

Query Independent Weighted Minwise Hashing

Dinesh Maharjan

Department of Computer Science and Information Technology

Patan Multiple Campus

Patan Dhoka, Nepal

mzn.dnes@gmail.com

Abstract—Weighted Minwise Hashing (WMH) efficiently estimates Generalized Jaccard Similarity between weighted sets using rejection sampling, but requires query-dependent upper bounds that prevent offline preprocessing of training data. This limitation makes WMH impractical for large-scale data mining and streaming applications. We propose Query Independent Weighted Minwise Hashing (QIWMH), which eliminates this dependency by randomly normalizing element weights to $[0,1]$ using only training data, enabling sketches of the training set to be generated fully offline before any query arrives. We formally prove that QIWMH is an approximate estimator of Generalized Jaccard Similarity (GJS). Experimental evaluation on seven real-world datasets across two tasks — 1-NN classification and Top-K retrieval — demonstrates that QIWMH achieves comparable or superior accuracy to state-of-the-art methods including WMH, ICWS and PCWS, while achieving speedups of 2.3x to 14x in generating sketches.

Keywords—Locality Sensitive Hashing, Weighted MinHash, Consistent Weighted Sampling, Generalized Jaccard Similarity, Nearest Neighbor Search.

I. INTRODUCTION

Similarity search between two sets is the basic and most important operation in the applications like near-duplicate image detection [1], content-based audio-video retrieval [2], outlier detection [3], Top- K Retrieval [4], Recommendation Systems [5] etc. Different types of similarity functions like Cosine similarity [6], Jaccard similarity [7], [8], l_p distance [9] can be used in such data mining tasks. This type of exact computation becomes expensive when sets are high dimensional. It is worth noting that the volume and features of the datasets are increasing exponentially. The online news [10] reports that the hundreds of sensors in one autonomous car can generate and consume 40 TB of data per day. The sparsity of datasets is another challenge of the similarity search techniques. In such cases, the exact computation of similarity between two sets becomes the major bottleneck. In contrast, the approximate estimation is efficient and effective.

Locality Sensitive Hashing (LSH) [11]–[14] approximately estimates the similarity between two sets by hashing the similar sets to the same hash code with high probability. LSH uses family of hash function H . The probability of collision of hash values of two sets X and Y generated with randomly selected hash function h from H is equal to the similarity between sets. Sketch is the vector of hash codes extracted by LSH for a given set. MinHash [15], [16] is the LSH method of Jaccard similarity for the quick detection of similar text documents. A. Shrivastava used MinHash-generated sketches

to estimate the Cosine similarity [17]. The major drawback of MinHash is that it cannot deal with the weighted sets.

Weighted set is very common in real life problems where weight represents the importance of corresponding element. A weighted set S consists of weights s_i for the elements i . Given a universal set, $\Omega = \{0, 1, \dots, p-1\}$, Generalized Jaccard Similarity (GJS), J , between two weighted sets $S = \{s_0, s_1, s_2, \dots, s_{p-1}\}$ and $T = \{t_0, t_1, t_2, \dots, t_{p-1}\}$ is defined in Eq. (1). Here, $i \in \Omega$. If $s_i > 0$, then i is considered as the non-zero element of S . If $s_i < 0$ then, i is not present in S and we use $s_i = 0$ while calculating GJS . GJS was introduced in [18] and was efficiently estimated by introducing the method of unweighted minwise hashing.

$$J(S, T) = \frac{\sum_{i=0}^{p-1} \min(s_i, t_i)}{\sum_{i=0}^{p-1} \max(s_i, t_i)} \quad (1)$$

Their method [18] converts a weighted set into an unweighted one by expanding the s_i with additional distinct elements $\{s_1, s_2, \dots, s_{s_i}\}$ in proportion to its weight s_i . Then Standard MinHash [16] is applied. This approach cannot deal with real weights and is inefficient. Other unweighted minwise hashing methods [19], [20] can deal with the real weights also but they still have to generate additional elements. The significant improvement was seen in [21] that uses pairs of active indices (w_i, z_i) to generate each hash value. Their method is referred to as Consistent Weighted Sampling (CWS). However, this approach still needs to sample many active indices per weight to obtain each hash value.

Improved CWS (ICWS) [22] is able to directly generate the pair of active indices (w_i, z_i) for each weight. The coupled hash value (i, w_i) is generated with the careful transformation of the active indices. Although ICWS is able to directly generate a pair of active indices (w_i, z_i) for each weight, it is very slow as it needs to generate random numbers three times the size of the universal set. It has to perform logarithmic and exponential transformations while generating hash values. ICWS needs $O(dp)$ time to generate the sketch of size d for each weighted set where p is the size of the universal set, Ω . For large p and d , the ICWS method becomes impractical. Moreover, the coupled hash value, (i, w_i) , does not reduce the dimension of set significantly.

In order to overcome the time complexity of ICWS, Weighted Minwise Hashing (WMH) [23] was suggested that

efficiently estimates GJS using Rejection Sampling [24]. The execution time of WMH does not depend on the size of the universal set but on the number of non-zeros. While WMH is very efficient for a less sparse weighted set, it becomes very slow for a highly sparse dataset. This is because Rejection Sampling rejects many unwanted samples that exist in the highly sparse set leading to high computational cost. Another crucial issue with WMH is that it needs to calculate upper bound for each weight of the elements in the universal set which cannot be done unless the query is known. Because of this limit, WMH cannot be applied to the recent work [25] on malware detection. Moreover, it cannot be applied to the stream datasets as unseen data point might contain the weights that exceed the upper bound [26]. Thus, WMH is unable to generate hash values for the training set offline.

In this work, we introduce the improved variant of WMH method that avoids the calculation of upper bounds. We call our method as Query Independent WMH (QIWMH). We, firstly, sample the non-zero element, i , of a weighted set, S , using discrete uniform random number generator and then randomly normalize its weight s_i to $[0, 1)$. Unlike WMH, QIWMH does not need queries to normalize the weights. This allows to generate hash values for the training dataset offline which will save the substantial amount of execution time. We summarize our contributions as the following.

- Unlike WMH, QIWMH does not calculate upper bounds from the entire dataset, instead randomly and consistently normalizes weight s_i of the sampled element to $[0, 1)$.
- QIWMH does not need to map all the integers between two consecutive cumulative upper bounds to the corresponding element.
- QIWMH can obtain hash values for the training datasets offline. Thus, QIWMH can be very useful for time crucial applications.
- Since QIWMH randomly normalizes weight s_i to $[0, 1)$, it does not need any information about future data point.

