Consistent Weighted Sampling Made More Practical

Wei Wu[1], Bin Li[2], Ling Chen[1], Chengqi Zhang[1]
[1]CAI, University of Technology Sydney, Ultimo NSW 2007, Australia
william.third.wu@gmail.com
{ling.chen, chengqi.zhang}@uts.edu.au
[2]Data61, CSIRO, Eveleigh NSW 2015, Australia
bin.li@data61.csiro.au

ABSTRACT

Min-Hash, which is widely used for efficiently estimating similarities of bag-of-words represented data, plays an increasingly important role in the era of big data. It has been extended to deal with real-value weighted sets – Improved Consistent Weighted Sampling (ICWS) is considered as the state-of-the-art for this problem. In this paper, we propose a Practical CWS (PCWS) algorithm. We first transform the original form of ICWS into an equivalent expression, based on which we find some interesting properties that inspire us to make the ICWS algorithm simpler and more efficient in both space and time complexities. PCWS is not only mathematically equivalent to ICWS and preserves the same theoretical properties, but also saves 20% memory footprint and substantial computational cost compared to ICWS. The experimental results on a number of real-world text data sets demonstrate that PCWS obtains the same (even better) classification and retrieval performance as ICWS with $1/5 \sim 1/3$ reduced empirical runtime.

Keywords

Weighted Min-Hash; Consistent Weighted Sampling; LSH

1. INTRODUCTION

Nowadays, data are growing explosively on the Web. In 2008, Google processed 20PB data every day [23]; in 2012, Google received more than 2 million search queries per minute; while in 2014 this number had been more than doubled [9, 27]. Social networking services also have to face the data explosion challenge. More than 500TB of data need to be processed in Facebook every day [28] and 7 million check-in records are produced in Foursquare every day [30] – Big data have been driving data mining research in both academia and industry [7, 22]. A fundamental research that underpins many high-level applications is to efficiently compute similarities (or distances) of data. Based on the data similarity, one can further conduct information retrieval, classification, and many other data mining tasks. However, the standard similarity computation has been incompetent for big data due to the "3V" nature (volume, velocity and variety). For example, in text mining, it is intractable to enumerate the complete feature set (e.g., over 10^8 elements in the case of 5-grams in the original data [22]). Therefore, it is urgent to develop efficient yet accurate similarity estimation algorithms.

A typical solution to the aforementioned problem is to approximate data similarities using a family of Locality Sensitive Hashing (LSH) techniques [12]. By adopting a collection of hash functions to map similar objects to the same hash code with higher probability than dissimilar ones, LSH is able to approximate certain similarity (or distance) measures. One can thus efficiently and, in many cases, unbiasedly calculate the similarity (or distances) between objects. Many LSH schemes have been successively proposed, e.g., Min-Hash for estimating the Jaccard similarity [1], Sim-Hash for estimating angle-based distance [3, 20], and LSH with p-stable distribution for estimating l_p distance [6]. In particular, Min-Hash has been widely used for approximating the similarity of documents which are usually represented as sets or bags-of-words. Recently, some variations of Min-Hash have further improved its efficiency. For example, b-bit Min-Hash [16] and odd sketches [21] remarkably improve the storage efficiency by storing only b bits of each hash value; while one-permutation Min-Hash [17, 25] employs only one permutation to reduce the computational workload.

Although the set similarity can be efficiently estimated based on the standard Min-Hash scheme and its variations, the target data are restricted to binary sets. However, in most real-world scenarios, weighted sets are more common. For example, a *tf-idf* value is assigned to each word to represent its importance in a collection of documents. As Min-Hash treats all the elements in a set equally for computing the Jaccard similarity, it cannot handle weighted sets properly. To address the limitation, weighted Min-Hash algorithms have been explored to approximate the generalized Jaccard similarity [11], which is used for measuring the similarity of weighted sets. Roughly speaking, existing works on weighted Min-Hash algorithms can be classified into *quantization-based* and *sampling-based* approaches.

Quantization-based methods quantize each weighted element into a number of distinct and equal-sized subelements. The resulting subelements are treated equally just like independent elements in the universal set. One can simply apply the standard Min-Hash scheme to the collection of subelements. Although in [8] an improved integer-value weighted

WWW 2017, April 3–7, 2017, Perth, Australia.
ACM 978-1-4503-4913-0/17/04.
http://dx.doi.org/10.1145/3038912.3052598

Min-Hash algorithm is proposed to avoid computing the hash value for each individual subelement, the algorithm is still inefficient for dealing with real-value weighted sets because each weight must be transformed into integer weight by multiplying a large constant, which dramatically expands the universal set with numerous quantized subelements.

To avoid computing hash values for all the subelements, researchers have resorted to sampling-based methods. In [24], a uniform sampling algorithm is proposed, which strictly complies with the definition of the generalized Jaccard similarity. However, this algorithm requires knowing the upper bound of each element in the universal set in advance, which makes it impractical in real-world applications. In [5], a sampling method is derived through the distribution of the minimum of a set of random variables for integer weighted sets, which results in a biased estimator. Later, Consistent Weighted Sampling (CWS) [19, 10] and Improved CWS (ICWS) [13] are proposed to remarkably improve the efficiency of weighted Min-Hash by introducing the notion of "active index" [8] into real-value weighted elements (the "active indices" on a weighted element are independently sampled as a sequence of subelements whose hash values monotonically decrease [29]). Thus far, ICWS [13], as an efficient and unbiased estimator of the generalized Jaccard similarity, is recognized as the state-of-the-art. Recently, [15] approximates ICWS by simply discarding one component of the Min-Hash values.

The mystery of ICWS [13] is that it implicitly constructs an exponential distribution for each real-value weighted element using only two "active indices". The minimum of the collection of exponential variables of all the elements yields one of the two components of the real-value weighted Min-Hash code (the other component can be easily obtained), meanwhile complying with the uniformity of the Min-Hash scheme – This enables ICWS to produce Min-Hash code for a real-value weight with computational complexity that is independent of the number of quantized subelements.

In this paper, we aim to further improve the efficiency of ICWS. By transforming the original form of ICWS into a different but mathematically equivalent expression, we find some interesting properties that inspire us to further make ICWS more practical. Thus, we propose the Practical CWS (PCWS) algorithm, which is simpler and more efficient in both space and time complexities compared to ICWS. Furthermore, we theoretically prove the uniformity and consistency of the PCWS algorithm for real-value weighted Min-Hash. We conduct extensive empirical tests on a number of real-world text data sets to compare the proposed PCWS algorithm and the state-of-the-arts for classification and retrieval. In summary, our contributions are three-fold:

1. We transform the original form of ICWS [13] into a simpler and easy-to-understand expression, uncovering the working mechanism of ICWS.

2. We propose the PCWS algorithm, which is mathematically equivalent to ICWS and preserves the same theoretical properties as ICWS.

3. PCWS has the same (even better) classification and retrieval performance as ICWS while saving 20% memory footprint and $1/5 \sim 1/3$ empirical runtime, which makes it more practical.

The remainder of the paper is organized as follows: Section 2 introduces the definitions of (generalized) Jaccard similarity, weighted Min-Hash, and ICWS [13]. Then, in Section 3 we revisit ICWS and present its equivalent version, based on which we propose our PCWS algorithm and give its theoretical analysis. The experimental results are presented in Section 4 and the paper is concluded in Section 5.

