EFFICIENT COMPRESSION ALGORITHM FOR MULTIMEDIA DATA

Rameshwar Pratap[1]*, Karthik Revanuru*[2]*, Ravi Anirudh*[2]*, Raghav Kulkarni*[3]

IIT Mandi[1], IIIT Bangalore[2], Chennai Mathematical Institute[3]

ABSTRACT

In this work, we consider the problem of Cosine Similarity preserving dimensionality reduction (compression) for the sparse binary dataset. [18] suggested a compression algorithm for high dimensional, sparse, binary data for preserving Inner product and Hamming distance. In this work, we show that their proposed algorithm also works well for Cosine Similarity. We present a theoretical analysis of the dimension reduction bound and complement it with rigorous experimentation on real-world datasets. We compare our results with the state-of-the-art for the considered problem – $\mathrm{SimHash}$ [9], $\mathrm{MinHash}$ [21], Circulant Binary Embedding [25], and Densified one Permutation Hashing [20], and show that our result offers a significant saving in the compression time and the number of random bits required for the compression, and simultaneously provides comparable performance.

***Index Terms*—** Cosine Similarity, Simhash, Minhash, Jaccard Similarity.

1. INTRODUCTION

Recent advancement of the Internet, IOT etc have generated a large volume of high dimensional data. In many industrial applications, the size of datasets has exceeded the memory capacity. In WWW, there are datasets having a dimension of the order of billions [1]. In this work, we focus on binary representations of texts and images due to wide adaptation of BoW (bag-of-words) and BoVW (bag-of-visual-words) [22] techniques, respectively. These binary representation of texts and images are very useful, even in the presence of popular deep learning based methods such as word2vec [17] and convolution neural network (CNN), as they are simple, less hardware intensive, and offer compact representations. The binary representations of text datasets, due to BoW, are high dimensional and sparse as word frequency within a document follows power law – most of the words occur rarely in a document, and higher order shingles occur only once. This also holds true for BoVW representations of images as well. Such sparse binary representation of datasets is also quite common in several industrial applications [24]. Given a dataset, computing similar data points under some predefined similarity measure is a fundamental subroutine in several machine learning and data mining applications such as clustering, classification, identifying nearest neighbors, ranking, etc. However, due to the "curse of dimensionality", a brute-force method to compute similarity scores on high dimensional datasets is computationally very expensive and many times, practically impossible. Moreover, storing such a high dimensional dataset is challenging and costly. Thus, algorithms that reduce the dimension of data while preserving the similarity scores between the data-points are going to be extremely useful. Such algorithms help in processing the data faster, and in turn, help to yield inferences at a faster rate. In this work, we address this challenge and present a dimensionality reduction or alternatively called compression algorithm for binary datasets.

Cosine Similarity is a popular similarity measure for computing the similarity between a pair of documents/images. In this work, we focus on sparse binary data and consider Cosine Similarity as the similarity of interest. Let $\mathbf{u}$ and $\mathbf{v}$ be d-dimensional vector representations of two documents. Then Cosine Similarity between them is defined as $\mathrm{Cos}(\mathbf{u},\mathbf{v}) = \langle \mathbf{u},\mathbf{v}\rangle/||\mathbf{u}||_2||\mathbf{v}||_2$, where $\langle \mathbf{u},\mathbf{v}\rangle = \sum_{i=1}^{d} \mathbf{u}[i]\mathbf{v}[i]$, and $||\mathbf{u}||_2$ denote l_2 norm of vector $\mathbf{u}$. Particularly, in the binary representation of vectors we have $\mathrm{Cos}(\mathbf{u},\mathbf{v}) = \langle \mathbf{u},\mathbf{v}\rangle/\sqrt{|\mathbf{u}||\mathbf{v}|}$, where $|\mathbf{u}|$ denotes number of 1's in vector $\mathbf{u}$. The degree or amount of similarity between two documents is captured by their Cosine Similarity.

The problem of deriving compression algorithms for Cosine Similarity has been very well studied. In the following, we discuss a few notable results in that regard. [9] suggested a dimension reduction algorithm for real-valued data preserving Cosine Similarity, which is also arguably a state-of-the-art algorithm for binary data as well. Their idea is to project data points on a random vector whose entries are sampled from $\{+1,-1\}$, each with probability $1/2$. Due to [12], this dimension reduction preserves Cosine Similarity. Circulant Binary Embedding (CBE) [25] suggest a faster variant of $\mathrm{SimHash}$ while offering almost a similar performance. $\mathrm{MinHash}$ [6, 7, 5] suggest an algorithm to compress a collection of sets while preserving the Jaccard Similarity between any pair of sets. [21] suggest that in the case of sparse binary data and for preserving Cosine Similarity, $\mathrm{MinHash}$ is an optimal choice over $\mathrm{SimHash}$. Further, Densified One Permutation Hashing (DOPH) [20] which is a faster but less accurate variant of $\mathrm{MinHash}$ can be used as a compression algorithm for Cosine Similarity.