II. BACKGROUND

The Weighted Minwise Hashing (WMH) [23] is our motivating paper so we briefly review WMH in Section II-A followed by the its limitation in Section II-B.

A. Weighted Minwise Hashing (WMH)

Given the universal set, $\Omega = \{0, 1, \dots, p-1\}$, WMH [23] efficiently estimates GJS between two weighted sets $S = \{s_0, s_1, s_2, \dots, s_{p-1}\}$ and $T = \{t_0, t_1, t_2, \dots, t_{p-1}\}$ by generating the sketch of each set using Rejection Sampling. For each element $i \in \Omega$, WMH determines upper bound b_i using a large dataset including the query. The upper bound can be maximum weight among the weights of corresponding component in the universal set and is always assumed to be an integer. WMH defines $r_0 = 0$, $r_{i+1} = \sum_{k=0}^i b_k$, and $R = b_0 + b_1 + \dots + b_{p-1}$. Now, each weight, s_i in S , falls in the interval, $[r_i, r_{i+1})$. The interval $[r_i, s_i)$ is considered as the white map. Similarly, the interval $[s_i, r_{i+1})$ is considered as the black map. WMH tries

to sample the white map by generating Continuous Uniform Random variable x in $[0, R)$. If the random number, x , lies in the white map, then number of attempts made to sample the white map is returned as the hash value. If the random variable, x , lies in the black map, then it again tries to sample the white map by generating another Uniform random number. This process goes on till the white map is sampled. For ensuring whether x lies in black or white map, WMH uses a function, $map(y)$, that maps y to i . Here y is all the possible integers in $[r_i, r_{i+1})$. WMH satisfies the LSH property i.e.

$$P(h(S) = h(T)) = J(S, T)$$

, where $h(S)$ and $h(T)$ are the hash values generated by executing the WMH algorithm for weighted set S and T . WMH proposes unbiased estimator $\hat{J}(S, T)$ defined in Eq. (2)

$$\hat{J}(S, T) = \frac{\sum_{j=0}^{d-1} \mathbb{1}(h_j(S) = h_j(T))}{d} \quad (2)$$

WMH is summarized in Algorithm 1.

B. Limitation of WMH

WMH has two serious problems. First one is the time complexity for a highly sparse dataset. The time complexity of WMH depends upon the number of unwanted samples i.e. the total size of the black map. The size of the black map increases with the decrease in the white map and with the increase in the sparsity of the weighted set. Remember that $[r_i, r_{i+1})$ is black map if $s_i = 0$. Since WMH uses a continuous uniform random number to sample black or white map, there are an infinite number of unwanted samples in each black map $[s_i, r_{i+1})$. This increases the number of rejections of the unwanted samples and increases the execution time. We reduce an infinite number of unwanted samples into a finite number by using a discrete uniform random number in the definite discrete sample space $[0, p)$ where p is the size of the universal set Ω .

The major drawback of WMH is that it needs to calculate upper bounds for each weight of the elements in universal set. This limits WMH from generating hash values for training sets before it knows the query. We explain this issue with the help of an example shown in Fig. 1. Upper bounds $\{5, 5, 2, 4, 6\}$ are considered for two weighted sets $S = \{1.6, 4.4, -9.9, 3.9, 3.2\}$ and $T = \{4.77, 1, 1.1, -1.3, 5.09\}$ of size 5. Since the future query is not known in advance, these upper bounds cannot be calculated offline. It means reduced sized sketch of S is generated only after all the queries are known. Even though we can calculate upper bounds for the training dataset offline, we cannot generate hash values. This is not desirable for large-scale data-mining tasks like classification. We solve this issue by randomly normalizing s_i to $[0, 1)$. Paper [23] also discusses the normalization of weights but it still needs the query. We randomly and consistently normalize the weights without query.

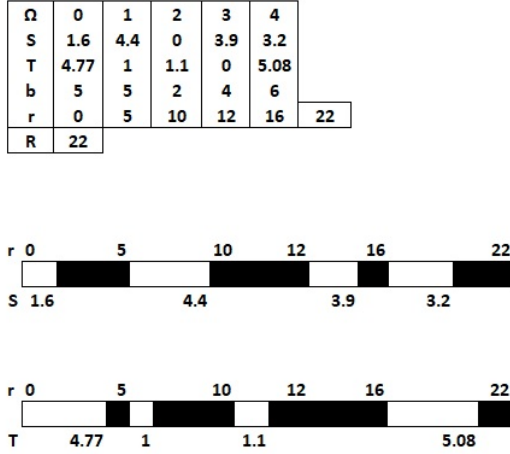


Fig. 1. Illustration of WMH execution with input set S and T of size 4. b and r , respectively, are upper bound and commutative upper bound of the corresponding elements in the universal set.

Algorithm 1 WMH

Input: weighted set, $S = \{s_0, s_1, s_2, \dots, s_{p-1}\}$, size of sketch, d , random seeds, $seed$, upper bounds, $B = \{b_0, b_1, \dots, b_{p-1}\}$, $r_0 = 0$, $r_{i+1} = \sum_{k=0}^i b_k$

Output: sketch of S , h

```

1:  $R = r_{p+1}$ 
2: for  $j = 0, 1, \dots, d-1$  do
3:    $attempt = 0$ 
4:    $set\ seed_j$ 
5:   while true do
6:      $x \sim ContinuousUniform(0, R)$ 
7:      $y = \lfloor x \rfloor$ 
8:      $i = map(y)$ 
9:     if  $x \leq r_i + s_i$  then
10:       $break$ 
11:   end if
12:    $attempt = attempt + 1$ 
13: end while
14:  $h_j = attempt$ 
15: end for
16: return  $h$ 
```

III. QIWMH: QUERY INDEPENDENT WEIGHTED MINWISE HASHING

We propose Query Independent Weighted Minwise Hashing (QIWMH) method as the improved version of WMH that does not need to calculate upper bounds for each weight. QIWMH uses a discrete random variable, i , in $[0, p)$ to sample the non-zero element, i , of the weighted set $S = \{s_0, s_1, s_2, \dots, s_{p-1}\}$. Unlike WMH, QIWMH enjoys the finite number of unwanted samples, $p - D$, where D is the number of non-zeros in S . After sampling non-zero element i , we randomly normalize s_i to $[0, 1)$ using the Eq. (3).

$$s'_i = \frac{s_i}{s_i + u} \quad (3)$$

, where $u \sim Uniform(0, 1)$. The beauty of WMH is that it carefully creates the interval $[r_i, r_{i+1})$ for each weight s_i . We also consistently normalize each sampled weight s_i to $[0, 1)$. So, our white map is $[0, w_i)$ and black map is $[w_i, 1)$. Now, we try to sample the white map by generating a continuous uniform random number x in $[0, 1)$. If $x \leq w_i$, we return the number of attempts made to sample the white map as the hash value otherwise we reject and restart the sampling process again. This simple approach helps to avoid the calculation of upper bounds for each weight of the elements in the universal set Ω . Our black map $[w_i, 1)$ does not increase the number of unwanted samples because we sample the non-zero element i from finite discrete distribution in $[0, p)$ before sampling the black or white map. Since we do not calculate upper bounds from the entire dataset, we can generate sketches of the training set offline. This will save the time required for generating sketches of training set. Thus, a query can be responded quickly.