2. PRELIMINARIES

In this section, we first give some notations which will be used throughout the paper. Next we will introduce the Min-Hash scheme and present the state-of-the-art method, ICWS [13], for weighted Min-Hash.

Given a universal set $\mathcal{U} = (U_1, U_2, \cdots, U_n)$ and its subset $\mathcal{S} \subseteq \mathcal{U}$, if for any element $S_k \in \mathcal{S}$, $S_k = 1$ or $S_k = 0$, then we call $\mathcal{S}$ a binary set; if for any element $S_k \in \mathcal{S}$, $S_k \geq 0$, then we call $\mathcal{S}$ a weighted set. A typical example of universal set is the dictionary of a collection of documents, where each element corresponds to a word.

For hashing a binary set $\mathcal{S}$, a Min-Hash scheme assigns a hash value to each element S_k, $h : k \mapsto v_k$. By contrast, for hashing a weighted set, there is a different form of hash function: $h : (k, y_k) \mapsto v_{k,y}$, where $y_k \in [0, S_k]$. A random permutation (or sampling) process returns the first (or uniformly selected) k from a binary set or (k, y_k) from a weighted set. If the set is sampled D times, we will obtain a fingerprint with D hash values.

2.1 The Min-Hash Scheme

DEFINITION 1 (MIN-HASH [1]). *Given a universal set $\mathcal{U}$ and a subset $\mathcal{S} \subseteq \mathcal{U}$, Min-Hash is generated as follows: Assuming a set of D hash functions (or D random permutations), $\{\pi_d\}_{d=1}^{D}$, are applied to $\mathcal{U}$, the elements in $\mathcal{S}$ which have the minimum hash value in each hash function (or which are placed in the first position of each permutation), $\{\min(\pi_d(\mathcal{S}))\}_{d=1}^{D}$, would be the Min-Hashes of $\mathcal{S}$.*

Min-Hash [1] is an approximate algorithm for computing the Jaccard similarity of two sets. It is proved that the probability of two sets, $\mathcal{S}$ and $\mathcal{T}$, to generate the same Min-Hash value (hash collision) is exactly equal to the Jaccard similarity of the two sets:

$$\Pr\big(\min(\pi_d(\mathcal{S})) = \min(\pi_d(\mathcal{T}))\big) = J(\mathcal{S}, \mathcal{T}) = \frac{|\mathcal{S} \cap \mathcal{T}|}{|\mathcal{S} \cup \mathcal{T}|}.$$

The Jaccard similarity is simple and effective in many applications, especially for document analysis based on the bag-of-words representations [26].

From the above Min-Hash scheme we can see that all the elements in $\mathcal{U}$ are considered equally because all the elements can be mapped to the minimum hash value with equal probability. To sample a weighted set based on the standard Min-Hash scheme, the weights, which indicates different importance of each element, will be simply replaced with 1 or 0, which in turn leads to serious information loss.

In most real-world scenarios, weighted sets are more commonly seen than binary sets. For example, a document is usually represented as a *tf-idf* set. In order to reasonably compute the similarity of two weighted sets, the generalized Jaccard similarity was introduced in [11]. Considering two

weighted sets, $\mathcal{S}$ and $\mathcal{T}$, the generalized Jaccard similarity is defined as

$$generalizedJ(\mathcal{S},\mathcal{T}) = \frac{\sum_k \min(S_k, T_k)}{\sum_k \max(S_k, T_k)}.$$

2.2 Improved CWS

Based on the generalized Jaccard similarity, some weighted Min-Hash algorithms have been proposed [8, 19, 13, 24]. To the best of our knowledge, Improved Consistent Weighted Sampling (ICWS) [13] is remarkable in both theory and practice, and considered as the state-of-the-art method for weighted Min-Hash [15].

DEFINITION 2 (CONSISTENT WEIGHTED SAMPLING [19]). *Given a weighted set $\mathcal{S} = \{S_1, \ldots, S_n\}$, where $S_k \geq 0$ for $k \in \{1, \ldots, n\}$, Consistent Weighted Sampling (CWS) produces a sample $(k, y_k) : 0 \leq y_k \leq S_k$, which is uniform and consistent.*

- **Uniformity:** The subelement (k, y_k) should be uniformly sampled from $\bigcup_k(\{k\} \times [0, S_k])$, i.e., the probability of selecting the k-th element is proportion to S_k, and y_k is uniformly distributed on $[0, S_k]$.
- **Consistency:** Given two non-empty weighted sets, $\mathcal{S}$ and $\mathcal{T}$, if $\forall k, T_k \leq S_k$, a subelement (k, y_k) is selected from $\mathcal{S}$ and satisfies $y_k \leq T_k$, then (k, y_k) will also be selected from $\mathcal{T}$.

CWS has the following property

$$\Pr[\text{CWS}(\mathcal{S}) = \text{CWS}(\mathcal{T})] = generalizedJ(\mathcal{S},\mathcal{T}).$$

In the following we briefly review how to deduce ICWS to meet the two conditions of CWS, uniformity and consistency. Firstly we note that the exponential distribution has an important and interesting property about the distribution of the minimum of exponential random variables: Let $X_1, \ldots, X_n$ be independently exponentially distributed random variables, with the parameters being $\lambda_1, \ldots, \lambda_n$, respectively, then we have $\Pr(X_k = \min\{X_1, \ldots, X_n\}) = \frac{\lambda_k}{\lambda_1 + \ldots + \lambda_n}$. Therefore, one can employ this property to implement the uniformity for CWS – If each hash value $a_{k'}$ of the k'-th element is drawn from an exponential distribution parameterized with its corresponding weight, i.e., $a_{k'} \sim \text{Exp}(S_{k'})$, the minimum hash value a_k will be sampled in proportion to S_k,

$$\Pr(a_k = \min\{a_1, \ldots, a_n\}) = \frac{S_k}{\sum_{k'} S_{k'}}. \tag{1}$$

In addition, note that k and y_k are mutually independent, which means that a_k and y_k are mutually independent as well. Formally, the condition of uniformity in terms of (y_k, a_k) can be expressed as

$$p(y_k, a_k) = \frac{1}{S_k}(S_k \mathrm{e}^{-S_k a_k}) = p(y_k)p(a_k), \tag{2}$$

where $y_k \sim \text{Uniform}(0, S_k)$ and $a_k \sim \text{Exp}(S_k)$.

In order to make y_k uniformly distributed in $[0, S_k]$, ICWS employs the following equation

$$\ln y_k = \ln S_k - r_k b_k, \tag{3}$$

where $r_k \sim \text{Gamma}(2, 1)$ and $b_k \sim \text{Uniform}(0, 1)$. Eq. (3) is used to prove the uniformity in [13]. However, in its algorithmic implementation of ICWS, the above equation is replaced with the following equation

$$\ln y_k = r_k \left(\left\lfloor \frac{\ln S_k}{r_k} + \beta_k \right\rfloor - \beta_k \right), \tag{4}$$

where $\beta_k \sim \text{Uniform}(0, 1)$. It is indicated in [13] that via Eq. (4), $\ln y_k$ is sampled from the same uniform distribution in $[\ln S_k - r_k, \ln S_k]$ as that sampled through Eq. (3). The floor function and the uniform random variable β_k in Eq. (4) ensures that a fixed y_k is sampled in an interval of r_k. Obviously, Eq. (4) gives rise to consistency because small changes in S_k cannot affect the value of y_k.