Organization of the paper: We first suggest some metrics on which a dimension reduction algorithm can be evaluated (Subsection 1.1). We then revisit the result of [18], and then building on it, we present our bound for Cosine Similarity (Subsection 1.3, Theorem 1). We present a theoreti-

cal comparison between BCS and other state-of-the-art algorithms in Subsection 1.4. In Subsection 1.5, we discuss some potential applications of our result. In Section 2, we present some necessary background for the paper. In Section 3 we give a proof of Theorem 1. In Section 4, we complement our theoretical results *via* extensive experimentation on real-world datasets. Finally, in Section 5 we conclude our discussion and state some open questions.

Notations	
N	dimension of the compressed data (compression length)
ψ	sparsity bound – upper bound on the number of 1's in binary data.
$\mathbf{u}[i]$	i-th bit position of vector $\mathbf{u}$.
$\lvert\mathbf{u}\rvert$	number of 1's in the binary vector $\mathbf{u}$
$\mathrm{Cos}(\mathbf{u},\mathbf{v})$	Cosine Similarity between binary vectors $\mathbf{u}$ and $\mathbf{v}$.
$\mathrm{JS}(\mathbf{u},\mathbf{v})$	Jaccard Similarity between binary vectors $\mathbf{u}$ and $\mathbf{v}$.
$\langle\mathbf{u},\mathbf{v}\rangle$	Inner Product between binary vectors $\mathbf{u}$ and $\mathbf{v}$.

1.1. Parameters for evaluating a compression algorithm

As also mentioned in [18, 19], the quality of a compression algorithm can be evaluated on the below parameters.

- *Randomness* is the number of random bits required.
- *Compression time* is the running time.
- The amount of *space* required to store the compressed dataset.
- *Compression length* is the dimension of data obtained after compression.

To preserve similarity/distance between every pair of data points, it would be ideal to have the values of all the four parameters as small as possible.

1.2. Revisiting compression algorithm of [18]

[18] suggested a compression algorithm BCS for binary data that reduces the dimension of data while preserving both Hamming Distance and Inner Product. The major advantage here is that the compression length (reduced dimension) depends only on the sparsity and is independent of the original dimension. We briefly discuss this algorithm:

Consider a set U of n binary vectors in d-dimensional space. Then, for any binary vector $\mathbf{u} \in \mathrm{U}$, the algorithm compresses it to an N-dimensional binary vector (say) $\mathbf{u}'$, where N to be specified later, as follows: it randomly assigns each bit position (say) $\{i\}_{i=1}^{d}$ of the original data to an integer $\{j\}_{j=1}^{\mathrm{N}}$. Further, to compute the j-th bit of the compressed vector $\mathbf{u}'$, bit-positions which have been mapped to j are checked and the parity of bits located at those positions is computed and assigned to the j-th bit position. Figure 1 illustrates this with an example. Henceforth, we refer to this analogy as BCS.

1.3. Our Result

This work contributes to show that BCS results in compression algorithm for Cosine Similarity too. We present our result as follows:

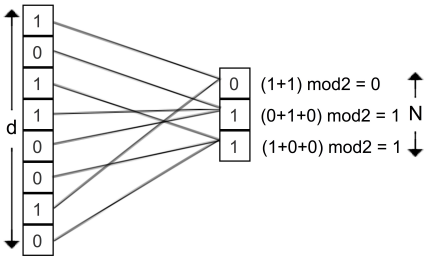

Fig. 1. Binary Compression Scheme (BCS) of [18]

Theorem 1. *Consider a pair of binary vectors* $\mathbf{u_i}, \mathbf{u_j} \in \{0,1\}^d$ *such that the maximum number of* 1*s in any vector is at most* ψ*. If we set* $\mathrm{N} = O(\psi^2)$*, and compress them into binary vectors* $\mathbf{u'}_i, \mathbf{u'}_j \in \{0,1\}^{\mathrm{N}}$ *via* BCS*, then the following holds with high probability,*

$$\mathrm{Cos}(\mathbf{u_i}, \mathbf{u_j}) = \mathrm{Cos}(\mathbf{u_i}', \mathbf{u_j}').$$

Remark 2. *A major benefit [18, 19] of* BCS *is that it works well even in a streaming setting. The only prerequisite is an upper bound on the sparsity* ψ *which requires to give a bound on the compression length* N.

Remark 3. *For a pair of* ψ*-sparse* d*-dimensional binary vectors, it is possible to represent them by the indices of the nonzeros and calculate the desired similarity measures. However, such representation depends on the original dimension* d*, and requires* $O(\psi \log d)$ *bits –* $O(\log d)$ *bits for each nonzero entries. Whereas our representation is independent of the original dimension* d*.*

1.4. Comparison between BCS and other state-of-the-art algorithms

We evaluate the quality of BCS with other competing algorithms on the parameters stated in Subsection 1.1. **Randomness.** Randomness is a key resource as the generation of random bits is a computationally expensive task. A major advantage of BCS over other state-of-the-art algorithms is that it requires significantly less amount of random bits. In BCS, each bit position of the input data is randomly assigned to one of the N buckets. This requires $O(\log \mathrm{N})$ random bits. Thus, for all the bits in d-dimension, the mapping requires $O(d \log \mathrm{N})$ amount of randomness in total. On the other hand, SimHash requires generating a random vector from $\{+1,-1\}^d$, which requires $O(d)$ random bits. This random vector results in one bit of the compressed vector after hashing. Thus for a compression length N, SimHash requires $O(d\mathrm{N})$ random bits. CBE [25] requires generating only one random vector from $\{+1,-1\}^d$, and requires $O(d)$ random bits. MinHash requires creating N permutations in d-dimension. One permutation in d dimension requires generating d random numbers each within 1 and d. Generating a number between 1 and d requires $O(\log d)$ random bits, and generating d such numbers require $O(d \log d)$ random bits. Thus, generating N such random permutations requires $O(\mathrm{N}d \log d)$ random bits. Finally,