The same seed is used to generate a corresponding hash value of each weighted set. Details about QIWMH is given in Algorithm 2. The sketch of hash values for S is denoted as $h_j(S)$. We also estimate J using Eq. (2) as our method preserves LSH property.

Algorithm 2 QIWMH

Input: weighted set $S = \{s_0, s_1, s_2, \dots, s_{p-1}\}$, size of sketch, d , random seeds, $seed$

Output: sketch of S , h

```

1: for  $j = 0, 1, \dots, d-1$  do
2:    $attempt = 0$ 
3:    $set\ seed_j$ 
4:   while true do
5:      $i \sim DiscreteUniform(0, p)$ 
6:     if  $s_i > 0$  then
7:        $u \sim ContinuousUniform(0, 1)$ 
8:        $w = \frac{s_i}{s_i + u}$ 
9:        $x \sim ContinuousUniform(0, 1)$ 
10:      if  $x \leq w$  then
11:         $break$ 
12:      end if
13:    end if
14:     $attempt = attempt + 1$ 
15:  end while
16:   $h_j = attempt$ 
17: end for
18: return  $h$ 
```

Theorem 1: Let us consider two weighted sets $S = \{s_0, s_1, \dots, s_{p-1}\}$, $T = \{t_0, t_1, \dots, t_{p-1}\}$, and a universal set Ω with $i \in \Omega$. Define $g : [0, \infty) \rightarrow [0, \infty)$ by $g(w) =$

$w \ln(1 + \frac{1}{w})$ for $w > 0$ and $g(0) = 0$. Then

$$P(h(S) = h(T)) = \frac{\sum_{i=0}^{p-1} g(\min(s_i, t_i))}{\sum_{i=0}^{p-1} g(\max(s_i, t_i))}$$

Proof: Let S_{white} and T_{white} be the element wise vector of white intervals in S and T respectively, computed for a fixed, realized set of normalization variables $u_0, \dots, u_{p-1} \sim \text{Uniform}(0, 1)$ (one per dimension, shared by S and T since, as noted above, the same seed generates the corresponding hash value of each weighted set). $|S_{white}| = \sum_{i=0}^{p-1} s'_i$ is the size of white map in S , with $s'_i = \frac{s_i}{s_i + u_i}$. Let $x \sim \text{ContinuousUniform}(0, 1)$. Let us consider three possible events at the time QIWMH returns hash value.

$h(S) = h(T)$ if and only if $x \in S_{white} \cap T_{white}$

$h(S) > h(T)$ if and only if $x \in S_{white} - T_{white}$

$h(S) < h(T)$ if and only if $x \in T_{white} - S_{white}$

Therefore, conditional on the realized $u_0, \dots, u_{p-1}$,

$$\begin{aligned} P(h(S) = h(T) \mid u_0, \dots, u_{p-1}) &= \frac{|S_{white} \cap T_{white}|}{|(S_{white} \cap T_{white}) \cup (S_{white} - T_{white}) \cup (T_{white} - S_{white})|} \\ &= \frac{|S_{white} \cap T_{white}|}{|S_{white} \cup T_{white}|} \end{aligned}$$

Notice that $S_{white} \cap T_{white}$ and $S_{white} \cup T_{white}$ are the element-wise intersection and union between the white intervals in S and T respectively. Since $f(w) = \frac{w}{w+u}$ is strictly increasing in w for fixed $u > 0$, whichever of s_i, t_i is smaller also yields the smaller normalized weight at dimension i , so $\min(s'_i, t'_i) = f(\min(s_i, t_i))$ and $\max(s'_i, t'_i) = f(\max(s_i, t_i))$ exactly. Thus,

$$\begin{aligned} |S_{white} \cap T_{white}| &= \sum_{i=0}^{p-1} \min(s'_i, t'_i). \\ |S_{white} \cup T_{white}| &= \sum_{i=0}^{p-1} \max(s'_i, t'_i). \end{aligned}$$

So, conditional on $u_0, \dots, u_{p-1}$,

$$P(h(S) = h(T) \mid u_0, \dots, u_{p-1}) = \frac{\sum_{i=0}^{p-1} \min(s'_i, t'_i)}{\sum_{i=0}^{p-1} \max(s'_i, t'_i)}$$

The unconditional probability is obtained by averaging over the randomness of $u_0, \dots, u_{p-1}$ at the level of the underlying per-dimension acceptance events, not by naively substituting $s'_i \approx s_i$ into the ratio above (which does not hold in general, since $s'_i = \frac{s_i}{s_i + u_i} \rightarrow 1$, not s_i , as s_i grows). At a single dimension i , S accepts with probability s'_i given u_i , so unconditionally $P(S \text{ accepts at } i) = \mathbb{E}_{u_i} \left[\frac{s_i}{s_i + u_i} \right] = \int_0^1 \frac{s_i}{s_i + u} du = s_i \ln \left(1 + \frac{1}{s_i} \right) = g(s_i)$, and

similarly, using $\min(s'_i, t'_i) = f(\min(s_i, t_i))$ shown above, $P(\text{both accept at } i) = \mathbb{E}_{u_i}[\min(s'_i, t'_i)] = g(\min(s_i, t_i))$. Aggregating these per-dimension probabilities over $i \in \Omega$ (each dimension is sampled with probability $1/p$, and the events are disjoint across i within a single trial) and combining via the same white/black map race used above, but now with $g(s_i)$ and $g(t_i)$ in place of s'_i, t'_i , gives

$$P(h(S) = h(T)) = \frac{\sum_{i=0}^{p-1} g(\min(s_i, t_i))}{\sum_{i=0}^{p-1} g(\max(s_i, t_i))}$$

where we used $g(s_i) + g(t_i) = g(\min(s_i, t_i)) + g(\max(s_i, t_i))$, which holds identically since $\{s_i, t_i\} = \{\min(s_i, t_i), \max(s_i, t_i)\}$ as a set. This proves the theorem.

