Improved Consistent Sampling, Weighted Minhash and L1 Sketching

Sergey Ioffe
Google Inc., 1600 Amphitheatre Pkwy, Mountain View, CA 94043, sioffe@google.com

***Abstract*—We propose a new Consistent Weighted Sampling method, where the probability of drawing identical samples for a pair of inputs is equal to their Jaccard similarity. Our method takes deterministic constant time per non-zero weight, improving on the best previous approach which takes expected constant time. The samples can be used as Weighted Minhash for efficient retrieval and compression (sketching) under Jaccard or L1 metric. A method is presented for using simple data statistics to reduce the running time of hash computation by two orders of magnitude. We compare our method with the random projection method and show that it has better characteristics for retrieval under L1. We present a novel method of mapping hashes to short bit-strings, apply it to Weighted Minhash, and achieve more accurate distance estimates from sketches than existing methods, as long as the inputs are sufficiently distinct. We show how to choose the optimal number of bits per hash for sketching, and demonstrate experimental results which agree with the theoretical analysis.**

***Keywords*-Sampling, Hashing, Sketching, Retrieval, Compression, Minhash.**

1. Introduction

As data sets become larger and more high-dimensional, it becomes increasingly important to find data representations that allow compact storage and efficient distance computation and retrieval. Among the common methods to achieve this is Locality Sensitive Hashing (LSH) [1]. LSH is a framework for mapping vectors into Hamming space, so that the distances in the Hamming hash space reflect those in the input space: similar vectors map to similar hashes. Many LSH schemes have been proposed, divided between metric-driven [2], [3], [4], [5], with the goal of approximating a given distance metric, and data-driven [6], [7], where the hash functions are learned to optimize performance on a task such as classification. In this work, we mostly follow the metric-driven approach, assuming that a good distance function is given (or already learned).

By mapping the (possibly sparse) floating-point vectors to a *sketch* in the Hamming hash space, we are able to considerably reduce the number of bits occupied by each data point while still being able to compute accurate approximations to the distances among the data. The distance computations become faster not only because the hash representations are more compact but also because the Hamming distances could be computed using bit operations which involve the minimal amount of floating-point computation and can benefit from bit-counting instructions available on modern processors. Hash representations often allow sublinear-time retrieval of near neighbors from large databases (e.g. [2]). In addition, the compact representations and fast distance computations can be used in kernel learning methods such as SVM or Kernel PCA for efficient kernel approximation.

In this work, we concentrate on two target metrics: Weighted Jaccard and ℓ_1. Our main motivation is text and image retrieval, where histograms, either normalized or unnormalized, are often used to represent documents (tf-idf-weighted histograms of terms or n-grams) or image statistics (such as histograms of color or texture).

The ℓ_1 distance between two vectors $\mathbf{S} = (S_k)$ and $\mathbf{T} = (T_k)$ is defined as $\|\mathbf{S} - \mathbf{T}\|_1 = \sum_k |S_k - T_k|$. This distance is a natural fit for data such as histograms, and is less sensitive to outliers than the Euclidean distance.

Weighted Jaccard distance between two vectors with non-negative entries is defined as $1 - J(\mathbf{S}, \mathbf{T})$ where J is the Jaccard Similarity

$$J(\mathbf{S}, \mathbf{T}) = \frac{\sum_k \min(S_k, T_k)}{\sum_k \max(S_k, T_k)} \tag{1}$$

There is a simple relationship between Jaccard similarity and ℓ_1 distance. Because $\min(S, T) = \frac{S+T-|S-T|}{2}$ and $\max(S, T) = \frac{S+T+|S-T|}{2}$ (as can be easily verified by considering the cases $S \leq T$ and $S \geq T$), we have

$$J(\mathbf{S}, \mathbf{T}) = \frac{\|\mathbf{S}\|_1 + \|\mathbf{T}\|_1 - \|\mathbf{S} - \mathbf{T}\|_1}{\|\mathbf{S}\|_1 + \|\mathbf{T}\|_1 + \|\mathbf{S} - \mathbf{T}\|_1}$$

In the case where the data is ℓ_1-normalized, the norms $\|\mathbf{S}\|_1$ and $\|\mathbf{T}\|_1$ are fixed and so the Jaccard similarity is a monotonically decreasing function of the ℓ_1 distance. If the data is not normalized then knowing $J(\mathbf{S}, \mathbf{T})$, $\|\mathbf{S}\|_1$ and $\|\mathbf{T}\|_1$ allows one to compute $\|\mathbf{S} - \mathbf{T}\|_1$ – so if compact representations of $\mathbf{S}$ and $\mathbf{T}$ allow one to approximate $J(\mathbf{S}, \mathbf{T})$ then their ℓ_1 distance can also be approximated by augmenting the compact representation of every vector with that vector's ℓ_1 norm.