In order to sample k in proportion to its corresponding weight S_k, in addition to an "active index", y_k, ICWS [13] introduces a second "active index", $z_k \in [S_k, +\infty)$, and builds the relationship among y_k, z_k and r_k:

$$r_k = \ln z_k - \ln y_k. \tag{5}$$

Furthermore, by using the two "active indices", ICWS implicitly constructs an exponential distribution parameterized with S_k, i.e., $a_k \sim \text{Exp}(S_k)$:

$$a_k = \frac{c_k}{z_k}, \tag{6}$$

where $c_k \sim \text{Gamma}(2, 1)$.

Essentially, ICWS proceeds the sampling process as follows: It first samples y_k using Eq. (7), which is derived from Eq. (4). Then the sampled y_k, as an independent variable, is fed into Eq. (8), which is derived from Eqs. (5) and (6), and outputs a hash value conforming to the exponential distribution parameterized with the corresponding weight S_k.

$$y_k = \exp\left(r_k \left(\left\lfloor \frac{\ln S_k}{r_k} + \beta_k \right\rfloor - \beta_k \right) \right), \tag{7}$$

$$a_k = \frac{c_k}{y_k \exp(r_k)}. \tag{8}$$

In ICWS [13], a Min-Hash code (y_k, a_k) for a weighted element S_k is calculated using the hash functions, Eqs. (7) and (8), respectively. The hash functions seem already simple but mysterious. Why does Eq. (8) work? Can we further improve the efficiency of ICWS?

3. PRACTICAL CWS

In this section we propose a more practical algorithm for consistent weighted sampling. First, we revisit ICWS [13] and transform its original form Eq. (8) for sampling a_k into an equivalent version. Based on this equivalent form, we find some interesting properties which inspire us to make the ICWS algorithm simpler and more efficient in both space and time complexities. We also demonstrate that the proposed PCWS algorithm still complies with the uniformity and consistency of CWS [19].

3.1 ICWS Revisited

Recall that in ICWS [13] one need to sample two Gamma variables, $r_k \sim \text{Gamma}(2, 1)$ and $c_k \sim \text{Gamma}(2, 1)$, in Eqs. (7) and (8) to directly generate y_k and a_k. In fact, the simplest programming implementation[1] to generate a random variable x from $\text{Gamma}(2, 1)$ is to first sample two

[1]Some programming languages may provide built-in functions to generate random variable from $\text{Gamma}(\alpha, \beta)$, which however has a complexity no better than this method in the case of $\alpha = 2, \beta = 1$.

uniform random variables, $u_1, u_2 \sim \text{Uniform}(0,1)$ and then conduct a transformation as $x = -\ln(u_1 u_2)$. Thus if we represent the two independent Gamma variables r_k and c_k in terms of uniform random variables, $r_k = -\ln(u_{k1}u_{k2})$ and $c_k = -\ln(v_{k1}v_{k2})$, where $u_{k1}, u_{k2}, v_{k1}, v_{k2} \sim \text{Uniform}(0,1)$, we can obtain an equivalent version of Eqs. (7) and (8):

$$y_k = \exp(-\ln(u_{k1}u_{k2})(t_k - \beta_k)), \quad (9)$$

$$a_k = \frac{-\ln(v_{k1}v_{k2})}{y_k(u_{k1}u_{k2})^{-1}}, \quad (10)$$

where

$$t_k = \left\lfloor \frac{\ln S_k}{-\ln(u_{k1}u_{k2})} + \beta_k \right\rfloor.$$

Now, the ICWS algorithm in [13] can be presented equivalently in Algorithm 1 for producing D Min-Hash codes.

Algorithm 1 is the standard implementation of ICWS. If we further slightly rearrange Eq. (10) as follows

$$a_k = \frac{-\ln(v_{k1}v_{k2})u_{k2}}{y_k u_{k1}^{-1}}, \quad (11)$$

we can find two interesting properties: (1) the denominator, $y_k u_{k1}^{-1}$, is essentially an unbiased estimator of the weight S_k; and (2) the numerator $-\ln(v_{k1}v_{k2})u_{k2}$ is actually a standard exponential distribution.

The first property can be easily verified based on the uniformity $y_k = u_{k1}S_k$, where $u_{k1} \sim \text{Uniform}(0,1)$, because the derivation of the ICWS algorithm [13] is based on $y_k = u_{k1}S_k$. However, in its algorithm, ICWS [13] samples y_k through Eq. (9) instead of $y_k = u_{k1}S_k$. Thus $y_k u_{k1}^{-1}$ is not exactly equivalent to S_k but its unbiased estimator:

$$\mathbb{E}(y_k u_{k1}^{-1}) = \mathbb{E}(\hat{S}_k) = S_k. \quad (12)$$

To see the second property, let $m_k = -\ln(v_{k1}v_{k2})u_{k2} = c_k u_{k2}$ then we have

$$\begin{aligned} \text{pdf}_{M_k}(m_k) &= \int_{0^+}^{1} \frac{1}{u_{k2}} \text{pdf}_{U_{k2}}(u_{k2}) \text{pdf}_{C_k}\left(\frac{m_k}{u_{k2}}\right) du_{k2} \\ &= \int_{0^+}^{1} \frac{1}{u_{k2}} \cdot 1 \cdot \frac{m_k}{u_{k2}} \mathrm{e}^{-\frac{m_k}{u_{k2}}} du_{k2} = \mathrm{e}^{-m_k} \end{aligned}$$

which implies that

$$m_k = -\ln(v_{k1}v_{k2})u_{k2} \sim \text{Exp}(1). \quad (13)$$

Substituting Eqs. (12) and (13) into Eq. (11) we obtain $a_k = \frac{m_k}{\hat{S}_k}$. Through the Jocobian transformation, we have $\text{pdf}_{A_k}(a_k) = \text{pdf}_{M_k}(m_k)|\frac{dm_k}{da_k}| = \hat{S}_k \mathrm{e}^{-\hat{S}_k a_k}$, which further implies $a_k \sim \text{Exp}(\hat{S}_k)$. Thus far, *we have found that the mystery of ICWS is to implicitly employ an exponential distribution parameterized with $\hat{S}_k$ to sample a_k*, that is

$$a_k = \frac{m_k}{\hat{S}_k} \sim \text{Exp}(\hat{S}_k). \quad (14)$$

In this case, the probability of the k-th element being sampled is equal to the probability of the k-th element being assigned with the minimum exponential random variable a_k, where the probability is in proportion to its weight: $\Pr(a_k = \min\{a_1, \cdots, a_n\}) = \frac{\hat{S}_k}{\sum_{k'} \hat{S}_{k'}}$.

3.2 The PCWS Algorithm

By transforming the original ICWS algorithm into an equivalent version in terms of five independent uniform random variables, we have revealed the mystery of ICWS Eq. (14). In addition to understanding the underlying mechanism of ICWS, one may ask what else we have been inspired to improve the ICWS algorithm. In the following, we will make use of the uncovered property Eq. (13) to reduce both time and space complexities of ICWS.