Algorithm	No of random bits	Dim. reduction time
BCS	$O(d \log \mathrm{N})$	$O(d \log \mathrm{N} + \psi)$
DOPH [20]	$O(d \log d)$	$O(d \log d + \psi + \mathrm{N})$
CBE [25]	$O(d)$	$O(d \log d)$
SimHash [9]	$O(d\mathrm{N})$	$O((d + \psi)\mathrm{N})$
MinHash [7]	$O((d \log d)\mathrm{N})$	$O((d \log d + \psi)\mathrm{N})$

Table 1. A comparison among the candidate algorithms, on the number of random bits and the dimensionality reduction time, to get a sketch of length N of one data object. Dimensionality reduction time includes both 1) time required to generate hash function, which is of order the number of random bits, 2) time required to generate the sketch using the hash functions.

DOPH [20] requires generating only one random permutation in d-dimension. Therefore, it requires $O(d \log d)$ random bits.

Compression time. BCS is significantly faster than the state-of-the-art algorithms because generation of random bits takes significant time, thus, the relative speed-up is proportional to the savings in the number of random bits needed. Furthermore, for a dimension N, SimHash requires to scan each input vector N times – one for each random vector $\{+1, -1\}^d$, and MinHash also requires to scan each input vector N times – one for each random permutation of size d. However, BCS requires just a single scan in order to get a compressed representation in dimension N. In Section 4, we numerically quantify the speed-up of BCS over the state-of-the-art algorithms *via* experimentations on real-world datasets. We summarize the comparison of the amount of randomnesses required and the compression time in Table 1.

Space. BCS, SimHash, and CBE generate binary matrix as a result of their compression algorithm. However, MinHash and DOPH generate an integer matrix as opposed to the binary matrix generated by BCS. Therefore, for a given compression length, the space required to store the compressed data of BCS, SimHash, and CBE is $O(\log d)$ times less as compared to MinHash and DOPH. Further, the binary form of compressed data leads to faster inference for the desired task as efficient bitwise operations can be used by the algorithm.

Compression length. We numerically quantify on the compression length of BCS with respect to the state-of-the-art *via* experimentations (see Section 4) on real-world datasets.

1.5. Applications.

In cases of high dimensional, sparse data, BCS can be used to improve numerous applications where currently other state-of-the-art algorithms such as SimHash, CBE and MinHash [21] are in use.

Faster/scalable ranking of documents. Given a corpus of documents and a set of query documents, the task is to find all documents in the corpus that are similar to the query documents. This problem is a fundamental sub-routine in many applications like near-duplicate data detection [16, 23], efficient document similarity search [14] plagiarism detection [8]. SimHash and MinHash are popular choices of algorithms for such problems. However, when documents are represented in "BoW" format, BCS outperforms the other algorithms, on the parameters stated earlier. **Scalable Clustering of documents.** Spherical k-means [11] is a popular choice of clustering text documents when they are represented in "BoW" format. Usually, such representation of documents consists of high dimensional sparse binary vectors. Here, by exploiting the sparsity of documents, BCS can be more effective than other competing algorithms.

Other Applications. Beyond above applications, SimHash compression has been widely used in applications like Spam detection [4], compressing social networks [10], all pair similarity [3], collaborative filtering [2]. As in most of these cases, data objects are sparse, BCS can provide almost accurate and efficient solutions to these problems.

We experimentally validate the performance of BCS for ranking experiments on UCI [15] and BBC [13] "BoW" dataset and achieved significant improvements over other competing algorithms. We discuss this in Subsection 4. Similarly, other mentioned applications can also be validated.

2. BACKGROUND

Fact 4 (Markov's inequality)**.** *Let X be a non-negative random variable, and λ be a positive real number. Then* $\Pr[|X \geq \lambda] \leq \frac{\mathbb{E}[X]}{\lambda}$.

SimHash – a sketching algorithm for Cosine Similarity [9].

Given a vector $\mathbf{u} \in \mathbb{R}^d$, SimHash [9] 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,

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

[12] suggests that SimHash offers the following guarantee . $\Pr[\mathrm{SimHash}^{(r)}(\mathbf{u}) = \mathrm{SimHash}^{(r)}(\mathbf{v})] = 1 - \frac{\theta}{\pi}$, where $\theta = \cos^{-1}\left(\langle \mathbf{u}, \mathbf{v} \rangle / ||\mathbf{u}||_2 ||\mathbf{v}||_2\right)$. As mentioned earlier the term $\langle \mathbf{u}, \mathbf{v} \rangle / ||\mathbf{u}||_2 ||\mathbf{v}||_2$ is Cosine Similarity between vectors $\mathbf{u}$ and $\mathbf{v}$. When $\mathbf{u}, \mathbf{v}$ are binary vectors it becomes $\langle \mathbf{u}, \mathbf{v} \rangle / \sqrt{|\mathbf{u}|.|\mathbf{v}|}$. The Cosine Similarity between two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$ can be computed *via* Hamming Distance between their sketch binary vectors $\mathbf{u}', \mathbf{v}' \in \{0, 1\}^{\mathrm{N}}$. Due to [9], we have the following $\cos(\mathbf{u}, \mathbf{v}) = \cos\left[\left(\frac{\pi}{\mathrm{N}}\right) \mathrm{d_H}(\mathbf{u}', \mathbf{v}')\right]$.