The equation above has the same $\min/\max$ structure as $J(S, T)$ in Eq. (1), but with each raw weight w replaced by $g(w) = w \ln(1 + \frac{1}{w})$. The two coincide exactly only when g is proportional to the identity, which is not the case in general; the following corollary makes precise when the approximation is good.

Corollary 1: Let w_{min} and w_{max} be the smallest and largest positive weights appearing in S and T combined, and let $c_{lo} = \ln \left(1 + \frac{1}{w_{max}} \right)$, $c_{hi} = \ln \left(1 + \frac{1}{w_{min}} \right)$, $\kappa = c_{hi}/c_{lo} \geq 1$. Then

$$\frac{1}{\kappa} J(S, T) \leq P(h(S) = h(T)) \leq \kappa J(S, T)$$

so $P(h(S) = h(T)) \rightarrow J(S, T)$, and correspondingly $\hat{J}(S, T) \rightarrow J(S, T)$ by Eq. (2), as $\kappa \rightarrow 1$, i.e., as the dynamic range of the nonzero weights in the data shrinks.

Proof: Since $w \mapsto g(w)/w = \ln(1 + 1/w)$ is strictly decreasing, every nonzero weight $w \in [w_{min}, w_{max}]$ appearing in the sums satisfies $c_{lo} \leq g(w)/w \leq c_{hi}$, hence $c_{lo} \cdot w \leq g(w) \leq c_{hi} \cdot w$. Applying this termwise to both the numerator and denominator of the collision-probability formula above and combining the resulting bounds gives the claim.

This gives a precise, checkable version of “QIWMH approximately estimates GJS ”: the approximation is controlled by κ , the ratio between the largest and smallest nonzero weights in the data. Datasets with small, tightly-clustered weights (e.g. Webspam, average weight 0.22 in Table I) should show QIWMH tracking $J(S, T)$ more closely than datasets with weights closer to 1 or spread over a wider range (e.g. StarLightCurves, average weight 0.91).

Running Time Analysis: Let hash value of S generated by QIWMH be $h_i(S)$. $h_i(S)$ is the number of trials including the first success. So, it has geometric distribution with success

probability $\frac{\sum_{i=0}^{p-1} s'_i}{p}$. Therefore, the expected value of $h_i(S)$ is $\frac{p}{\sum_{i=0}^{p-1} s'_i}$. In case of WMH, the success probability is $\frac{\sum_{i=0}^{p-1} s_i}{R}$ and

the expected value of $h_i(S)$ at which while loop stops is $\frac{R}{\sum_{i=0}^{p-1} s_i}$.

We have $s'_i < s_i$. If $s_i \geq 1$, then $R > p$. In such case, $\frac{p}{\sum_{i=0}^{p-1} s'_i} < \frac{R}{\sum_{i=0}^{p-1} s_i}$ i.e. the expected value that WMH returns hash value in $h_i(S)$ trials will be greater than that of QIWMH. On the other hand, if $s_i < 1$, then $p = R$. Thus, $\frac{p}{\sum_{i=0}^{p-1} s'_i} > \frac{R}{\sum_{i=0}^{p-1} s_i}$ i.e.

the expected value that QIWMH returns hash value in $h_i(S)$ trials will be greater than that of WMH. However, QIWMH can generate hash value for training set offline. Thus, QIWMH can be faster in real-time.

IV. EXPERIMENTAL EVALUATION

Classification and Top- K retrieval tasks are important applications of sketching methods. Thus, we compare QIWMH with four state-of-the-art methods, WMH [23], ICWS [22], CCWS [27] and PCWS [28] using these two important tasks on seven real datasets. We discuss the results from 1-NN classification experiments in Section IV-A. We demonstrate that QIWMH is able to extract the similar Top- K data samples to a given query efficiently in Section IV-B. We performed both tasks with $d \in \{32, 64, 128, 512\}$.

We store d seeds to generate sketches of size d . The basic statistics on the datasets used for our experiments is shown in Table I. For the smooth execution of other methods, which may be very slow, we selected sample data points from some large datasets as shown in Table I. The sample dataset contains a balanced number of positive and negative classes. All the algorithms were implemented in Python3.14.4 in a laptop with Intel i7 2.40GHz CPU and 4GB RAM.

TABLE I
BASIC STATISTICS OF DATASETS

Dataset	Size of Train Set	Size of Test Set	p	Sparsity	Average Weight
Gisette	6000	1000	5000	0.87	0.68
FordA	3601	1320	500	0.5	0.8
HandOutlines	1000	370	2709	0.41	0.66
CinCECGtorso	1380	40	1639	0.5	0.51
Leukemia	38	34	7129	0.53	0.8
StarLightCurves	8236	1000	1024	0.55	0.91
Webspam	2500	1000	309277	0.66	0.22

A. 1-NN Classification Task

We demonstrate the effectiveness of QIWMH in building the 1-NN classifier using six real datasets, Gisette, FordA, HandOutlines, CinCECGtorso, Leukemia and StarLightCurves. Gisette and Leukemia are downloaded from [29] and others from [30]. We compare the performance of selected methods on the basis of classification accuracy and sketch generating time. Classification Accuracy is the proportion of correctly classified weighted sets in test data. Results are shown in Fig. 2.

a) **Discussion on the Results:** From Fig. 2, we can see that all four methods have almost the same classification accuracy on most of datasets. The classification accuracy of QIWMH is lower than other methods when $d=32$ on CinCECGtorso and Gisette but it becomes same when $d=512$. The gap between accuracies of QIWMH and other methods decreases with an increase in the number of hashes. Thus, the performance of QIWMH is comparable to other state-of-the-art methods. QIWMH performs slightly worse on StarLightCurves datasets. QIWMH outperforms all other methods in case of HandOutlines dataset.

The plots in even rows of Fig. 2 show the total hash generation time for train and test set. QIWMH clearly outperforms other methods on CinCECGtorso dataset. QIWMH is 2 to 14 times faster than WMH. The improvement is not surprising as QIWMH can generate hash values offline.