Algorithm 1 The ICWS Algorithm (equivalent to [13])

Input: $\mathcal{S} = \{S_1, \cdots, S_n\}$; number of samples D
Output: $\{(k_*^{(d)}, y_{k_*^{(d)}}^{(d)})\}_{d=1}^{D}$

1: **for** $k = 1, \ldots, n$ **do**
2: **for** $d = 1, \ldots, D$ **do**
3: $u_{k1}^{(d)}, u_{k2}^{(d)} \sim \text{Uniform}(0,1)$
4: $\beta_k^{(d)} \sim \text{Uniform}(0,1)$
5: $v_{k1}^{(d)}, v_{k2}^{(d)} \sim \text{Uniform}(0,1)$
6: **end for**
7: **end for**
8: **for** all k such that $S_k > 0$ **do**
9: **for** $d = 1, \ldots, D$ **do**
10: $t_k^{(d)} = \left\lfloor \frac{\ln S_k}{-\ln(u_{k1}^{(d)} u_{k2}^{(d)})} + \beta_k^{(d)} \right\rfloor$
11: $y_k^{(d)} = \exp(-\ln(u_{k1}^{(d)} u_{k2}^{(d)})(t_k^{(d)} - \beta_k^{(d)}))$
12: $a_k^{(d)} = \frac{-\ln(v_{k1}^{(d)} v_{k2}^{(d)})}{y_k^{(d)}(u_{k1}^{(d)} u_{k2}^{(d)})^{-1}}$
13: **end for**
14: **end for**
15: **for** $d = 1, \ldots, D$ **do**
16: $k_*^{(d)} = \arg\min_k a_k^{(d)}$
17: **end for**
18: **return** $\{(k_*^{(d)}, y_{k_*^{(d)}}^{(d)})\}_{d=1}^{D}$

Recall in Eq. (11), the numerator employs three independent uniform random variables to produce a sample in the form of $-\ln(v_1 v_2)u_2$, which is proved to be a standard exponential distribution, $m_k \sim \text{Exp}(1)$ in Eq. (13). Obviously, it is costly in terms of both time and space to produce $-\ln(v_1 v_2)u_2$. Due to the fact that $-\ln x_k \sim \text{Exp}(1)$, $x_k \sim \text{Uniform}(0,1)$, we can adopt $-\ln x_k$ insted of $-\ln(v_1 v_2)u_2$ to achieve the same goal.

To this end, we can simply replace $a_k = \frac{-\ln(v_{k1}v_{k2})}{y_k(u_{k1}u_{k2})^{-1}}$ (Line 12 in Algorithm 1) with $a_k = \frac{-\ln x_k}{y_k u_{k1}^{-1}}$ and obtain the proposed PCWS algorithm (see Algorithm 2). The only two differences between ICWS and PCWS lie in

- ICWS has to generate one more uniform random variable than PCWS for each element k (Line 5).
- ICWS has a relatively more complicated expression than PCWS to calculate a_k (Line 12).

Complexity: In programming implementation, ICWS requires sampling five global uniform random variables for each element k (i.e., $u_{k1}, u_{k2}, \beta_k, v_{k1}, v_{k2}$); while PCWS only requires sampling four (i.e., $u_{k1}, u_{k2}, \beta_k, x_k$). From Algorithm 1 and Algorithm 2 it is easy to see that the space complexity of PCWS is $\mathcal{O}(4nD)$ while the space complexity of ICWS is $\mathcal{O}(5nD)$, where n denotes the size of the universal set and D the number of samples (number of Min-Hashes). Although the uniform random variables can be sampled off-line, it is worth noting that these variables have to be cached in the memory during the hashing process. In text mining applications, the size of the universal set (number of features) can easily reach 10^7; if we adopt 10^3 samples, an additional memory footprint of 10^{10} floats have to be al-

Algorithm 2 The PCWS Algorithm

Input: $\mathcal{S} = \{S_1, \cdots, S_n\}$; number of samples D
Output: $\{(k_*^{(d)}, y_{k_*^{(d)}}^{(d)})\}_{d=1}^{D}$

for $k = 1, \ldots, n$ **do**
  for $d = 1, \ldots, D$ **do**
    $u_{k1}^{(d)}, u_{k2}^{(d)} \sim \text{Uniform}(0, 1)$
    $\beta_k^{(d)} \sim \text{Uniform}(0, 1)$
    $x_k^{(d)} \sim \text{Uniform}(0, 1)$ // different from Algorithm 1
  end for
end for
for all k such that $S_k > 0$ **do**
  for $d = 1, \ldots, D$ **do**
    $t_k^{(d)} = \left\lfloor \frac{\ln S_k}{-\ln(u_{k1}^{(d)} u_{k2}^{(d)})} + \beta_k^{(d)} \right\rfloor$
    $y_k^{(d)} = \exp(-\ln(u_{k1}^{(d)} u_{k2}^{(d)})(t_k^{(d)} - \beta_k^{(d)}))$
    $a_k^{(d)} = \frac{-\ln x_k^{(d)}}{y_k^{(d)} (u_{k1}^{(d)})^{-1}}$ // different from Algorithm 1
  end for
end for
for $d = 1, \ldots, D$ **do**
  $k_*^{(d)} = \arg\min_k a_k^{(d)}$
end for
return $\{(k_*^{(d)}, y_{k_*^{(d)}}^{(d)})\}_{d=1}^{D}$

located. Thus, using one less uniform random variable can save substantial amount of memory cost on large-scale data sets. The simpler expression of PCWS for calculating a_k also saves time complexity $\mathcal{O}(nD)$. On a large-scale data set, the aggregated time saving due to one less uniform random variable and simpler expression can be substantial (see empirical test in Section 4). Compared to ICWS, PCWS enjoys 20% lower memory footprint and saves $1/5 \sim 1/3$ empirical runtime.

3.3 Analysis

In this subsection, we will prove that our PCWS algorithm generates (y_k, a_k) which indeed satisfy uniformity and consistency of the CWS scheme [19] .

3.3.1 Uniformity

We drop the element index k for conciseness. Following ICWS [13], the random variable $\ln y = r\left(\left\lfloor \frac{\ln S}{r} + \beta \right\rfloor - \beta\right)$, where $r = -\ln(u_1 u_2) \sim \text{Gamma}(2, 1)$, $\beta \sim \text{Uniform}(0, 1)$, shares the same distribution as $\ln y = \ln S - rb$, where $b \sim \text{Uniform}(0, 1)$. In the both situations, $\ln y$ is uniformly sampled from $[\ln S - r, \ln S]$. Since our PCWS algorithm still employs two "active indices", y and z, as ICWS does, we have $r = \ln z - \ln y$ for the sake of proof of uniformity. The distribution $\text{pdf}(y, z, a)$, defined for $y \le S$, $z \ge S$ and $a > 0$, can be obtained by transforming the distributions of the random variables as

$$\text{pdf}(y, z, a) = \text{pdf}(r, b, x) \left| \det \frac{\partial(r, b, x)}{\partial(y, z, a)} \right|,$$

where $r = \ln z - \ln y$, $b = \frac{\ln S - \ln y}{\ln z - \ln y}$, and $x = \exp(-ayu_1^{-1})$. Recall that r, b, x are mutually independent and have the following probability density functions: $\text{pdf}(r) = re^{-r}$ and $\text{pdf}(b) = \text{pdf}(x) = 1$. By computing the Jacobian determinant, we obtain

$$\text{pdf}(y, z, a) = \frac{1}{z^2}(yu_1^{-1})\text{e}^{-(yu_1^{-1})a}.$$

Marginalizing out z in $\text{pdf}(y, z, a)$ gives

$$\text{pdf}(y, a) = \int_S^{+\infty} \text{pdf}(y, z, a)dz = \frac{1}{S}(yu_1^{-1})\text{e}^{-(yu_1^{-1})a}.$$