2. Related Work

Minhash for unweighted sets or binary vectors was introduced by [3]. They showed that the probability of hash collision is given by (1), which for binary data represents the ratio of the size of the intersection to that of the union. Extensions have since been proposed. A method dealing with integer weights is mentioned in [5] and was extended in [8] to the case of $S_k = W_k\alpha_k$ where W_k is an integer weight and α_k is an *input-independent* real-valued multiplier that is fixed for each index k. For instance, W_k can be the frequency of the term k in a document, while α_k is its inverse document frequency. The time to compute a hash is $O(\sum_k W_k)$.

A scheme described in [9] and reviewed in [5] suggests a rejection sampling approach for handling real-valued weights: a sequence of samples (k, y) is sampled from an upper bound on all possible vectors, and the first sample satisfying $y \leq S_k$ is output. This method is not suitable for sparse vectors and widely varying feature weights.

The consistent sampling algorithm of [10] runs in expected constant time per non-zero weight, and can be made quite fast in practice through a number of optimizations. Here, we propose an algorithm that runs in *worst-case constant time*, is simpler to implement, and has the added advantage of requiring a fixed number of random values (3 random numbers per index k) which means that if the possible set of indices is known in advance then all the random values can be generated offline.

For a common case where all the vectors to be hashed have the same ℓ_1 norm, Weighted Jaccard Similarity is a monotonic function of ℓ_1 distance, therefore our hashing technique can be used in such cases. One of the best known LSH methods for handling ℓ_1 distances is based on stable distributions [2]. As we will show, our method has better performance characteristics for retrieval and sketching under some common conditions.

In section 4.3, we show how to compress the sketches by using a fixed number of bits per hash. This problem was previously addressed in [11], where each Minhash value is mapped to its lowest b bits. Compared to that work, our approach has two advantages. One is that it is more general and applies to any underlying hashing scheme, not just Minhash. The other is that by mapping hashes to b-bit representations randomly we end up with much simpler estimators of the distances between inputs, given the Hamming distances between their hash vectors.

3. Consistent Weighted Sampling

In this paper, we follow [10] who presented an algorithm for *consistent weighted sampling* from a weighted set and showed that consistent sampling leads to an LSH scheme where the probability of hash collision is equal to the Jaccard similarity.

Consistent weighted sampling is a sampling process that generates, for any vector $S = (S_k \geq 0)$, a sample $(k, y) : 0 \leq y \leq S_k$ which is *uniform* and *consistent*.

Uniformity: The sample should be uniformly sampled from $\cup_k\{k\} \times [0, S_k]$, i.e. the probability of selecting k is proportional to S_k, and y is uniformly distributed on $[0, S_k]$.

Consistency: If a vector $\mathbf{S}$ dominates $\mathbf{S}'$ ($\forall k\ 0 \leq S'_k \leq S_k$), a sample (k, y) is selected for $\mathbf{S}$ and satisfies $y \leq S'_k$, then (k, y) would be selected for $\mathbf{S}'$ as well.

For a uniform and consistent sampling scheme, it is easy to show that the probability of collision is the Jaccard similarity:

$$\Pr[\text{sample}(\mathbf{S}) = \text{sample}(\mathbf{T})] = J(\mathbf{S}, \mathbf{T}) = \frac{\sum_k \min(S_k, T_k)}{\sum_k \max(S_k, T_k)} \quad (2)$$

To demonstrate this, consider a sample (k, y) from the union of $\mathbf{S}$ and $\mathbf{T}$, $\mathbf{R} = (R_k = \max(S_k, T_k))$. By consistency, if $y \leq \min(S_k, T_k)$ then (k, y) would be selected for both $\mathbf{S}$ and $\mathbf{T}$, resulting in a collision. Otherwise, (k, y) would be selected for one of $\mathbf{S}$ or $\mathbf{T}$ (the former if $y \leq S_k$, the latter if $y \leq T_k$), but not the other, thus no collision. Now using the uniformity, $\Pr[\text{collision}] = \Pr[y \leq \min(S_k, T_k)] = J(\mathbf{S}, \mathbf{T})$.

In this paper, we will sometimes refer to the consistent weighted samples (or invertible functions of such samples) as *Weighted Minhash*, since they generalize the min-wise permutation hashes of [3] from binary to real weights. In sec. 3.1, we will show how to extend the binary case (standard minwise-hashing) to integer weights and then to real weights. The resulting algorithm is presented in sec. 3.2. We then prove its correctness in sec. 3.3, and describe important data-driven optimizations in sec. 3.4.