3. ANALYSIS

Compressing a ψ-sparse d-dimensional binary vector into a N dimensional binary vector via BCS can be thought of as a experiment where ψ balls are thrown randomly into N bins. We call an event as *collision* if more than one ball falls into a bin. Our aim is to choose a value of N such that, with high

probability, collision can be avoided. The lemma below suggests such a bound on N which depends only on the sparsity and is independent of the original dimension d.

Lemma 5. *Consider a binary vector* $\mathbf{u} \in \{0,1\}^d$*. If we set* $\mathrm{N} = 10\psi^2$*, and compress it into a binary vector* $\mathbf{u}' \in \{0,1\}^{\mathrm{N}}$ *via* BCS*. Then the following is true with probability* 0.9*,*

$$|\mathbf{u}'| = |\mathbf{u}|.$$

Proof. Consider a vector $\mathbf{u}$ with $\mathbf{u}[i] = \mathbf{u}[j] = 1$, then we use an indicator random variable X_{ij} to indicate the event when both i and j bit position fall in the same bin. We call that even as collision. Let we denote X to be the total number of collisions, that is, $X = \Sigma_{i \neq j} X_{ij}$. The number of collisions determine the number of 1's, or the value of $|\mathbf{u}'|$. We give a bound on the expected number of collisions as follows

$$\mathbb{E}[X] = \Sigma_{i \neq j} \mathbb{E}[X_{ij}] = \Sigma_{i \neq j} \Pr[X_{ij} = 1] = \frac{1}{\mathrm{N}} \binom{\psi}{2}.$$

The above expression holds due to the linearity of expectation. If we set $\mathrm{N} = 10\psi^2$, then due to Markov Inequality (see Section 2), we have

$$\Pr[X \geq 1] \leq \frac{1}{10},$$

which implies that with probability at least 0.9, we have $|\mathbf{u}'| = |\mathbf{u}|$. □

The following lemma extends the result of Lemma 5 for a pair of ψ-sparse binary vectors $\mathbf{u_i}, \mathbf{u_j} \in \{0,1\}^d$. Suppose we get the vectors $\mathbf{u_i}', \mathbf{u_j}' \in \{0,1\}^{\mathrm{N}}$ by compressing $\mathbf{u_i}, \mathbf{u_j}$ via BCS. We derive a new binary vector $\mathbf{U_{ij}} \in \{0,1\}^d$ which is obtained by taking bitwise-AND between $\mathbf{u_i}$ and $\mathbf{u_j}$, that is, k-th bit of $\mathbf{U_{ij}}$ is obtained by taking bitwise-AND between the k-th bits of $\mathbf{u_i}$ and $\mathbf{u_j}$. Similarly, we derive $\mathbf{U'_{ij}} \in \{0,1\}^{\mathrm{N}}$ which is obtained by taking bitwise-AND between $\mathbf{u'_i}$ and $\mathbf{u'_j}$. The number of 1's in $\mathbf{U_{ij}}$ and $\mathbf{U'_{ij}}$ corresponds to the inner product between $\mathbf{u_i}$ and $\mathbf{u_j}$, and the inner product between $\mathbf{u'_i}$ and $\mathbf{u'_j}$, respectively. A proof of Lemma 6 holds by considering $\mathbf{U_{ij}}$ as an input to Lemma 5.

Lemma 6. *Consider a pair of binary vectors* $\mathbf{u_i}, \mathbf{u_j} \in \{0,1\}^d$ *such that the maximum number of* 1*s in any vector is at most* ψ*. If we set* $\mathrm{N} = O(\psi^2)$*, and compress them into binary vectors* $\mathbf{u_i}', \mathbf{u_j}' \in \{0,1\}^{\mathrm{N}}$ *via* BCS*, then the following holds with high probability,*

$$\langle \mathbf{u_i}, \mathbf{u_j} \rangle = \langle \mathbf{u_i}', \mathbf{u_j}' \rangle.$$

A proof of the Theorem 1 follows easily due to lemma 5 and lemma 6 and the definition of Cosine Similarity.

The following corollary extends the result of Theorem 1 for a set of n binary vectors.

Proof of Lemma 6 is similar to the proof of Theorem 2 of [18]. However, in this paper, we present a simplified version of the proof.