Actually $\text{pdf}(y, a)$ is still conditioned on u_1. We can do an expectation over the distribution of yu_1^{-1} and, according to Eq. (12), obtain $\mathbb{E}(yu_1^{-1}) = S$. Therefore, we have

$$\text{pdf}(y, a) = \frac{1}{S}(S\text{e}^{-Sa}) = \text{pdf}(y)\text{pdf}(a).$$

It is easy to see that y is uniformly distributed in $[0, S]$ and a complies with an exponential distribution parameterized with S, that is, $a \sim \text{Exp}(S)$; meanwhile, y and a are independent. For all the weights $\{S_1, \ldots, S_n\}$ in weighted set $\mathcal{S}$, there exist a set of exponential distributions parameterized with the corresponding weights. According to Eq. (1), a_{k_*} is the minimum hash value with a probability in proportion to S_{k_*}, $\Pr(a_{k_*} = \min_k a_k) = \frac{S_{k_*}}{\sum_k S_k}$. Therefore, (k_*, y_{k_*}) is uniformly sampled from $\bigcup_k (\{k\} \times [0, S_k])$.

3.3.2 Consistency

Following ICWS [13], we will demonstrate that, for two non-empty weighted sets $\mathcal{S}$ and $\mathcal{T}$, if $\forall k, T_k \le S_k$, a subelement (k_*, y_{k_*}) is sampled from $\mathcal{S}$ and satisfies $y_{k_*} \le T_{k_*}$, then (k_*, y_{k_*}) will be sampled from $\mathcal{T}$.

Considering an element k_*, $t_{k_*}^S = \left\lfloor \frac{\ln S_{k_*}}{-\ln(u_{k_*1} u_{k_*2})} + \beta_{k_*} \right\rfloor$, and thus $\frac{\ln S_{k_*}}{-\ln(u_{k_*1} u_{k_*2})} + \beta_{k_*} - 1 < t_{k_*}^S \le \frac{\ln S_{k_*}}{-\ln(u_{k_*1} u_{k_*2})} + \beta_{k_*}$. By hypothesis, $y_{k_*}^S = y_{k_*} \le T_{k_*} \le S_{k_*}$, thus $\frac{\ln T_{k_*}}{-\ln(u_{k_*1} u_{k_*2})} + \beta_{k_*} - 1 < t_{k_*}^S = \frac{\ln y_{k_*}}{-\ln(u_{k_*1} u_{k_*2})} + \beta_{k_*} \le \frac{\ln T_{k_*}}{-\ln(u_{k_*1} u_{k_*2})} + \beta_{k_*}$. Obviously,

$$t_{k_*}^S = \left\lfloor \frac{\ln T_{k_*}}{-\ln(u_{k_*1} u_{k_*2})} + \beta_{k_*} \right\rfloor = t_{k_*}^T,$$

which indicates $y_{k_*}^S = y_{k_*} = y_{k_*}^T$. Thus $y_{k_*}^S$ and $y_{k_*}^T$ will be sampled from the k_*-th elements of $\mathcal{S}$ and $\mathcal{T}$, respectively.

On the other hand, we note that, for any k, a_k is essentially a monotonically non-increasing function of S_k:

$$a_k = \frac{-\ln x_k}{u_{k1}^{-1} \exp\left(r_k \left(\left\lfloor \frac{\ln S_k}{r_k} + \beta_k \right\rfloor - \beta_k\right)\right)},$$

where $r_k = -\ln(u_{k1} u_{k2})$. Therefore, $\forall k, a_k^T \ge a_k^S$ due to $T_k \le S_k$, while $a_{k_*}^T = a_{k_*}^S = \min_k a_k^S$ because of $y_{k_*}^S = y_{k_*}^T$. As a result, $a_{k_*}^T \le a_k^S \le a_k^T$ and in turn $\arg\min_k a_k^T = \arg\min_k a_k^S = k_*$, which demonstrates that (k_*, y_{k_*}) is sampled from $\mathcal{S}$ and $\mathcal{T}$ simultaneously. Thus consistency holds.

In summary, our PCWS algorithm satisfies the two properties, uniformity and consistency, of the CWS scheme. Thus, PCWS not only has an equivalent (but more efficient) algorithm to ICWS but also holds the same theoretical properties as ICWS.

4. EXPERIMENTAL RESULTS

In this section, we report the performance of our PCWS algorithm and its competitors on five real-world text data sets. In Subsection 4.2, we investigate the effectiveness and efficiency of the compared methods for classification. In Subsection 4.3, we investigate the effectiveness and efficiency of the compared methods for information retrieval.

4.1 Experimental Preliminaries

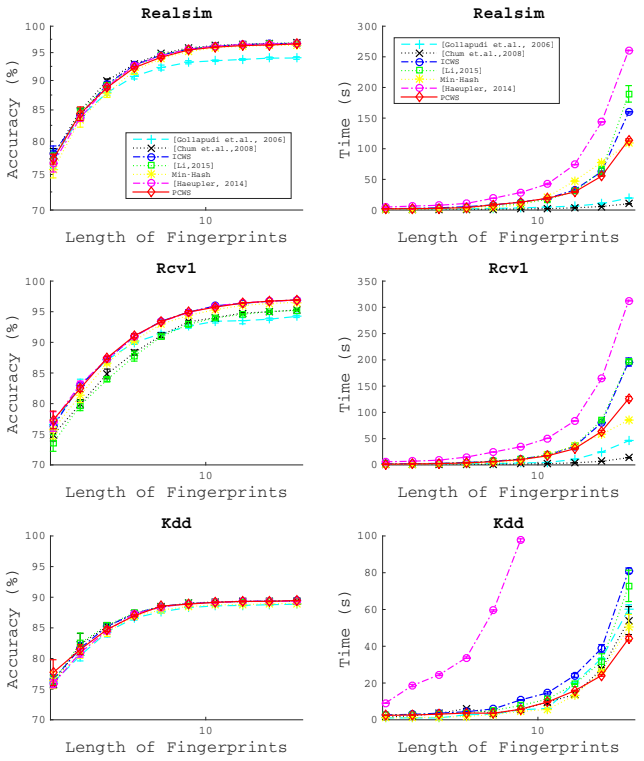

Figure 1: Classification results in accuracy (left column) and runtime (right column) of the compared methods on Real-sim, Rcv1 and Kdd. The x-axis denotes the length of fingerprints, D.

We compare our PCWS algorithm with six state-of-the-arts: (1) **Min-Hash**: The standard Min-Hash scheme is applied by simply treating weighted sets as binary sets; (2) **[Gollapudi et.al., 2006] [8]**: It transforms weighted sets into binary sets by thresholding real-value weights with random samples and then applies the standard Min-Hash scheme (another algorithm is introduced in the same paper which is however extremely inefficient for real-value weights and thus not reported). (3) **[Chum et.al., 2008] [5]**: It approximates the generalized Jaccard similarity with a bias for real-value weighted sets despite that the derivation is based on integer weighted sets; (4) **ICWS [13]**: It is introduced in Section 3.1, which is currently the state-of-the-art for weighted Min-Hash in terms of both effectiveness and efficiency; (5) **[Li, 2015] [15]**: It approximates ICWS by simply discarding one of the two components, that is y_k in Eq. (7), of ICWS; (6) **[Haeupler et.al., 2014] [10]**: It approximates the generalized Jaccard similarity by rounding the real-value weights with probability and then quantizing the integer weights.

