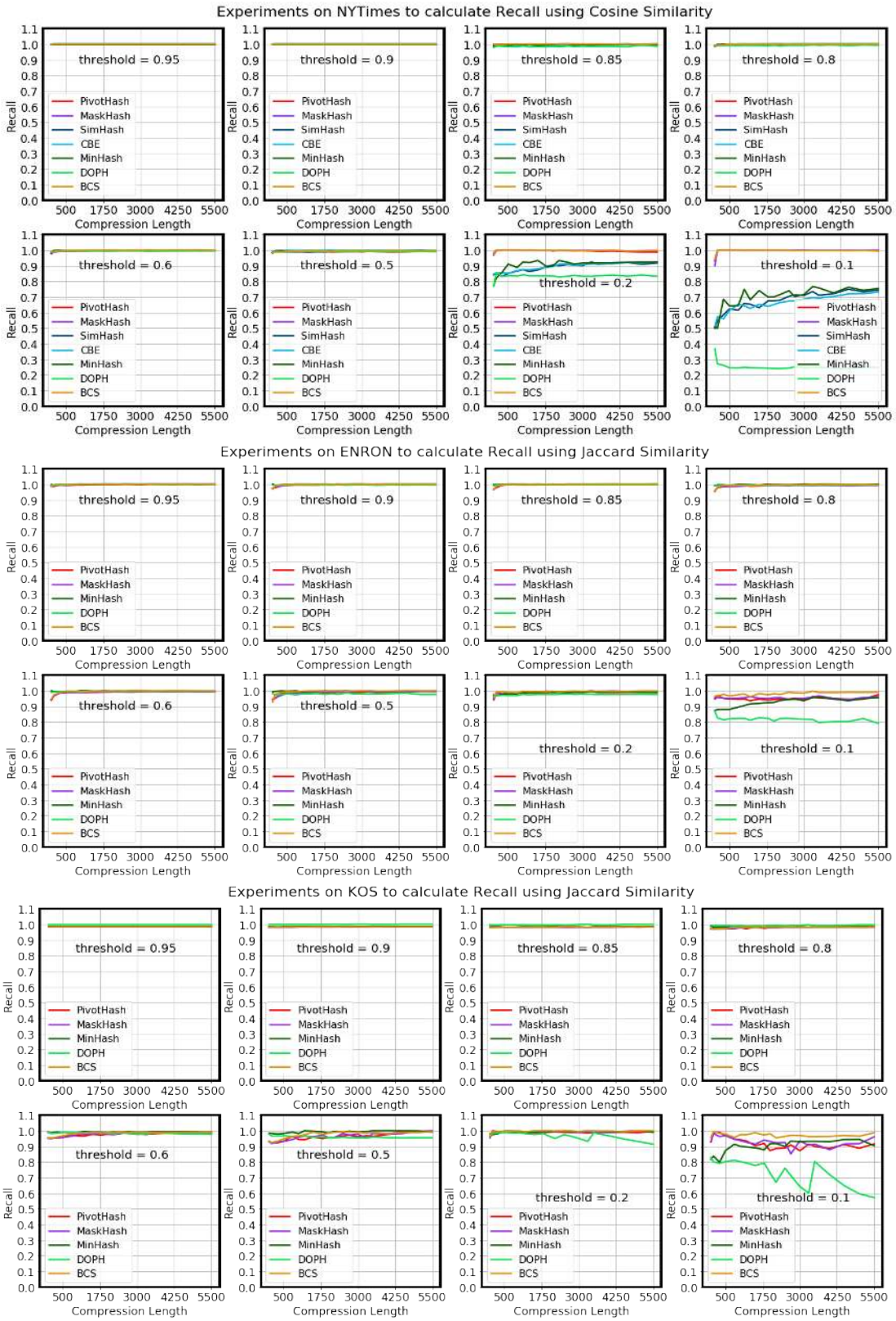


Figure 6: Comparison of Recall measure on NYTimes, Enron, and KOS datasets.