B. Top- K Retrieval Task

In Top- K Retrieval task, we extract the most similar K points from training set to the query in test set. Mean Precision is commonly used as the quality measure for Top- K retrieval applications. Mean Average Precision (MAP) is another important quality measure that depends upon the sequence of retrieved points. Thus, we compare the performance of selected methods by plotting Mean Precision@ K and MAP@ K against the number of hash values. We use four datasets, FordA, HandOutlines, Gisette, and Webspam for this task. The first two datasets are downloaded from [30] and others from [29]. The task was performed with $K \in \{50, 200, 500\}$. The results for this task is shown in Figs. 3 and 4.

a) **Discussion on Results:** Mean Precision and MAP plots in Figs. 3 and 4 demonstrate that QIWMH performs better than other methods on FordA dataset. The improvement in performance increases with an increase in the number of hash values. QIWMH performs slightly worse on Webspam dataset. All methods have almost similar performance on Gisette dataset. It is also to be noted that the performance of QIWMH improves with larger values of K .

V. RELATED WORK

The exact computation [9], [31] of different similarity functions like Jaccard similarity, Cosine similarity, l_p distance, Dynamic Time Warping (DTW) between two sets is very effective. Especially, DTW method [32] can compute the distance between two unequal sets by relatively aligning the elements. It means DTW does not need to convert a set into a vector. Thus, it is widely used in many applications like classification and Top- K retrieval. Although the exact computation of distance is very effective, it is not scalable to high dimensional datasets as computations are very expensive. Remember that the internet-connected devices like smartphones, tablets, surveillance cameras, smart TVs etc are producing the tsunami of data [33] every day. This is the major challenge for the similarity search methods.

Unlike exact methods, Locality Sensitive Hashing (LSH) approximately estimates the similarity between two sets. It was

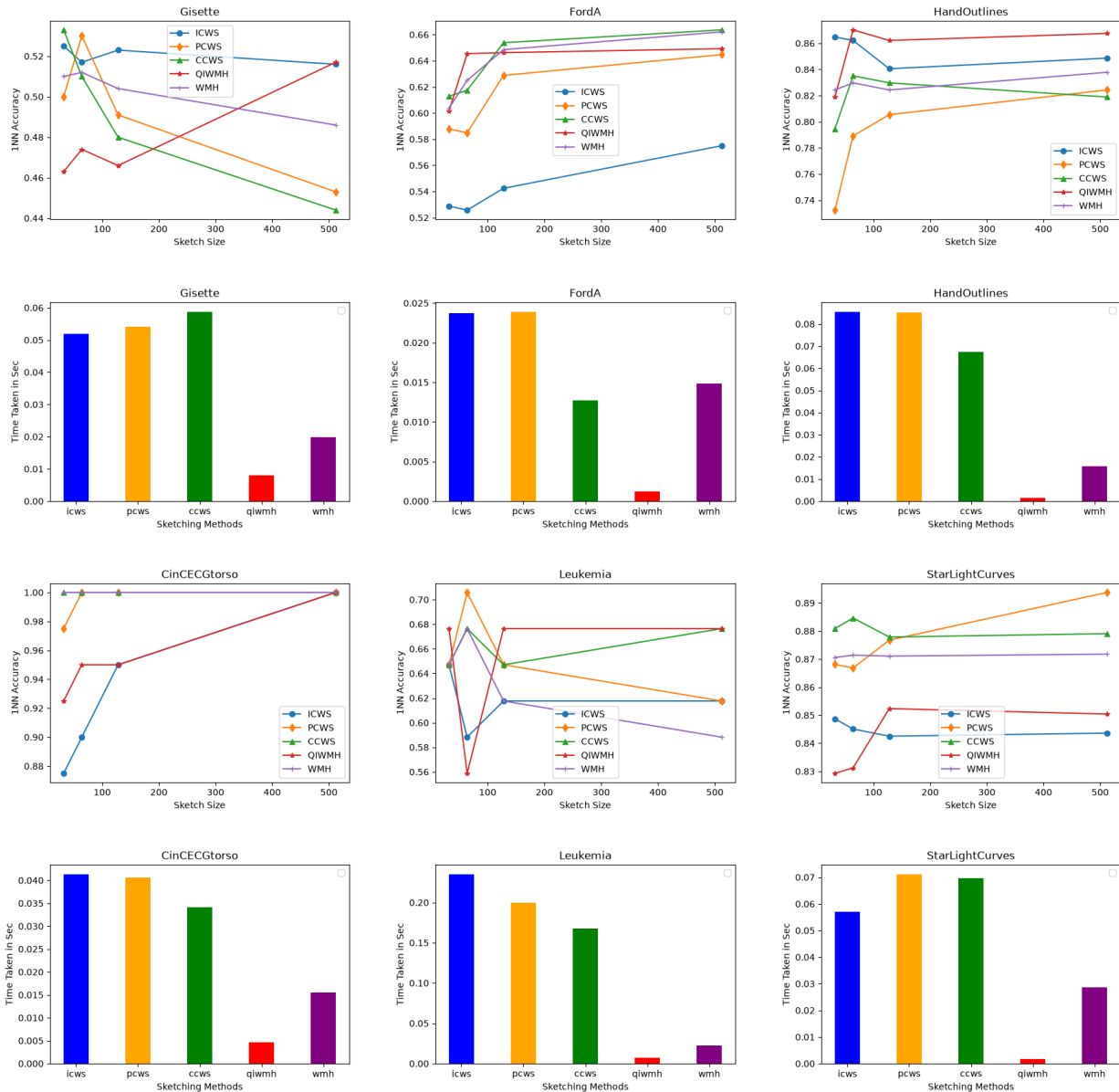


Fig. 2. The plot shows the average classification accuracy and total time (sec) with size of the sketch for six real datasets based on 5 executions. The total time includes hash generation time (for train and test set) generalized Jaccard similarity estimation time.

introduced by Motawani et al. [34] for efficiently searching the most similar data points to the query. It generates consistent and small-sized sketches of each data point by employing the family of hash functions. Thus, it is heavily used in commercial search engines [14], [35], [36], in multimedia [37], [38], in machine learning [39], [40] etc. MinHash [15], [16] for Jaccard similarity, random projection based LSH for l_2 norm [11], SimHash [41] for angular distance are some important methods of LSH.