All the compared algorithms are implemented in Matlab. For [Haeupler et.al., 2014], each weight is scaled up by a factor of 100. We first apply all the algorithms to generate the fingerprints of the data. Suppose that each algorithm generates $\mathbf{x}_S$ and $\mathbf{x}_T$, which are the fingerprints with the length of D for the two real-value weighted sets, $\mathcal{S}$ and $\mathcal{T}$, respectively, the similarity between $\mathcal{S}$ and $\mathcal{T}$ is $\mathbf{Sim}_{\mathcal{S},\mathcal{T}} = \sum_{d=1}^{D} \frac{\mathbf{1}(x_{S,d}=x_{T,d})}{D}$, where $\mathbf{1}(state) = 1$ if $state$ is true, and

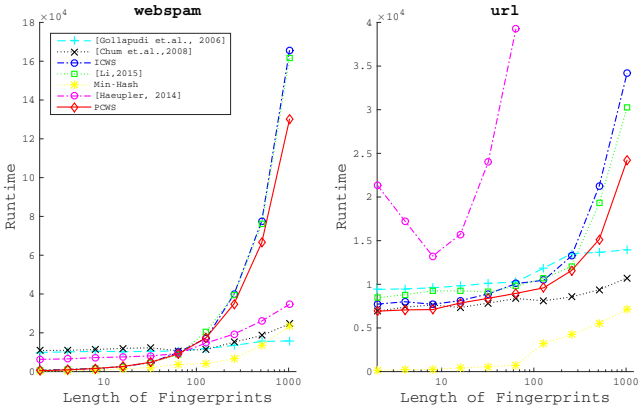

Figure 2: Retrieval runtime of the compared methods on Webspam and Url. The x-axis denotes the length of fingerprints, D.

$\mathbf{1}(state) = 0$ otherwise. The above equation calculates the ratio of the same Min-Hash values (i.e., collision) between $\mathbf{x}_S$ and $\mathbf{x}_T$, which is used to approximate the probability that $\mathcal{S}$ and $\mathcal{T}$ generate the same Min-Hash value. We set D, the parameter of the number of hash functions (or random samples), such that $D \in \{2, 4, 8, 16, 32, 64, 128, 256, 512, 1024\}$. All the random variables are globally generated at random. That is, in one sampling process, the same elements in different sets use the same set of random variables. All the experiments are conducted on a node of a Linux Cluster with 8×3.1 GHz Intel Xeon CPU (64 bit) and 1TB RAM.

4.2 Results on Classification

We investigate classification performance of the compared methods using LIBSVM [2] with 10-fold cross-validation on three binary classification benchmarks[2]:

1. **Real-sim**: The data set is a collection of UseNet articles from four discussion groups about simulated auto racing, simulated aviation, real autos and real aviation, respectively. The formatted document data set, with 72,309 samples and 20,958 features, has been arranged into two classes: real and simulated. Since the real class and the simulated class of the original data are unbalanced, data preprocessing is performed by randomly selecting 10,000 real samples and 10,000 simulated samples as positive and negative instances, respectively.

2. **Rcv1**: The data set is a large collection of newswire stories drawn from online databases. The formatted data set has 20,242 training samples with 47,236 features. The data set has been categorized into two classes: positive instances contain CCAT and ECAT while negative ones contain GCAT and MACT on the website. Similarly, we randomly select 10,000 positive instances and 10,000 negative ones to compose a balanced data set.

3. **Kdd**: This is a large educational data set from the KDD Cup 2010 competition. The formatted data set

[2]Real-sim, Rcv1, Kdd and Webspam can be downloaded at https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html

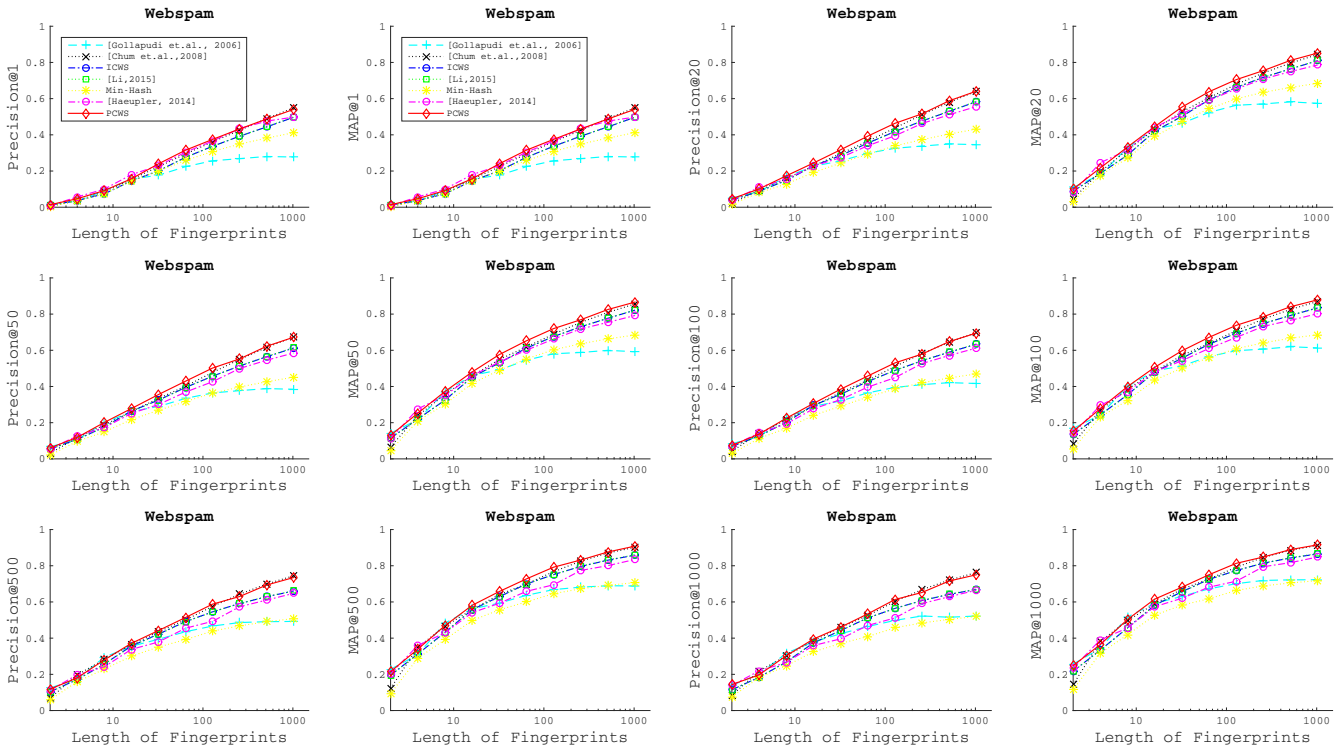

Figure 3: Retrieval results in Precision@K (odd columns) and MAP@K (even columns) of the compared methods on Webspam. The x-axis denotes the length of fingerprints, D.

has 8,407,752 training samples with 20,216,830 features. We also randomly select 10,000 positive instances and 10,000 negative ones to form a balanced data set for classification.

We repeat each experiment 5 times and compute the mean and the standard derivation of results.