Corollary 7. *Consider a set* U *of binary vectors* $\{\mathbf{u_i}\}_{i=1}^n \subseteq \{0,1\}^d$*such that the maximum number of* 1*s in any vector is at most* ψ*. If we set* $\mathrm{N} = O(\psi^2 \log n)$*, and compress them into a set* U′ *of binary vectors* $\{\mathbf{u'_i}\}_{i=1}^n \subseteq \{0,1\}^{\mathrm{N}}$ *via* BCS*. Then for all* $\mathbf{u_i}, \mathbf{u_j} \in$ U*, the following is true w.h.p.*

$$\mathrm{Cos}(\mathbf{u_i}, \mathbf{u_j}) = \mathrm{Cos}(\mathbf{u_i}', \mathbf{u_j}').$$

4. EXPERIMENTAL EVALUATION

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 [15]; and BBC News Datasets (number of points = 2225, dimension = 9635) [13]. We considered the entire corpus of KOS and BBC News datasets, while for NYTimes, ENRON datasets we sampled 5000 data points.

Competing Algorithms: We compared the performance of BCS with the state-of-the-art sketching algorithms for the task such as SimHash [9], Circulant Binary Embedding (CBE) [25] – a faster variant of SimHash, MinHash [21], and Densified One Permutation Hashing (DOPH) [20] – a faster variant of MinHash. For a pair of binary vectors, we calculate the Cosine Similarity using DOPH as follows: we first calculate the Jaccard Similarity using DOPH, and then using the result of [21] we find an estimate of their Cosine Similarity.

4.1. Experiment 1: Accuracy of Estimation

We first evaluated the fidelity of the estimate of BCS. We discuss it below.

Evaluation Metric. To understand the behaviour of BCS on various similarity regime, we extracted samples of similar pairs (on various thresholds) from our datasets. For this purpose, we enumerated over all the pairs, and extracted those whose Cosine Similarities were higher than the given thresholds $\in \{0.95, \ldots, 0.1\}$. We used mean square error (MSE) as our evaluation criteria. Using all the candidate algorithms we compressed the datasets to various values of compression length N. We then calculated their MSE values for different values of N. We illustrate this as follows. For example, in order to calculate the MSE of BCS *w.r.t.* the ground truth result, for every pair of data points, we calculated the square of the difference between their estimated Cosine Similarity obtained after compression, and the corresponding ground truth Cosine Similarity. We added these values for all such pairs and calculated its mean. The value of this quantity is at most 1, and we computed the negative logarithm base e of this quantity. A smaller MSE corresponds to a larger $-\log(\mathrm{MSE})$, therefore, a higher value $-\log(\mathrm{MSE})$ is an indication of better performance.

Insights. We summarize our results in Figure 5. The main advantage of BCS was observed on higher and intermediate thresholds, where BCS tend to be performing significantly better/comparable than all the competing algorithms. On low threshold values such as $\{0.2, 0.1\}$ performance of BCS was observed better than DOPH [20] and simultaneously comparable with respect to the remaining competitive algorithms.

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 Cosine Similarity. To do so, we split the dataset into two parts 90% and 10% – the bigger partition is used to compress the data and is referred as the *training partition*, while the second one is used to evaluate the quality of the 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 vectors in the training partition whose Cosine similarity is higher than a certain threshold (ranging from 0.1 to 0.95). We first do this on the uncompressed data in order to find the underlying ground truth result – for every query vector compute all vectors that are similar to them. We calculate the ground truth result using a brute-force linear search algorithm on the original data. Then we compress the original data, on various values of compression lengths, using all the competing algorithms, and compute all vectors that are similar to the query vector. To evaluate the performance of the competing algorithms, we used the *accuracy-precision-recall* ratio as our standard measure. 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 = $|\mathcal{O} \cap \mathcal{O}'|/|\mathcal{O}|$. For each query vector, we calculate the accuracy/precision/recall of all the competing algorithms using the approach described above, on various values of compression length. This gave us the accuracy/precision/recall of compression of that particular query vector. We repeat this for every vector in the querying partition, and take the average, and we plot the average accuracy/precision/recall for each value in support threshold and compression length. We also note down the corresponding compression time on each of the compression lengths for all the competing algorithms.

Insights. We summarize the comparison of the recall measure in Figure 4. A major advantage of BCS on the recall measure was observed on low-threshold values where it significantly outperformed with respect to the other candidate algorithms. On higher and intermediate threshold values performance of BCS was observed similar to the other competing algorithms. We summarize the comparison of the accuracy measure in Figure 3, 4. Here again, on higher and intermediate threshold values performance of BCS on the accuracy measure was observed similar to the other competing algorithms. However, on low threshold values such as $\{0.1, 0.2\}$ the performance of BCS was moderate with respect to other the candidate algorithms. Performance of BCS on the precision measure was observed similar to the accuracy measure. We defer the plots to the full version of the paper.

Efficiency of BCS**.** We comment on the efficiency of BCS with the other competing algorithms and summarize the results in Figure 2. We noted the time required to compress the original dataset using all the competing algorithms, and notice that the time required by BCS is negligible for all values of compression length N, on all the datasets. Compression time of CBE is higher than ours, however, it is independent of the N. For the remaining algorithms, their respective compression time grows linearly with N.