MinHash [15], [16] is the most popular LSH technique for estimating Jaccard similarity. MinHash generates the sketches for data points using the random hash functions based on random permutations. Generating large permutation for high di-

mensional datasets is computationally expensive. Thus, many research work has been done to improve the efficiency of MinHash. One Permutation Hashing [42] generates multiple hash codes with single permutation. It partitions the long universal set into many equal-sized bins and generates one hash value in each bin. This method is efficient, but some of the bins may not have any hash value. Such empty bin was filtered out in papers [43]–[45] with the method of densification. It assigns the hash value of an empty bin with that of randomly sampled non-empty bin. Another method, Conditional Random Sampling [46] considers some number of consecutive non-zero elements to generate multiple hash codes with single permutation. Since consecutive non-zero elements

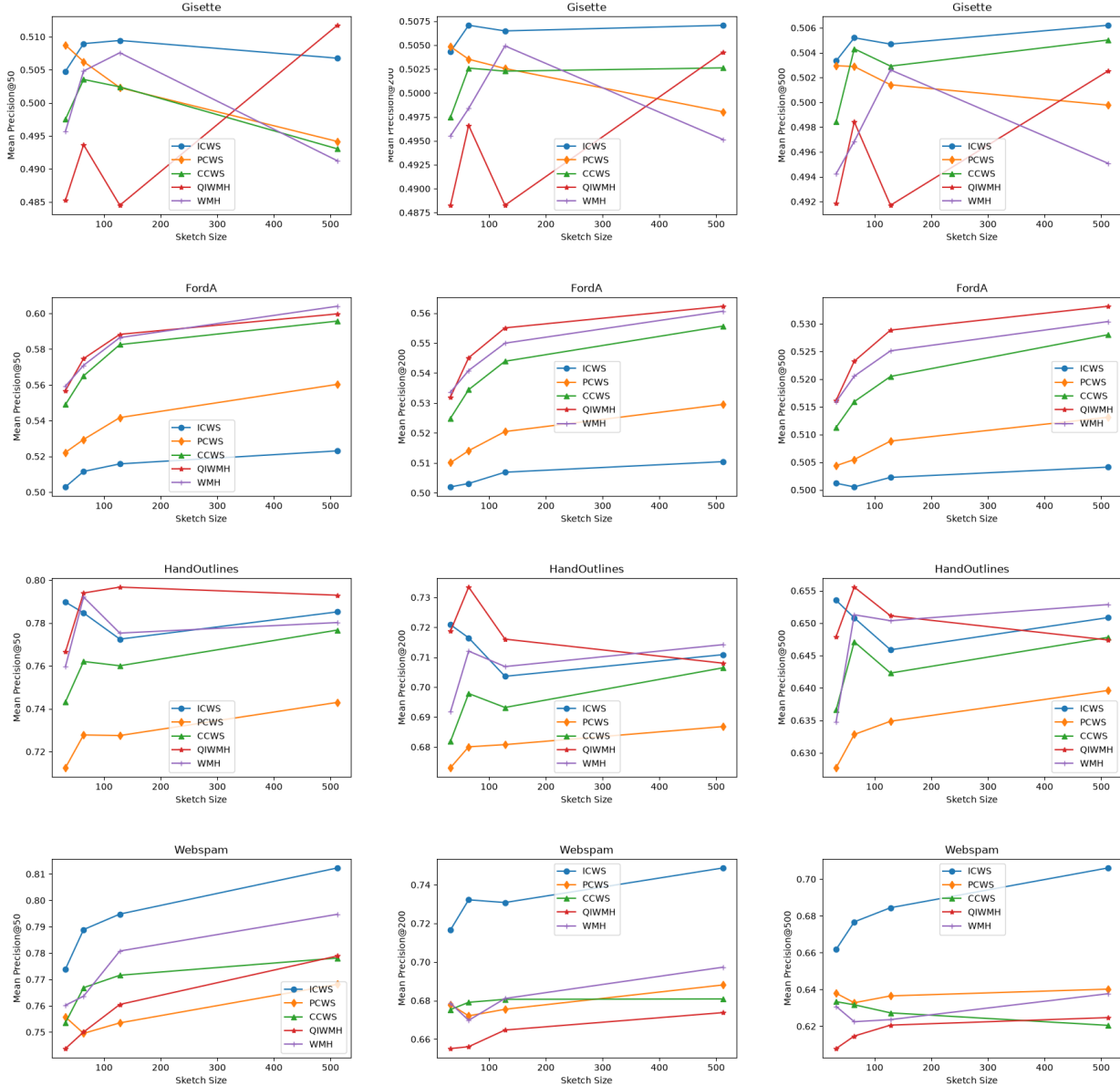


Fig. 3. Top- K results showing the precision@ K of each methods with $K \in \{50, 200, 500\}$. The x-axis denotes size of sketch, d .

can't be the same across data points, this approach does not satisfy the LSH property.

The major limitation of these variants of Minhash is that they all are designed for approximating the Jaccard similarity between binary sets. MinHash is unable to consider the importance of elements in a weighted set, S . In this regard, paper [18] defined Generalized Jaccard similarity and empirically showed that the similarity between weighted sets is equivalent to Generalized Jaccard similarity (GJS). For this, they add new distinct elements, $s_1, s_2, \dots, s_{s_i}$, in proportion to corresponding weight s_i . Then, they generate hash values by applying the Standard MinHash. To extend this approach for real weights, [20] divides each weight into equal sized sub

weights called quantized weight. Float part after quantization is saved with some probability. These approaches increase the size of the universal set and add up more computational cost while generating hash values.

To reduce the computational cost, the techniques in [19] normalizes bag of elements with the use of upper bound and generates the normalized set. Then they store only those elements that satisfy the given threshold. Even though it does not increase the size of the universal set, it has to perform normalization of each weighted set. In order to generate hash values efficiently, M. Mannase et al. [21] proposed the pair of active indices (w_{ik}, z_{ik}) that bracket the weight s_i . Although this method can be extended to real weight, it still needs to