Discussions on Real-sim: The subplots in the first row in Figure 1 show the comparison results on Real-sim. One can see that our PCWS algorithm achieves almost the same accuracy as ICWS, [Li, 2015] and [Chum et.al., 2008]. The comparison between PCWS and ICWS demonstrates that our PCWS algorithm indeed satisfies the CWS scheme. The reason that [Chum et.al., 2008] also performs similarly with [Li, 2015] may be the one discussed in [15], that is, one component of ICWS, y_k, is trivial to approximate the generalized Jaccard similarity for most data sets. In terms of runtime, our PCWS algorithm performs similarly with ICWS and [Li, 2015] when D ranges from 2 to 128, and subsequently, the gap between PCWS and ICWS clearly widens. Particularly, PCWS takes around 3/4 of ICWS runtime and nearly 3/5 of [Li, 2015] runtime, when $D = 1024$.

Discussions on Rcv1: The subplots in the second row of Figure 1 show the comparison results on Rcv1. Our PCWS algorithm preserves almost the same accuracy as ICWS. Furthermore, the two CWS algorithms clearly perform better than the standard Min-Hash scheme and other algorithms which biasedly estimate the generalized Jaccard Similarity. In terms of runtime, our PCWS algorithm maintains the same level as ICWS and [Li, 2015] with D varying from 2 to 64. Our PCWS algorithm is remarkably superior to ICWS and [Li, 2015] when D varies from 128 to 1024. Again, the runtime of PCWS is reduced by a factor of 1/3 compared to ICWS and [Li, 2015] when $D = 1024$.

Discussions on Kdd: In order to evaluate the classification ability on data with a large number of features (i.e., size of the universal set), we test the compared methods on the Kdd data set. The comparison results are reported in the subplots in the third row of Figure 1 (in this experiment, each algorithm is given a cutoff time of 100 seconds). Our PCWS algorithm still preserves the same accuracy as ICWS. In terms of runtime, PCWS runs much more efficiently than ICWS and [Li, 2015] this time on the data set with a large number of features. The performance gain of our PCWS algorithm in terms of runtime starts in the very beginning; as the length of fingerprint D increases, the gap becomes more significant: 1/3 faster than ICWS and [Li, 2015] when D approaches to 1024.

4.3 Results on Top-K Retrieval

In this experiment, we carry out top-K retrieval, for $K = \{1, 20, 50, 100, 500, 1000\}$. We adopt Precision@$K$ and Mean Average Precision (MAP)@K to measure the performance in terms of accuracy because precision is relatively more important than recall in large-scale retrieval; furthermore, MAP contains information of relative orders of the retrieved samples, which can reflect the retrieval quality more accurately. To this end, we select two large-scale public data sets:

1. **Webspam**: It is a web text data set provided by a large-scale learning challenge. The data set has 350,000 instances and 16,609,143 features. We randomly select 1,000 samples from the original data set as query examples and the rest as the database.

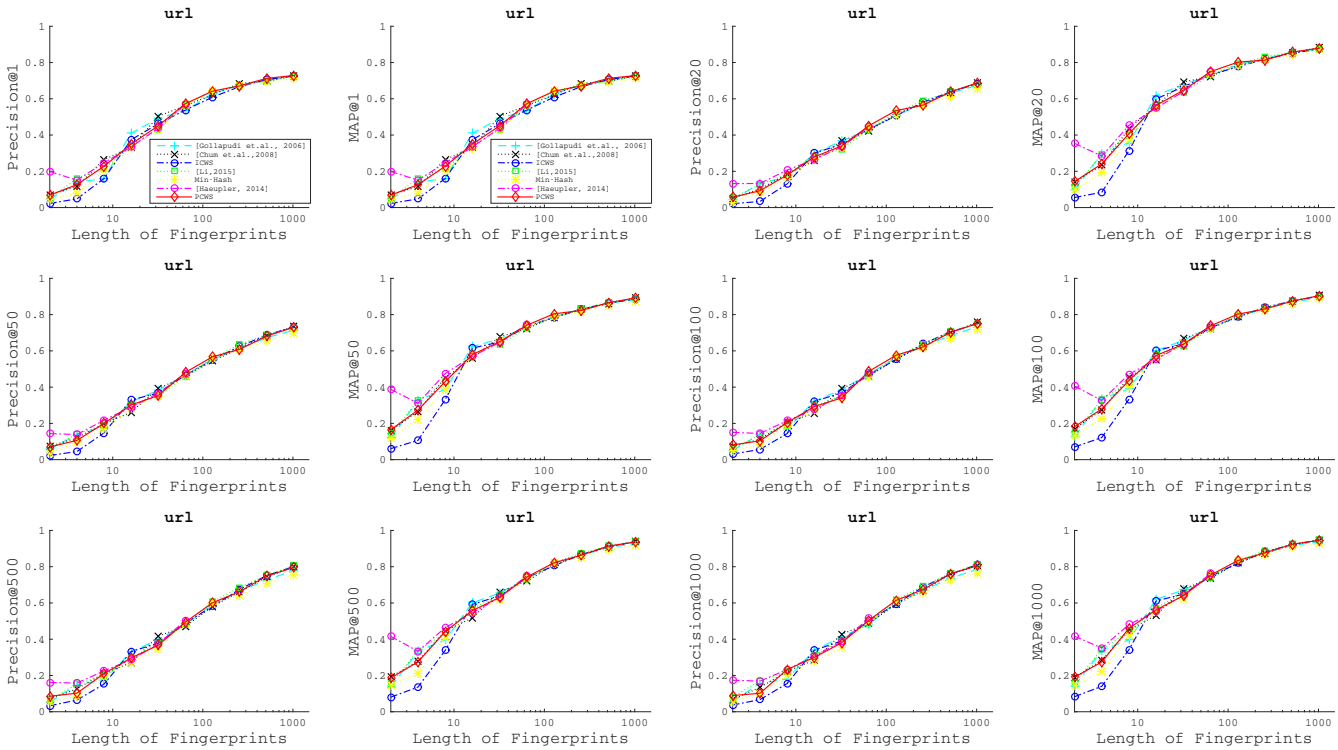

Figure 4: Retrieval results in Precision@K (odd columns) and MAP@K (even columns) of the compared methods on Url. The x-axis denotes the length of fingerprints, D.

2. **Url** [18]: The data set contains 2,396,130 URLs and 3,231,961 features. We randomly select 1,000 samples from the original data set as query examples and the rest as the database.

Discussions on Webspam: Figure 3 reports the comparison results on the Webspam data set which has more than 16 million features. We observe that our PCWS algorithm generally outperforms all the competitors under all configurations. Again, as the length of fingerprints D increases, the performance gain of PCWS compared to the other algorithms becomes clearer. PCWS outperforms ICWS and [Li, 2015] by about 5% when $D = 1024$; this improvement over its counterparts might be due to the positive effect of using one less random variable. PCWS performs much better than Min-Hash and [Gollapudi et.al., 2006] and it is superior to the two algorithms by around 25% and 40%, respectively, when $D = 1024$. PCWS also shows better performance than [Chum et.al., 2008] and [Haeupler et.al., 2014] in most cases. In the left subplot of Figure 2 for comparison of runtime, PCWS clearly runs faster than ICWS and [Li, 2015] as the length of fingerprints D increases. For example, PCWS performs approximately 30% faster than the two algorithms when $D = 1024$.