3.1. From Integer to Real Weights

Restricting our problem to integer weights, one sampling scheme [5] works by drawing independent, identically distributed random numbers (from some fixed distribution) $v_k(j)$ for each $(k, j) : j \in \{1, \ldots, S_k\}$, and returning the pair $(k^\star, j^\star) = \arg\min_{k,j\leq S_k} v_k(j)$. It is important for consistency that $v_k(j)$ depend *only* on k and j and not on $\mathbf{S}$. In practice, we achieve this by using (k, j) as the seed for the random number generator used to draw the corresponding value. This scheme is uniform because all $v_k(j)$ are i.i.d. and so have equal chances of being the minimum. It is also consistent: if $\forall k\ S'_k \leq S_k$ then $\min_{k,j\leq S'_k} v_k(j) \geq \min_{k,j\leq S_k} v_k(j)$, with the equality achieved if and only if the element $(k^\star, j^\star)$ achieving the minimum for $\mathbf{S}$ on the right-hand side also appears on the left-hand side, i.e. if $j^\star \leq S'_{k^\star}$.

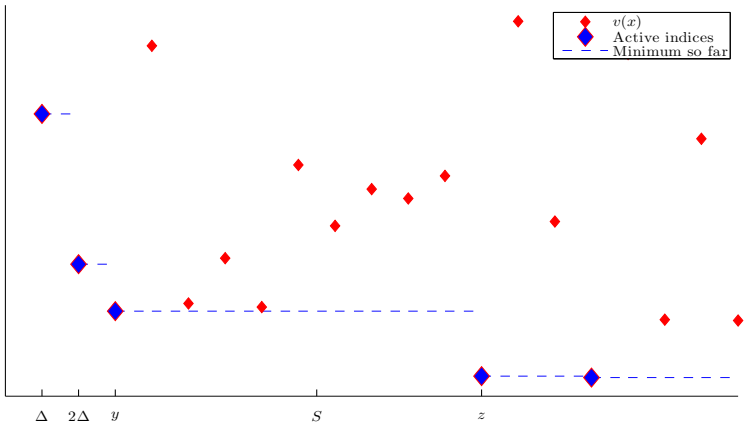

Figure 1. *The motivation for the consistent sampling algorithm is that the real weights can be discretized with granularity Δ, with random variables $v(x)$ sampled for each $x = j\Delta$, $j = 1 \dots S/\Delta$ (small red diamonds in the figure). We look for the minimum of $v(x)$, and avoid the need to sample $O(1/\Delta)$ variables by observing [10] that we need to consider only the* active indices *where the minimum found so far is reduced (large blue diamonds). Instead of sampling the individual values $v(x)$, we directly sample the active indices y and z that bracket the weight S.*

Integer weights can be used to approximate real weights, by quantizing each S_k with granularity Δ. For each index k we compute

$$y_k = \arg \min_{x=\Delta j, j=1\dots\lfloor S_k/\Delta \rfloor} v_k(x)$$

and define a_k as the corresponding minimum (achieved for $x = y_k$). Then the sample returned is $(k^\star, y^\star)$ where $k^\star = \arg\min_k a_k$ and $y^\star = y_{k^\star}$. Clearly, as $\Delta \to 0$, the integer weights tend to infinity, which makes it impossible to sample all the $v_k(x)$ directly. However, it is easy to see that y_k must be one of "active indices", where $v_k(y) < \min_{x<y} v_k(x)$. Fig. 1 illustrates this. In [10], the active indices are sampled in expected constant time per non-zero weight. Let us study the properties of active indices to find a sampling scheme that is worst-case constant time.

We will drop the subscript k where unambiguous, and let $\Delta \to 0$. It is easy to see that $v(y) = \min_{x \le S} v(x)$ is achieved at the *largest active index* y satisfying $y \le S$. Let z be the smallest active index greater than S, so that y and z are the active indices bracketing S (as shown in Fig. 1).

Since $y = \arg\min_{x<S} v(x)$ and all $v(x)$ are i.i.d., y is uniformly distributed on the interval $[0, S]$. (We often use the closed- and open-interval notation interchangeably, since the endpoints have measure zero.) Therefore, $y = u_1 S$ where $u_1 \sim \text{Uniform}(0, 1)$.

To obtain the distribution $P(z)$, let us analyze its cumulative probability function $\text{cdf}(z)$. For $z \ge S$, we can see that $\text{cdf}(z)$ is the probability that the interval $[S, z]$ contains a value $v(x)$ which falls below the minimum achieved on $[0, S]$. In other words,

$$\text{cdf}(z) = \Pr[\min_{S \le x \le z} v(x) < \min_{0 \le x \le S} v(x)]$$

which is the probability that the smallest $v(x)$ over $0 \le x \le z$ is achieved for $x > S$. Since all $v(x)$ are i.i.d., we have $\text{cdf}(z) = \frac{z-S}{z} = 1 - S/z$ (and so $P(z) = \text{cdf}'(z) = S/z^2$). Therefore, we can write $z = \text{cdf}^{-1}(1 - u_2) = \frac{S}{u_2}$, where $u_2 \sim \text{Uniform}(0, 1)$ (and so is $1 - u_2$). Note that z is independent of y – knowing *where* the minimum v occurred on $[0, S]$ tells us nothing about where the first smaller value will be obtained on (S, ∞).