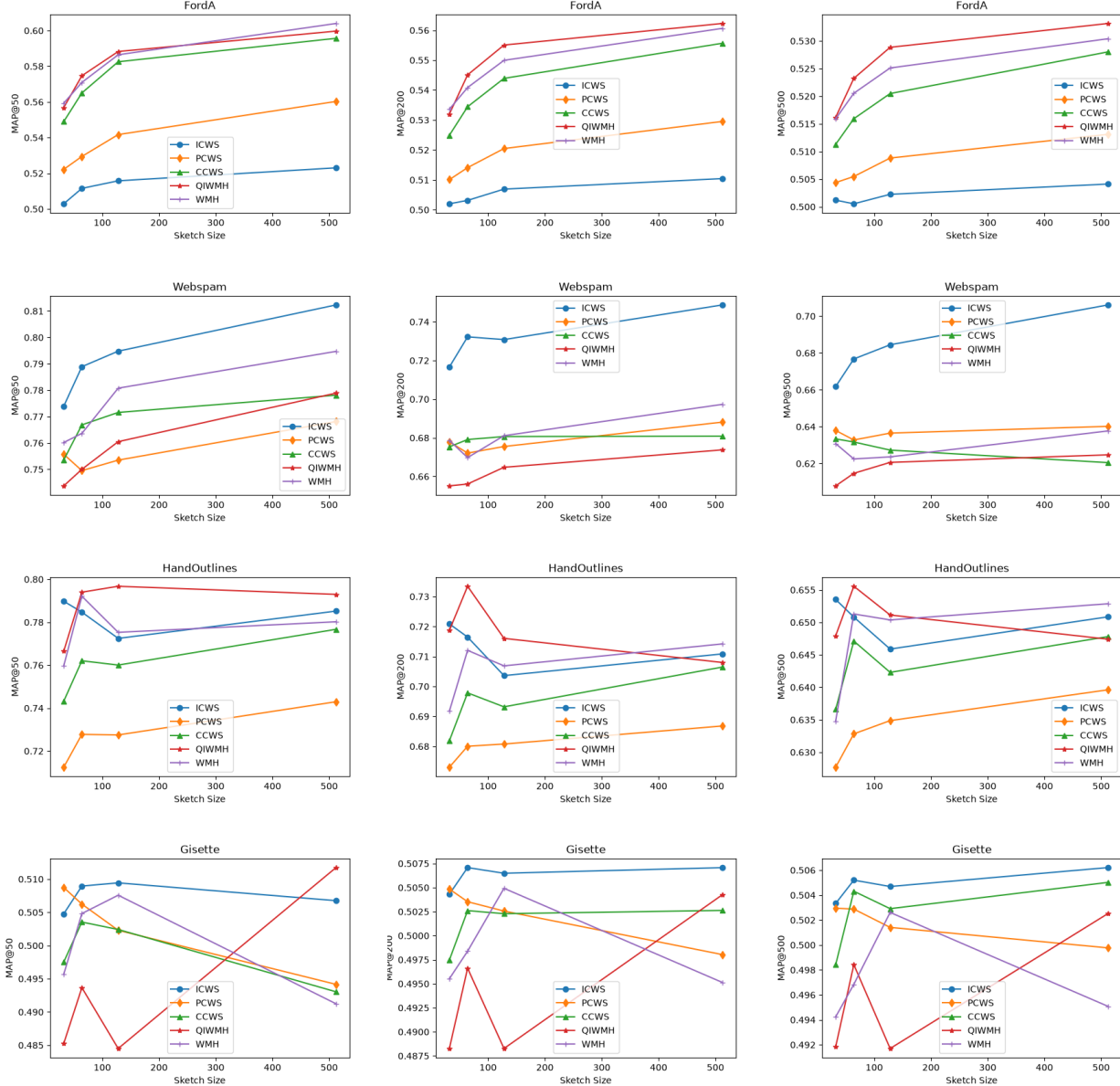


Fig. 4. Plot shows the MAP@K of each methods with $K \in \{50, 200, 500\}$. The x-axis denotes size of sketch, d.

generate many active indices for each non-zero element i . To improve its efficiency further, S. Ioffe [22] directly generated the pair of active indices (w_i, z_i) from a uniform distribution in worst case constant time. w_i is the greatest active index such that $w_i < s_i$ and z_i is the least active index such that $s_i > z_i$. Some variants of ICWS have been published. PCWS [28] uses 4 different random variables from the Uniform Distribution to generate active indices. CCWS [27] estimates GJS by uniformly discretizing the weights but it is less consistent as the small change in weight may change the active index w .

VI. CONCLUSION

Our proposed method, QIWMH, efficiently generates the sketches of weighted sets using the rejection sampling in the

discrete sample space $[0, p)$. QIWMH firstly samples the non-zero element i of weighted set S and then consistently normalizes the weight s_i to $[0, 1)$. Since we randomly normalize the weights of the sampled element, we do not need the query before actual execution starts. This creates the white map in $[0, w)$ and the black map in $[w, 1)$. Finally, QIWMH returns the number of attempts made to sample the white map as the hash value. Unlike WMH, QIWMH does not need to generate upper bounds of each weight. Thus QIWMH can generate sketches of the training set offline.

We evaluated the performance of QIWMH through comparison experiments using two tasks, the 1-NN classification and the Top-K retrieval, on seven real datasets. Experiments

tal results demonstrate that QIWMH generates high-quality sketches of weighted sets for estimating generalized Jaccard similarity. QIWMH mimics the general characteristics of WMH. Since QIWMH is much faster than other methods, the performance of QIWMH can be increased by increasing the number of hash values. Therefore, our approach can be used for the analysis of large-scale sparse weighted sets efficiently.