We further give a numerical speedup of BCS *w.r.t.* other algorithms and summarize the results in Table 2. To do so, we pick the compression length value such that by ignoring the low thresholds results, the minimum accuracy across all the algorithms is at least 0.9. It turns out that the minimum compression length of 500 holds the purpose for all the four datasets. We obtained a significant speed up in the compression time. For BBC dataset, on the compression length 500, BCS was $108.6\times$ faster than SimHash, $370.9\times$ faster than CBE, $130.2\times$ faster than MinHash, and $48.4\times$ faster than DOPH. Please note that SimHash is faster than CBE on the compression length 500. However, the compression time for CBE remains constant even for higher compression lengths whereas the compression time for SimHash increases linearly with the same. Moreover, after ignoring the plots corresponding to the low thresholds $\in \{0.1, 0.2, 0.3, 0.4\}$, the average accuracy of BCS was 0.99, while for SimHash, CBE, MinHash and DOPH it was $0.98, 0.98, 0.99$ and 0.94 only. We observed a similar performance on the other datasets as well.

5. CONCLUDING REMARKS AND OPEN QUESTIONS

We showed that BCS is able to compress sparse, high-dimensional binary data while approximating the Cosine Similarity. Our algorithm is considerably faster than the "state-of-the-art" SimHash, MinHash, CBE [25], and DOPH [20] and also maintains almost equal accuracy while significantly reducing the amount of randomness required. Moreover, the compressed representation obtained from BCS is in binary form, as opposed to the integer in case of MinHash and DOPH, due to which the space required to store the compressed data is reduced, and consequently leads to a faster search on the compressed representation. Another major advantage of BCS is that its compression bound is independent of the dimensions of the data, and only grows polynomially with the sparsity and logarithmically with the number of data points. Our work leaves the possibility of several open questions : a) improving the dimensionality reduction bound and giving a matching lower bound on the same, and b) giving a variance analysis on the cost. Finally, given the simplicity of our method, we hope that it will be adopted in practice.

For both the experiments, even on the KOS and BBC dataset, we obtained a similar comparison between BCS and other competing algorithms. We defer the plots to the full version of the paper.

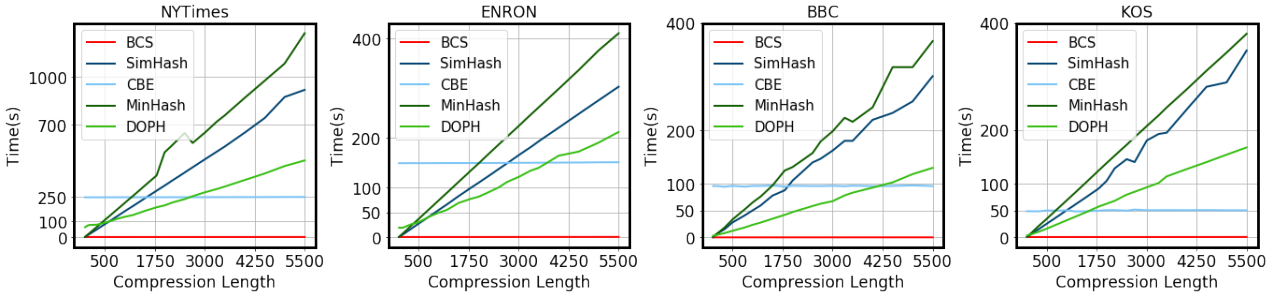

Fig. 2. Comparison on the Compression Time for Ranking Experiments.

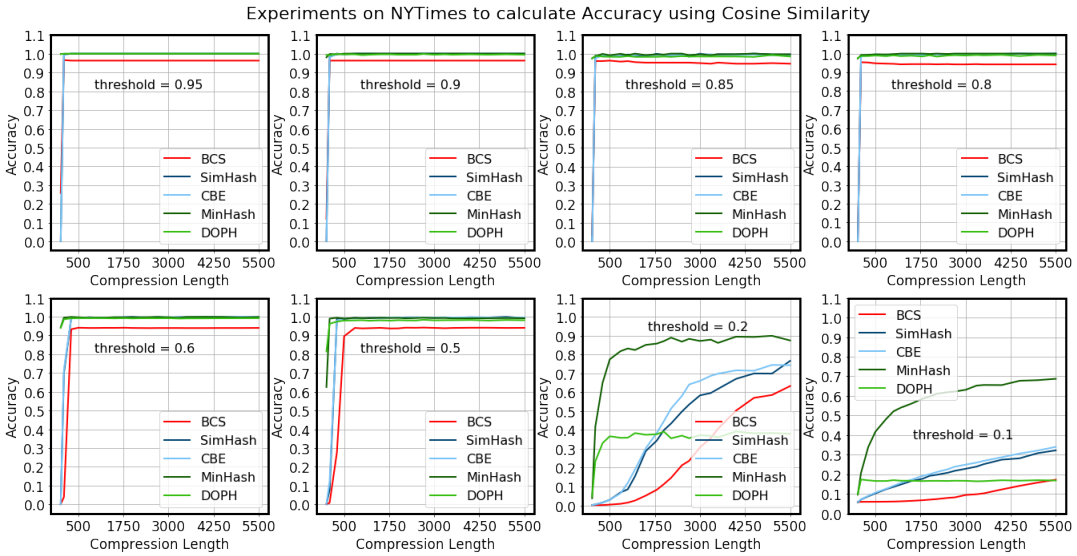

Fig. 3. Comparison of Accuracy measure on NYTimes datasets.