It follows that $r = \ln z - \ln y = (\ln S - \ln u_2) - (\ln S + \ln u_1) = -(\ln u_1 + \ln u_2)$, where $u_1, u_2 \sim \text{Uniform}(0, 1)$. Significantly, r does not depend on S. To compute the distribution of r, we first note that if $t_1 = -\ln u_1$ then $P(t_1) = P(u_1) \left|\frac{du_1}{dt_1}\right| = e^{-t_1}$, and similar for $t_2 = -\ln u_2$. Then, $r = t_1 + t_2$ is the sum of independent variables with the distribution $P(r) = \int_0^r e^{-t_1} e^{-(r-t_1)} dt_1 = re^{-r}$. This is the definition of the $\text{Gamma}(2, 1)$ distribution:

$$P(\ln z - \ln y) = P(r) = re^{-r}$$

defined for $r \ge 0$. Consider a transformation of variables $(y, z) \to (\ln y, r)$. The distribution $P(y, z)$ is defined for $0 \le y \le S \le z$ and, from independence, $P(y, z) = P(y)P(z) = (1/S) \times (S/z^2) = z^{-2}$. The distribution $P(\ln y, r)$ is defined for $r \ge 0$ and for $\ln y$ such that $\ln y \le \ln S$ and $\ln z - \ln y = r$ for some $z \ge S$; in other words, $\ln y \in [\ln S - r, \ln S]$. From the transformation properties of probability distributions, we have $P(\ln y, r) = P(y, z) \left|\det \frac{\partial(y,z)}{\partial(\ln y, r)}\right| =$

$z^{-2}\left|\det \frac{\partial(\exp(\ln y),\ \exp(\ln y + r))}{\partial(\ln y, r)}\right| = \exp(-r)$. Therefore $P(\ln y \,|\, r) = \frac{P(\ln y, r)}{P(r)} = 1/r$ which means that, conditioned on r, $\ln y$ is uniformly distributed on $[\ln S - r, \ln S]$.

The way in which we sample $\ln y$ uniformly from $[\ln S - r, \ln S]$ has a critical effect on the consistency of our sampling scheme. For example, one option is to set $\ln y = \ln S - rb$ where $b \sim \text{Uniform}(0, 1)$ does not depend on S. This, however, does not produce consistent samples: even a small change in S will cause y to change – while in a consistent scheme we should have a non-zero probability of producing identical samples for similar inputs. Instead, we use a uniform $\beta \sim \text{Uniform}(0, 1)$ to specify the offset of a regular grid with spacing r, and select, as the sample $\ln y$, the unique element of the grid that falls inside the interval $(\ln S - r, \ln S]$. Specifically, we set

$$\ln y = \left(\left\lfloor \frac{\ln S}{r} + \beta \right\rfloor - \beta\right) r$$
$$\ln z = r + \ln y$$

The use of the piecewise-constant floor function ensures that y and z are usually unaffected by small changes in S, and give rise to consistent samples (as we will show below).

Next, we turn to the random variable $a_k = \min_{0 \le x \le S_k} v_k(x)$, which we need to minimize over indices k to produce the consistent sample. Notice that we are free to choose any distribution for the $v_k(x)$. Because the minimum of a set of exponential variables is itself exponential, we will draw each $v_k(x)$ from the exponential distribution with rate Δ: $P(v) = \Delta e^{-\Delta v}$, $v \ge 0$. It is easy to show that the minimum a of S/Δ such variables is then exponential with rate S, and the product $\epsilon_1 = aS$ is exponential with rate 1: $P(\epsilon_1) = e^{-\epsilon_1}$.

The value $a = \min_{0 \le x \le S} v(x)$ is connected to the next active index $z > S$, which is where $v(z)$ drops below a for the first time. The probability of any v falling below a is $\Pr[v < a] = 1 - e^{-\Delta a} \approx 1 - (1 - \Delta a) = \Delta a$, and it follows that, as $\Delta \to 0$, the waiting time $z - S$ until encountering a sample below a is exponential, with rate a. Therefore, $\epsilon_2 = (z - S)a$ is exponential with rate 1.

Putting it together, we have, for two independent exponential variables ϵ_1 and ϵ_2, $a = \frac{Sa + (z-S)a}{z} = \frac{\epsilon_1 + \epsilon_2}{z} = c/z$ where $c = \epsilon_1 + \epsilon_2 \sim \text{Gamma}(2, 1)$: $P(c) = ce^{-c}$. This means that a can be sampled conditioned on z and independent of S. Furthermore, if z is not usually affected by small changes in S then neither is a, giving the promise of consistency – which we will prove in Sec. 3.3.

3.2. Consistent Weighted Sampling Algorithm

From the derivations in Sec. 3.1, we obtain a scheme for sampling from a weighted set $\mathbf{S} = (S_k \ge 0)$.