REFERENCES

- [1] O. Chum, J. Philbin, M. Isard, and A. Zisserman, "Scalable near identical image and shot detection," in *Proceedings of the 6th ACM international conference on Image and video retrieval*. ACM, 2007, pp. 549–556.
- [2] S. Hu, "Efficient video retrieval by locality sensitive hashing," in *Proceedings (ICASSP'05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005.*, vol. 2. IEEE, 2005, pp. ii–449.
- [3] Y. Wang, S. Parthasarathy, and S. Tatikonda, "Locality sensitive outlier detection: A ranking driven approach," in *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*. IEEE, 2011, pp. 410–421.
- [4] F. Zou, X. Tang, K. Li, Y. Wang, J. Song, S. Yang, and H. Ling, "Hidden semantic hashing for fast retrieval over large scale document collection," *Multimedia Tools and Applications*, vol. 77, no. 3, pp. 3677–3697, 2018.
- [5] C. Shahabi, F. Banaei-Kashani, Y.-S. Chen, and D. McLeod, "Yoda: An accurate and scalable web-based recommendation system," in *International Conference on Cooperative Information Systems*. Springer, 2001, pp. 418–432.
- [6] S. Zhu, J. Wu, H. Xiong, and G. Xia, "Scaling up top-k cosine similarity search," *Data & Knowledge Engineering*, vol. 70, no. 1, pp. 60–83, 2011.
- [7] I. E. Vorontsov, I. V. Kulakovskiy, and V. J. Makeev, "Jaccard index based similarity measure to compare transcription factor binding site models," *Algorithms for Molecular Biology*, vol. 8, no. 1, p. 23, 2013.
- [8] S. Niwattanakul, J. Singthongchai, E. Naenudorn, and S. Wanapu, "Using of jaccard coefficient for keywords similarity," in *Proceedings of the International MultiConference of Engineers and Computer Scientists*, vol. 1, 2013.
- [9] L. Wang, Y. Zhang, and J. Feng, "On the euclidean distance of images," *IEEE transactions on pattern analysis and machine intelligence*, vol. 27, no. 8, pp. 1334–1339, 2005.
- [10] P. Nelson. (2016) Just one autonomous car will use 4,000 gb of data per day. [Online]. Available: <https://www.networkworld.com/article/3147892/internet/one-autonomous-car-will-use-4000-gb-of-dataday.html>
- [11] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," in *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*. IEEE, 2006, pp. 459–468.
- [12] A. Andoni, P. Indyk, T. Laarhoven, I. Razenshteyn, and L. Schmidt, "Practical and optimal lsh for angular distance," in *Advances in Neural Information Processing Systems*, 2015, pp. 1225–1233.
- [13] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proceedings of the twentieth annual symposium on Computational geometry*. ACM, 2004, pp. 253–262.
- [14] B. Kulis and K. Grauman, "Kernelized locality-sensitive hashing for scalable image search," in *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, 2009, pp. 2130–2137.
- [15] A. Z. Broder, "On the resemblance and containment of documents," in *Compression and Complexity of Sequences 1997. Proceedings*. IEEE, 1997, pp. 21–29.
- [16] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher, "Min-wise independent permutations," *Journal of Computer and System Sciences*, vol. 60, no. 3, pp. 630–659, 2000.
- [17] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*. ACM, 2002, pp. 380–388.
- [18] T. HAVELIWALA, "Scalable techniques for clustering the web," in *Proc. of the Third International Workshop on the Web and Databases (Informal Proceedings)*, 2000, 2000.
- [19] S. Gollapudi and R. Panigrahy, "Exploiting asymmetry in hierarchical topic extraction," in *Proceedings of the 15th ACM international conference on Information and knowledge management*. ACM, 2006, pp. 475–482.
- [20] B. Haeupler, M. Manasse, and K. Talwar, "Consistent weighted sampling made fast, small, and easy," *arXiv preprint arXiv:1410.4266*, 2014.
- [21] M. Manasse, F. McSherry, and K. Talwar, "Consistent weighted sampling," *Unpublished technical report* <http://research.microsoft.com/en-us/people/manasse>, vol. 2, 2010.
- [22] S. Ioffe, "Improved consistent sampling, weighted minhash and 11 sketching," in *Data Mining (ICDM), 2010 IEEE 10th International Conference on*. IEEE, 2010, pp. 246–255.
- [23] A. Shrivastava, "Simple and efficient weighted minwise hashing," in *Advances in Neural Information Processing Systems*, 2016, pp. 1498–1506.
- [24] G. Casella, C. P. Robert, M. T. Wells *et al.*, "Generalized accept-reject sampling schemes," in *A Festschrift for Herman Rubin*. Institute of Mathematical Statistics, 2004, pp. 342–347.
- [25] E. Raff and C. Nicholas, "Malware classification and class imbalance via stochastic hashed lszd," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. ACM, 2017, pp. 111–120.
- [26] W. Wu, B. Li, L. Chen, C. Zhang, and P. Yu, "Improved consistent weighted sampling revisited," *IEEE Transactions on Knowledge and Data Engineering*, 2018.
- [27] W. Wu, B. Li, L. Chen, and C. Zhang, "Canonical consistent weighted sampling for real-value weighted min-hash," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 2016, pp. 1287–1292.
- [28] —, "Consistent weighted sampling made more practical," in *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2017, pp. 1035–1043.
- [29] C.-W. Hsu, C.-C. Chang, C.-J. Lin *et al.*, "A practical guide to support vector classification," Tech. Rep., 2003. [Online]. Available: <https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/>
- [30] Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen, and G. Batista, "The ucr time series classification archive," July 2015, www.cs.ucr.edu/~eamonn/time_series_data/.
- [31] Z. Xu and M. Xia, "Distance and similarity measures for hesitant fuzzy sets," *Information Sciences*, vol. 181, no. 11, pp. 2128–2138, 2011.
- [32] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh, "Searching and mining trillions of time series subsequences under dynamic time warping," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2012, pp. 262–270.
- [33] C. H. News. (2017) Tsunami of data could consume one fifth of global electricity by 2025. [Online]. Available: <http://www.climatechangenews.com/2017/12/11/tsunami-data-consume-one-fifth-global-electricity-2025/>
- [34] S. Har-Peled, P. Indyk, and R. Motwani, "Approximate nearest neighbor: Towards removing the curse of dimensionality," *Theory of computing*, vol. 8, no. 1, pp. 321–350, 2012.
- [35] M. Henzinger, "Finding near-duplicate web pages: a large-scale evaluation of algorithms," in *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2006, pp. 284–291.
- [36] A. Shrivastava and P. Li, "Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips)," in *Advances in Neural Information Processing Systems*, 2014, pp. 2321–2329.
- [37] Y. Lu, W. He, and A. Nabatchian, "Random draw forest: A salient index for similarity search over multimedia data," in *2018 IEEE Fourth International Conference on Multimedia Big Data (BigMM)*. IEEE, 2018, pp. 1–5.
- [38] W. Jiang and M. Gokhale, "Real-time classification of multimedia traffic using fpga," in *Field Programmable Logic and Applications (FPL), 2010 International Conference on*. IEEE, 2010, pp. 56–63.
- [39] J. Wang, S. Kumar, and S. CHANG, "Semi-supervised hashing for large scale search," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 6, no. 1, p. 1, 2010.
- [40] Z. Kang, W. T. Ooi, and Q. Sun, "Hierarchical, non-uniform locality sensitive hashing and its application to video identification," in *Multimedia and Expo, 2004. ICME'04. 2004 IEEE International Conference on*, vol. 1. IEEE, 2004, pp. 743–746.
- [41] C. Sadowski and G. Levin, "Simhash: Hash-based similarity detection," 2007.
- [42] P. Li, A. Owen, and C.-H. Zhang, "One permutation hashing," in *Advances in Neural Information Processing Systems*, 2012, pp. 3113–3121.

- [43] A. Shrivastava and P. Li, “Improved densification of one permutation hashing,” in *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*. AUAI Press, 2014, pp. 732–741.
- [44] —, “Densifying one permutation hashing via rotation for fast near neighbor search,” in *International Conference on Machine Learning*, 2014, pp. 557–565.
- [45] A. Shrivastava, “Optimal densification for fast and accurate minwise hashing,” in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 3154–3163.
- [46] P. Li, K. W. Church, and T. J. Hastie, “One sketch for all: Theory and application of conditional random sampling,” in *Advances in Neural Information Processing Systems*, 2009, pp. 953–960.