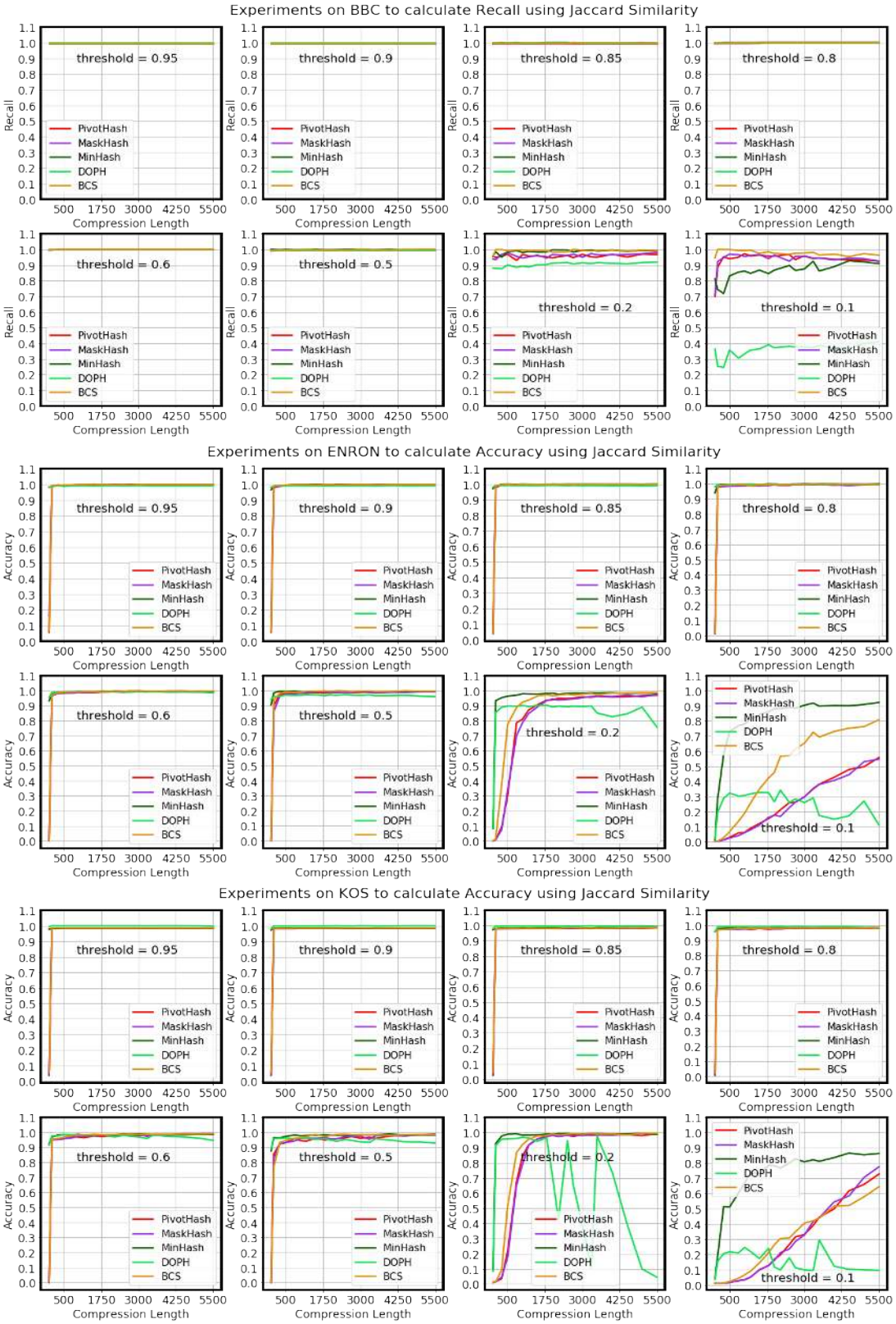


Figure 7: Comparison of Recall on BBC and Accuracy measure on ENRON and KOS datasets.