- For each k:
  - Sample $r_k, c_k \sim \text{Gamma}(2, 1)$ ($P(r) = re^{-r}$) and $\beta_k \sim \text{Uniform}(0, 1)$.
  - Compute

$$t_k = \left\lfloor \frac{\ln S_k}{r_k} + \beta_k \right\rfloor$$
$$y_k = \exp\left(r_k(t_k - \beta_k)\right)$$
$$z_k = y_k \exp(r_k)$$
$$a_k = c_k / z_k$$

- Find $k^\star = \arg\min_k a_k$, and return the sample $(k^\star, y_{k^\star})$.

If we do not care about the actual sample but only the collision of samples, then we output the Weighted Minhash $(k^\star, t_{k^\star})$ instead (since the mapping $t_k \to y_k$ does not depend on $\mathbf{S}$), which avoids the use of floating-point values as hashes. To compute multiple hashes, we simply draw different independent sets of variables r_k, β_k, c_k.

We can avoid most transcendental function computations by working in the log domain. To sample a variable $r \sim \text{Gamma}(2, 1)$, we can represent it as $r = -\ln(u_1 u_2)$ where $u_1, u_2 \sim \text{Uniform}(0, 1)$. Alternatively, we can use the efficient Ziggurat method [12]. We can sample c in the same way, or can instead sample $\ln c$ directly, also using Ziggurat. Figure 2 summarizes the algorithm.

3.3. Proof of Correctness

Below, we will prove that the algorithm does in fact return uniform and consistent samples.

1) Uniformity

We shall prove that our sampling scheme produces a uniform sample from $\{(k, y) : 0 \le y \le S_k\}$.

Dropping k from notation, observe that the random variable $y = \exp\left(r\left(\left\lfloor \frac{\ln S}{r} + \beta \right\rfloor - \beta\right)\right)$, where β is uniform, comes from the same distribution as $y = \exp(\ln S - rb)$ for uniform b (in both cases, $\ln y$ is uniformly distributed on $[\ln S - r, \ln S]$). The distribution $P(y, z, a)$ is defined for $y \le S$, $z \ge S$ and $a > 0$, and can be computed using a transformation of variables:

$$P(y, z, a) = P(r, b, c)\left|\det \frac{\partial(r, b, c)}{\partial(y, z, a)}\right|$$

where $r = \ln z - \ln y$, $b = \frac{\ln S - \ln y}{\ln z - \ln y}$, $c = az$. From independence of r, b and c, and computing the Jacobian, we get

$$P(y, z, a) = \frac{re^{-r}ce^{-c}}{y(\ln z - \ln y)} = ae^{-az}$$

Input: Vector $\mathbf{S} = (S_k \geq 0)$
Output: Consistent uniform sample $(k^\star, y^\star)$ from $\{(k, y) : 0 \leq y \leq S_k\}$

Sample independent random variables, independent of the input $\mathbf{S}$
for all k **do**
  Sample:
    $r_k \sim \text{Gamma}(2, 1)$ (i.e. $P(r_k) = r_k e^{-r_k}$, $r_k \geq 0$)
    $c_k \sim \text{Gamma}(2, 1)$
    $\beta_k \sim \text{Uniform}(0, 1)$
end for

The remainder depends on the input $\mathbf{S}$
for all k such that $S_k > 0$ **do**
  $t_k \leftarrow \left\lfloor \frac{\ln S_k}{r_k} + \beta_k \right\rfloor$
  $y_k \leftarrow \exp\left(r_k(t_k - \beta_k)\right)$
  $a_k \leftarrow c_k / \left(y_k \exp(r_k)\right)$
end for
$k^\star \leftarrow \arg\min_k a_k$
$y^\star \leftarrow y_{k^\star}$
return Consistent sample $(k^\star, y^\star)$ or Weighted Minhash $(k^\star, t_{k^\star})$

Figure 2. *Algorithm for drawing a uniform consistent sample. The resulting* $(k^\star, y^\star)$ *(or, equivalently,* $(k^\star, t_{k^\star})$*) can be used as a* Weighted Minhash*, where the probability of hash collision for inputs* $\mathbf{S}$ *and* $\mathbf{T}$ *equals the Jaccard similarity* $J(\mathbf{S}, \mathbf{T})$*. We sample* $r \sim \text{Gamma}(2, 1)$ *as the sum of two independent exponential variables, or* $r = -\ln u_1 - \ln u_2$ *where* $u_1, u_2 \sim \text{Uniform}(0, 1)$*. In practice, we work with the logs* $\ln y_k$ *and* $\ln a_k$*, and sample* r_k *and* $\ln c_k$ *from their respective distributions using Ziggurat method.*

Integration over z yields

$$P(y, a) = \int_S^\infty P(y, z, a)dz = e^{-Sa} = \frac{1}{S}(Se^{-Sa}) = P(y)P(a)$$

which shows that y is uniform on $[0, S]$, a is exponential with rate S, and y and a are independent. Now considering all indices k, it is easy to show that, for a set of exponential variables a_k with respective rates S_k, the probability that a particular $a_{k^\star}$ is the smallest is proportional to $S_{k^\star}$: $\Pr[a_{k^\star} = \min_k a_k] = S_{k^\star}/(\sum_k S_k)$. By independence of a and y, the corresponding $y^\star = y_{k^\star}$ is uniformly distributed on $[0, S_k]$. Therefore, $(k^\star, y^\star)$ is uniformly sampled from $\{(k, y) : 0 \leq y \leq S_k\}$.