Dataset	Compression length	Speedup of BCS *w.r.t.* SimHash	Speedup of BCS *w.r.t.* CBE	Speedup of BCS *w.r.t.* MinHash	Speedup of BCS *w.r.t.* DOPH
BBC	500	**108.66**×	**370.9**×	**130.2**×	**48.4**×
Enron	500	**43.3**×	**233.6**×	**58.1**×	**48.01**×
KOS	500	**50.8**×	**100.8**×	**69.2**×	**32.3**×
NYTimes	500	**51.03**×	**158.16**×	**67.66**×	**56.87**×

Table 2. Comparison among BCS, SimHash, CBE, MinHash and DOPH on real-world datasets experiments. For comparison, we set the minimum accuracy as 0.94 which is obtained at the compression length 500 for all the algorithms across all the datasets. We ignored the results of the low thresholds $\in \{0.1, 0.2, 0.3, 0.4\}$ as accuracies on them were low across all the candidate algorithms.

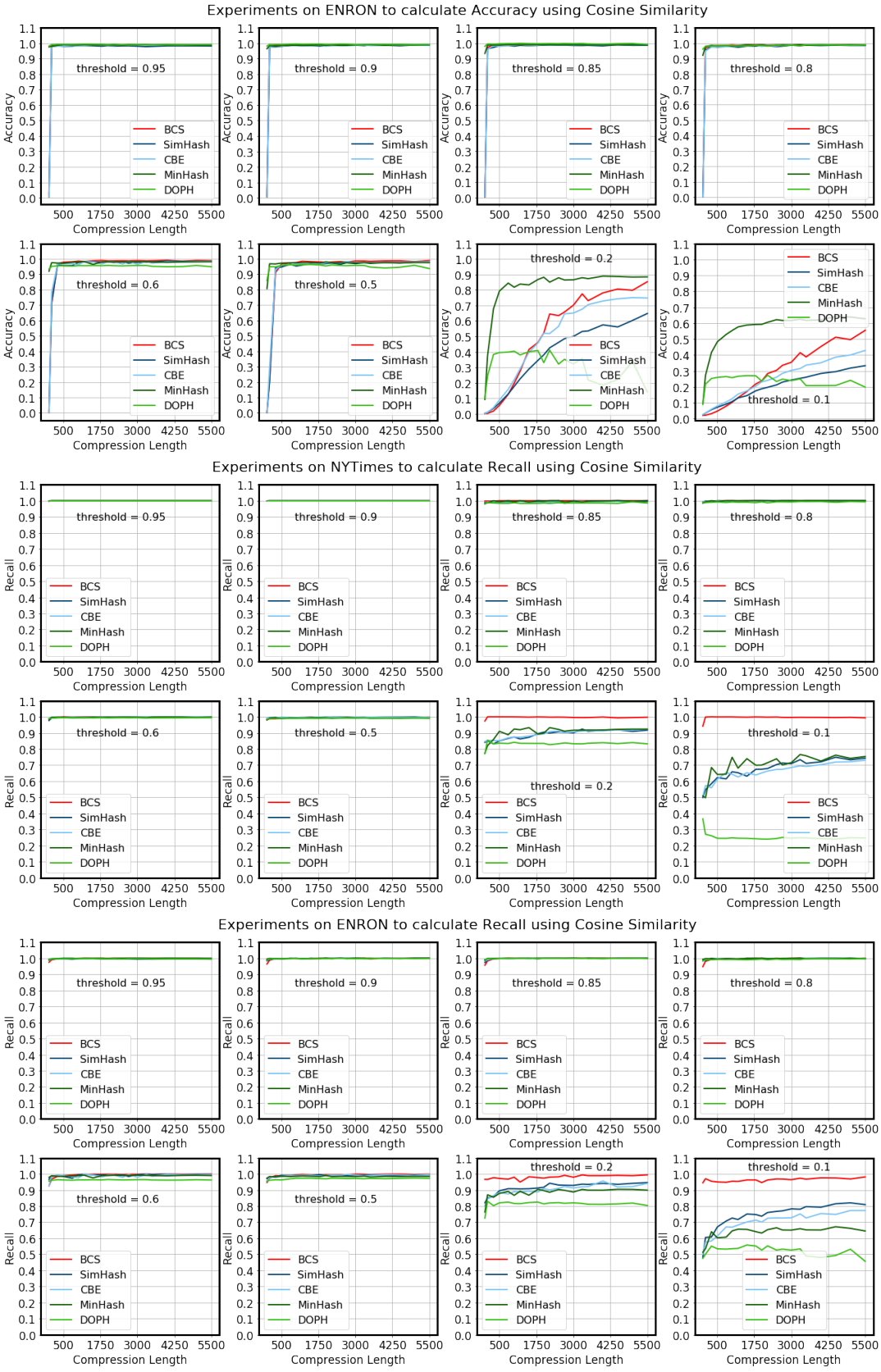

Fig. 4. Comparison of Accuracy measure on ENRON datasets, and Recall measure on NYTimes and ENRON datasets.