Discussions on Url: Figure 4 reports the comparison results on the Url data set which has more than 2 million instances and over 3 million features (in this experiment, each algorithm is given a cutoff time of 40,000 seconds). Generally, our PCWS algorithm performs similarly with other algorithms, and even does better than ICWS with D ranging from 2 to 8. In the right subplot of Figure 2 for comparison of runtime, PCWS clearly outperforms ICWS and [Li, 2015]; in particular, PCWS runs nearly 30% faster than ICWS and around 1.25 times as fast as [Li, 2015] when $D = 1024$.

4.4 Discussion on Space Efficiency

Finally, we analyze the advantage of our PCWS algorithm on space efficiency. Recall that our PCWS algorithm is able to save $\mathcal{O}(nD)$ of memory footprint (see Section 3.2), where n is the number of features and D is the length of fingerprints, because PCWS generates one less uniform random variable than ICWS does. This advantage in space complexity may not be remarkable when the size of the universal set n is small; however, when n is large in most cases of real-world data sets, the saved memory footprint can be substantial. For example, Kdd, Webspam, and Url have 20.2 million, 16.6 million and 3.2 million features, respectively, and our PCWS algorithm can enjoy 150 GB, 130 GB and 24 GB lower memory footprints in the case of $D = 1024$, compared to ICWS without accuracy degradation. If the number of features becomes even larger or more samples of Min-Hashes are used, the advantage of PCWS on space efficiency will be more remarkable. Obviously, our PCWS algorithm significantly improves ICWS in terms of both time and space efficiency, which indicates that PCWS is more practical.

5. CONCLUSION AND FUTURE WORK

In this paper, we propose Practical Consistent Weighted Sampling (PCWS) to further improve the efficiency of Improved CWS [13], which is considered as the state-of-the-art method for real-value weighted Min-Hash. The proposed PCWS algorithm is mathematically equivalent to ICWS and

preserves the same theoretical properties as ICWS, but with reduced theoretical complexity in both time and space. We conduct extensive empirical tests of our PCWS algorithm and a number of state-of-the-art methods on five real-world text data sets for classification and information retrieval. The experimental results show that PCWS is able to achieve the same (even better) performance than ICWS with $1/5 \sim 1/3$ reduced empirical runtime and 20% reduced memory footprint. In the cases of large number of features, PCWS can save hundreds of GB of memory footprint, which makes it more practical in dealing with real-world data sets in the era of big data.

Existing similarity-preserving hashing techniques can only deal with nested binary sets [14] and tree-structured categorical data [4]. It will be interesting to extend CWS schemes to hash nested weighted sets, which not only encode the importance of feature but also preserve the *multi-level exchangeability* [4] of feature, in our future work.

Acknowledgment

This work is partially supported by ARC Discovery Grant DP140100545.

6. REFERENCES

[1] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise Independent Permutations. In *STOC*, pages 327–336, 1998.

[2] C.-C. Chang and C.-J. Lin. LIBSVM: A Library for Support Vector Machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):1–27, 2011.

[3] M. S. Charikar. Similarity Estimation Techniques from Rounding Algorithms. In *STOC*, pages 380–388, 2002.

[4] L. Chi, B. Li, and X. Zhu. Context-preserving Hashing for Fast Text Classification. In *SDM*, pages 100–108, 2014.

[5] O. Chum, J. Philbin, A. Zisserman, et al. Near Duplicate Image Detection: Min-Hash and Tf-idf Weighting. In *BMVC*, pages 1–10, 2008.

[6] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive Hashing Scheme Based on p-stable Sistributions. In *SOCG*, pages 253–262, 2004.

[7] E. Dumbill. A Revolution That Will Transform How We Live, Work, and Think: An Interview with the Authors of Big Data. *Big Data*, 1(2):73–77, 2013.

[8] S. Gollapudi and R. Panigrahy. Exploiting Asymmetry in Hierarchical Topic Extraction. In *CIKM*, pages 475–482, 2006.

[9] S. Gunelius. *The Data Explosion in 2014 Minute by Minute Infographic*, Jul 2014. http://aci.info/2014/07/12/the-data-explosion-in-2014-minute-by-minute-infographic.

[10] B. Haeupler, M. Manasse, and K. Talwar. Consistent Weighted Sampling Made Fast, Small, and Easy. *arXiv preprint arXiv:1410.4266*, 2014.

[11] T. H. Haveliwala, A. Gionis, and P. Indyk. Scalable Techniques for Clustering the Web. In *WebDB*, pages 129–134, 2000.

[12] P. Indyk and R. Motwani. Approximate Nearest Neighbors: towards Removing the curse of Dimensionality. In *STOC*, pages 604–613, 1998.

[13] S. Ioffe. Improved Consistent Sampling, Weighted Minhash and L1 Sketching. In *ICDM*, pages 246–255, 2010.

[14] B. Li, X. Zhu, L. Chi, and C. Zhang. Nested Subtree Hash Kernels for Large-scale Graph Classification over Streams. In *ICDM*, pages 399–408, 2012.

[15] P. Li. 0-Bit Consistent Weighted Sampling. In *KDD*, pages 665–674, 2015.

[16] P. Li and C. König. *b*-Bit Minwise Hashing. In *WWW*, pages 671–680, 2010.

[17] P. Li, A. Owen, and C.-H. Zhang. One Permutation Hashing. In *NIPS*, pages 3113–3121, 2012.

[18] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Identifying Suspicious URLs: an Application of Large-scale Online Learning. In *ICML*, pages 681–688, 2009.

[19] M. Manasse, F. McSherry, and K. Talwar. Consistent Weighted Sampling. *Unpublished technical report*, 2010.

[20] G. S. Manku, A. Jain, and A. Das Sarma. Detecting Near-duplicates for Web Crawling. In *WWW*, pages 141–150, 2007.

[21] M. Mitzenmacher, R. Pagh, and N. Pham. Efficient Estimation for High Similarities Using Odd Sketches. In *WWW*, pages 109–118, 2014.

[22] A. Rajaraman, J. D. Ullman, J. D. Ullman, and J. D. Ullman. *Mining of Massive Datasets*, volume 1. Cambridge University Press Cambridge, 2012.

[23] E. Schonfeld. *Google Processing 20,000 Terabytes A Day, And Growing*, Jan 2008. http://techcrunch.com/2008/01/09/google-processing-20000-terabytes-a-day-and-growing.

[24] A. Shrivastava. Exact Weighted Minwise Hashing in Constant Time. *arXiv preprint arXiv:1602.08393*, 2016.

[25] A. Shrivastava and P. Li. Densifying One Permutation Hashing via Rotation for Fast Near Neighbor Search. In *ICML*, pages 557–565, 2014.

[26] A. Shrivastava and P. Li. In Defense of Minhash Over SimHash. In *AISTATS*, pages 886–894, 2014.

[27] D. Sullivan. *Google Still Doing At Least 1 Trillion Searches Per Year*, Jan 2015. http://searchengineland.com/google-1-trillion-searches-per-year-212940.

[28] D. Tam. *Facebook processes more than 500 TB of data daily*, Jul 2014. http://www.cnet.com/news/facebook-processes-more-than-500-tb-of-data-daily.

[29] W. Wu, B. Li, L. Chen, and C. Zhang. Canonical Consistent Weighted Sampling for Real-Value Weighted Min-Hash. In *ICDM*, pages 1287–1292, 2016.

[30] D. Yang, B. Li, and P. Cudré-Mauroux. POIsketch: Semantic Place Labeling over User Activity Streams. In *IJCAI*, pages 2697–2703, 2016.