2) Consistency

Let us fix all of r_k, β_k and c_k. We will show that for two weighted sets $\mathbf{S}$, $\mathbf{S}'$ such that $\forall k\ S_k \geq S'_k$, if $(k^\star, y^\star)$ was sampled for $\mathbf{S}$ and satisfies $y^\star \leq S'_k$, then $(k^\star, y^\star)$ will also be sampled for $\mathbf{S}'$.

Note that $t_{k^\star} = \left\lfloor \frac{\ln S_{k^\star}}{r_{k^\star}} + \beta_{k^\star} \right\rfloor$ is the unique integer such that $\frac{\ln S_{k^\star}}{r_{k^\star}} + \beta_{k^\star} - 1 < t_{k^\star} \leq \frac{\ln S_{k^\star}}{r_{k^\star}} + \beta_{k^\star}$. Since, by assumption, $y^\star \leq S'_{k^\star} \leq S_{k^\star}$, we have

$$\frac{\ln S'_{k^\star}}{r_{k^\star}} + \beta_{k^\star} - 1 \leq \frac{\ln S_{k^\star}}{r_{k^\star}} + \beta_{k^\star} - 1 < t_{k^\star} = \frac{\ln y^\star}{r_{k^\star}} + \beta_{k^\star} \leq \frac{\ln S'_{k^\star}}{r_{k^\star}} + \beta_{k^\star}.$$

We see that

$$\frac{\ln S'_{k^\star}}{r_{k^\star}} + \beta_{k^\star} - 1 < t_{k^\star} \leq \frac{\ln S'_{k^\star}}{r_{k^\star}} + \beta_{k^\star}$$

and so

$$t_{k^\star} = \left\lfloor \frac{\ln S'_{k^\star}}{r_{k^\star}} + \beta_{k^\star} \right\rfloor = t'_{k^\star}$$

Therefore $y_{k^\star} = y^\star = y'_{k^\star}$ where $y'_{k^\star}$ is the value sampled for index $k^\star$ for input $\mathbf{S}'$. For any given k, notice that $a_k = c_k e^{-r_k}/y_k$ is a monotonically decreasing function of y_k, which in turn is a non-decreasing function of S_k. Thus, for all k, $a'_k \geq a_k$, while $a'_{k^\star} = a_{k^\star}$ (since $y'_{k^\star} = y_{k^\star}$). It follows that $\arg\min_k a'_k = \arg\min_k a_k = k^\star$ which means that the sample $(k^\star, y^\star = y_{k^\star})$ is output for $\mathbf{S}'$ as well as for $\mathbf{S}$, and so our sampling scheme is consistent.

3.4. Efficient Hash Computation

We can significantly speed up the computation, by computing simple data statistics. Observe that in each sample $(k^\star, y^\star)$ (or, equivalently, hash $(k^\star, t^\star)$), if we know $k^\star$ then $y^\star$ depends only on $k^\star$ and the corresponding weight

$S_{k^\star}$, but not on any of the other weights. Similarly, if we know that $k^\star \in \mathcal{K}$ then we only need to examine a subset of indices $\{k \in \mathcal{K} : S_k > 0\}$ to generate the hash. In practice, we have observed that we can find a rather small set $\mathcal{K}$ for each hash, such that most of the hashes have $k^\star \in \mathcal{K}$. Different hashes may have different candidate index sets $\mathcal{K}$, with some indices belonging to most candidate sets (if the corresponding S_k are usually large) and some belonging to none (if their S_k are usually zero or very small). We count how frequently each index k is chosen by the hash, over a large data set, and for future hash computations consider only the indices for which this frequency is above a threshold. In our experiments, we were able to reduce the number of considered hash/feature pairs by a factor of 200, while affecting only 0.5% of the hashes.

An extra advantage of our method compared to that of [10] is that for each feature and each hash we sample 3 random variables (r, c and β), independently of the input weight. This suggests that these random variables may be generated ahead of time (perhaps only for those hash / feature pairs which get selected sufficiently frequently) – eliminating the need to sample those for every input, and further reducing the hash computation time.

4. Weighted Minhash for Retrieval and Sketching under ℓ_1 Metric

Below, we show how Weighted Minhash can be used for data in ℓ_1 spaces. We will discuss the use of these hashes for near neighbor retrieval and for sketching the data, where the ℓ_1 distance between pairs is approximated based on their hashes. We will demonstrate how the hashes can be mapped to a small number of bits to save space, and how the theoretical properties of Weighted Minhash compare with the commonly-used random projection method. It should be pointed out that the discussion in this section is not specific to our method, and applies to any consistent weighted sampling scheme.