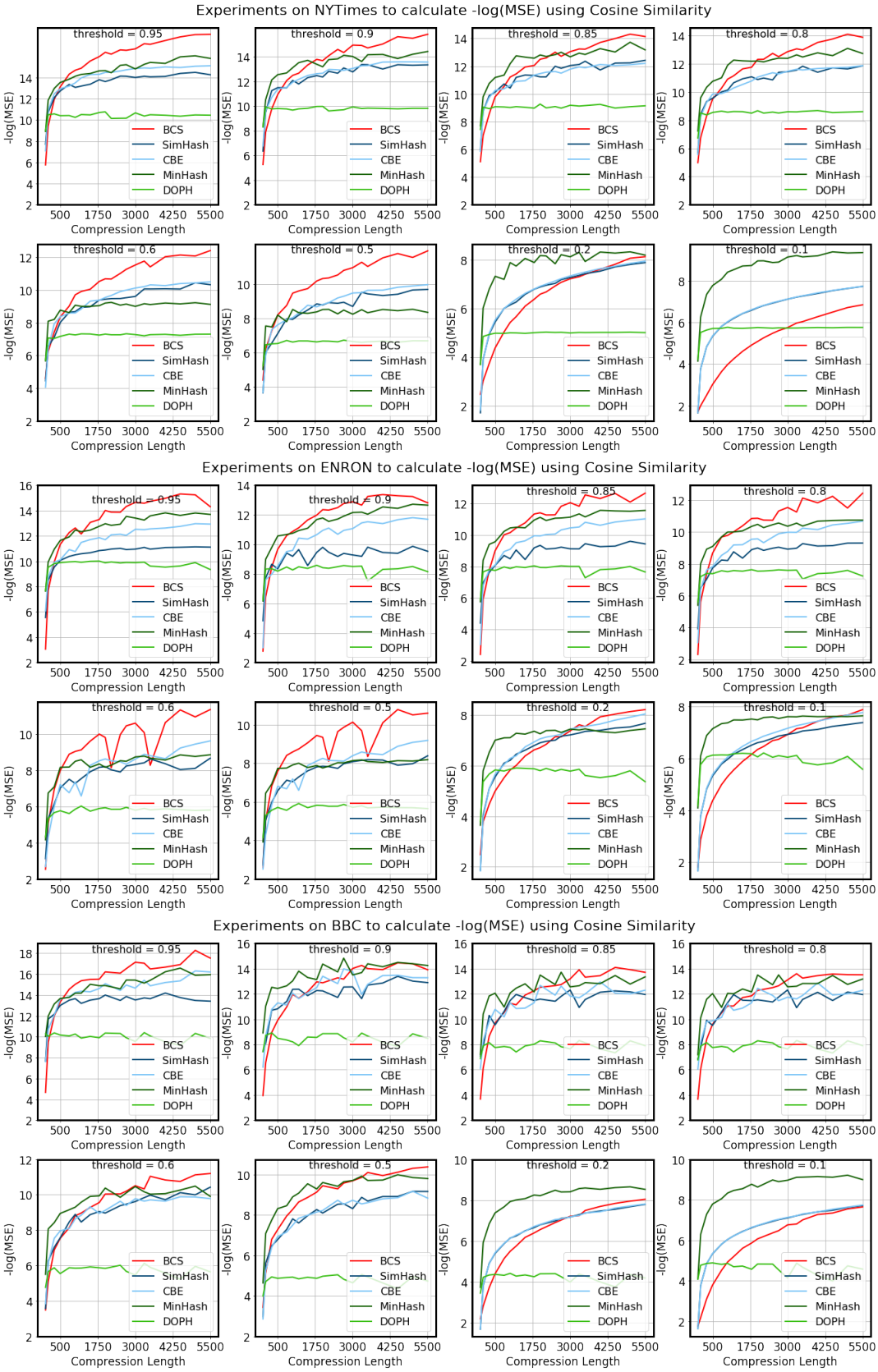

Fig. 5. Comparison of $\log(\text{MSE})$ measure on NYTimes, ENRON and BBC datasets. A higher value is an indication of better performance.

6. REFERENCES

[1] Alekh Agarwal, Oliveier Chapelle, Miroslav Dudík, and John Langford. A reliable effective terascale linear learning system. *Journal of Machine Learning Research*, 15:1111–1133, 2014.

[2] Yoram Bachrach, Ely Porat, and Jeffrey S. Rosenschein. Sketching techniques for collaborative filtering. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 2016–2021, 2009.

[3] 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.

[4] Andrei Z Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*, pages 21–29. IEEE, 1997.

[5] 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.

[6] Andrei Z. Broder. Min-wise independent permutations: Theory and practice. In *Automata, Languages and Programming, 27th International Colloquium, ICALP 2000, Geneva, Switzerland, July 9-15, 2000, Proceedings*, page 808, 2000.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] Inderjit S. Dhillon and Dharmendra S. Modha. Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42(1/2):143–175, 2001.

[12] 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.

[13] 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.

[14] 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.

[15] M. Lichman. UCI machine learning repository, 2013.

[16] Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. Detecting near-duplicates for web crawling. In *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, pages 141–150, 2007.

[17] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 3111–3119, 2013.

[18] 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, 2018.

[19] 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, 2018.

[20] 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.

[21] 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.

[22] Josef Sivic and Andrew Zisserman. Video google: A text retrieval approach to object matching in videos. In *ICCV*, pages 1470–1477. IEEE Computer Society, 2003.

[23] 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, New York, NY, USA, 2011. ACM.

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

[25] 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.