4.1. Retrieval under ℓ_1 Metric

Often, we deal with ℓ_1 normalized data, such as histograms — i.e., for all $\mathbf{S}$, $\mathbf{S}_1 = 1$. In this case, Weighted Jaccard Similarity is a monotonic function of ℓ_1 distance:

$$J(\mathbf{S}, \mathbf{T}) = \frac{2 - \|\mathbf{S} - \mathbf{T}\|_1}{2 + \|\mathbf{S} - \mathbf{T}\|_1}.$$

This allows Weighted Minhash to be used for efficient retrieval under ℓ_1. The constraint on the norm is the only one that we impose — the non-negativity constraint in (1) can be relaxed by replacing each weight S_k with a pair

$$S_k \rightarrow (S_{k+}, S_{k-}) = (\max(0, S_k), \max(0, -S_k)) \quad (3)$$

which preserves ℓ_1 distances and norms.

We compare Weighted Minhash with the stable distribution method of [2], which uses a quantized dot-product with a vector of Cauchy variates. The crucial quantity affecting the retrieval speed in the (R, c)-approximate nearest neighbor problem [1] is $\rho = \frac{\ln(1/p_1)}{\ln(1/p2)}$ where p_1 is the probability of hash collision for points at distance $R > 0$ and p_2 is the probability for points at distance Rc, with $c > 1$ the approximation factor. Smaller ρ results in faster retrieval. Both [1] and [2] propose hashing schemes that achieve $\rho = 1/c$. With Weighted Minhash, $\rho_c(R)$ depends on R and is defined on $R \in (0, 2/c)$: $\rho_c(R) = \frac{\ln((2+R)/(2-R))}{\ln((2+Rc)/(2-Rc))}$. Fig. 3 shows $\rho_c(R)$ for several values of c. We can show that $\lim_{R\rightarrow 0} \rho_c(R) = 1/c$ and $\lim_{R\rightarrow 2/c} \rho_c(R) = 0$. Furthermore, $\rho_c(R)$ decreases on $(0, 2/c)$ for all values of c that we examined. We can see that $\rho_c(R) < 1/c$ and so, in the case of ℓ_1-normalized data, Weighted Minhash allows for more efficient retrieval than the stable distribution method.

In addition to its suitability for retrieval, Weighted Minhash can also be faster to compute. Indeed, computing a single stable-distribution hash requires us to consider every non-zero weight $S_k \neq 0$ in the input vector to compute the dot product. On the other hand, with the preprocessing described in Sec. 3.4, we can consider only a small fraction of the features to compute a given hash — which is a win compared to the dot product with a Cauchy vector, even though our algorithm spends more time on each weight that it actually considers.

4.2. Weighted Minhash for Sketching Under ℓ_1

We often want to represent the data using as few bits as possible, while being able to reconstruct the distances efficiently and accurately. We now present a method for accomplishing this with Weighted Minhash, study the accuracy of the ℓ_1 distance estimation for a given number of bits, and compare with several existing methods.

Consider the mapping of a vector $\mathbf{S}$ to a *sketch* containing its ℓ_1 norm and H Weighted Minhash values, where each WMH_h is an independent sample from the parameterized family $\mathcal{H}$ of hash functions:

$$\mathrm{sketch}(\mathbf{S}) = (\|\mathbf{S}\|_1, \mathrm{WMH}_1(\mathbf{S}), \ldots, \mathrm{WMH}_H(\mathbf{S}))$$

Recall that $\Pr_{\mathrm{WMH}\in\mathcal{H}}[\mathrm{WMH}(\mathbf{S}) = \mathrm{WMH}(\mathbf{T})] = J(\mathbf{S}, \mathbf{T})$, where the probability is with respect to a random selection of the hash function from $\mathcal{H}$. Since all of WMH_h are independent random samples from $\mathcal{H}$, the normalized Hamming similarity

$$\mathrm{HashSim}_H(\mathbf{S}, \mathbf{T}) = \frac{1}{H} \sum_{h=1}^{H} \delta(\mathrm{WMH}_h(\mathbf{S}), \mathrm{WMH}_h(\mathbf{T}))$$

(where δ is Kronecker's delta) is the average of H i.i.d. Bernoulli variables. Thus the Hamming similarity has the

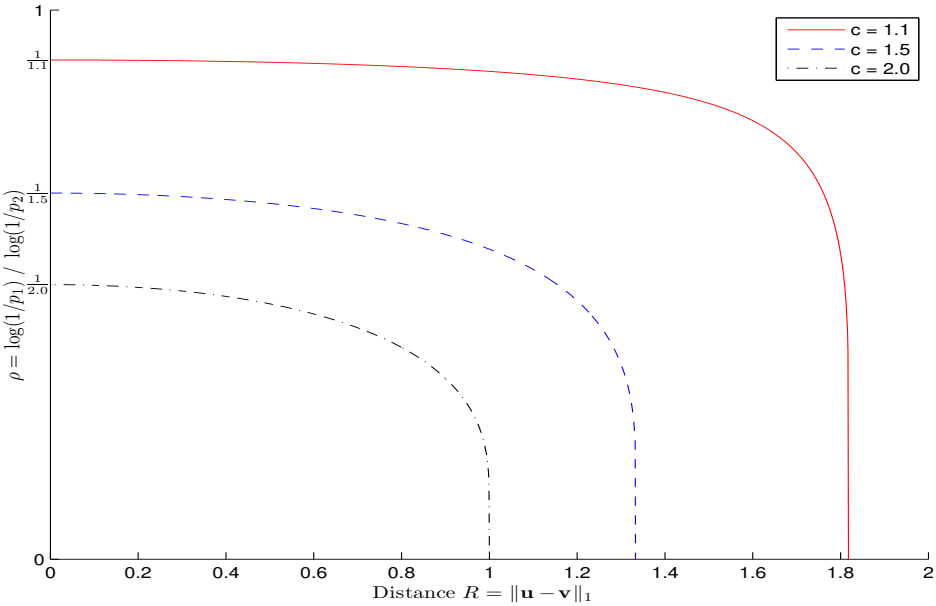

Figure 3. *The value ρ achieved by our method for different approximation factors c and ℓ_1 distances R, under the condition $\|\mathbf{S}\|_1 = \|\mathbf{T}\|_1 = 1$. p_1 is the probability of hash collision for points at distance R and p_2 is the probability for points at distance Rc. Smaller values of ρ result in more efficient retrieval. Previously, $\rho = 1/c$ has been achieved for ℓ_1 distances using the stable distribution method, whereas Weighted Minhash achieves $\rho = 1/c$ for distances $R \approx 0$ and quickly decreases after that. These graphs show that when the data is ℓ_1 normalized (such as histograms), Weighted Minhash provides a more efficient retrieval method than the method of [2].*

expected value of $J(\mathbf{S}, \mathbf{T})$ and variance

$$\sigma_H^2 = \frac{J(\mathbf{S}, \mathbf{T})(1 - J(\mathbf{S}, \mathbf{T}))}{H}$$

Being able to approximate $J(\mathbf{S}, \mathbf{T}) = \frac{\sum_k \min(S_k, T_k)}{\sum_k \max(S_k, T_k)}$ allows us to approximate $\|\mathbf{S} - \mathbf{T}\|_1$. Indeed, $\sum_k \min(S_k, T_k) = \sum_k (S_k + T_k - |S_k - T_k|)/2$, and similar for $\sum_k \max(S_k, T_k)$, so

$$J(\mathbf{S}, \mathbf{T}) = \frac{\|\mathbf{S}\|_1 + \|\mathbf{T}\|_1 - \|\mathbf{S} - \mathbf{T}\|_1}{\|\mathbf{S}\|_1 + \|\mathbf{T}\|_1 + \|\mathbf{S} - \mathbf{T}\|_1}.$$

Defining $N = \|\mathbf{S}\|_1 + \|\mathbf{T}\|_1$ and $d = \|\mathbf{S} - \mathbf{T}\|_1$, we have

$$d = N \frac{1 - J}{1 + J}$$

Replacing J with $\mathrm{HashSim}_H(\mathbf{S}, \mathbf{T})$ yields an estimate $\hat{d} \approx d$.

How accurate is this estimate? Let us locally approximate the relationship between J and d as being linear. Then,

$$\mathrm{Var}[\hat{d}] = \sigma_H^2 \times \left(\frac{\partial d}{\partial J}\right)^2 \tag{4}$$

$$= \frac{J(1 - J)}{H} \times \left(\frac{\partial J}{\partial d}\right)^{-2} \tag{5}$$

4.3. Generating b-bit hashes

In the above discussion we ignored the number of bits required to represent each Weighted Minhash WMH_h. One possibility is to use one of the standard compression techniques so that the average number of bits per hash is roughly the entropy of the hash. This approach, however, has several disadvantages. One is that the distance computations between hashes become much more expensive, as each hash needs to be decompressed on the fly. More importantly, it may often be advantageous to reduce the number of bits per hash at the cost of some spurious hash collisions, with the benefit of being able to use more hashes. The extra hash collisions, which a compression scheme would work hard to avoid, may not be quite so harmful in reality, since it is unlikely that the spurious collisions of *different* hashes would be consistent enough to significantly affect the distance estimates. This insight was has been successfully used in the context of hash learning [6]. Below, we propose a scheme to represent each hash using a given number b of bits – where b trades off between the number of hashes and their accuracy. This problem has also been addressed in [11], but the method below applies to any hashing scheme and not just Minhash, and allows simpler analysis and much simpler estimators of distances given the number of hash collisions.

Given a hash value $\mathrm{WMH}_h(\mathbf{S})$, we will randomly map it to a b-bit value. First, we initialize a random

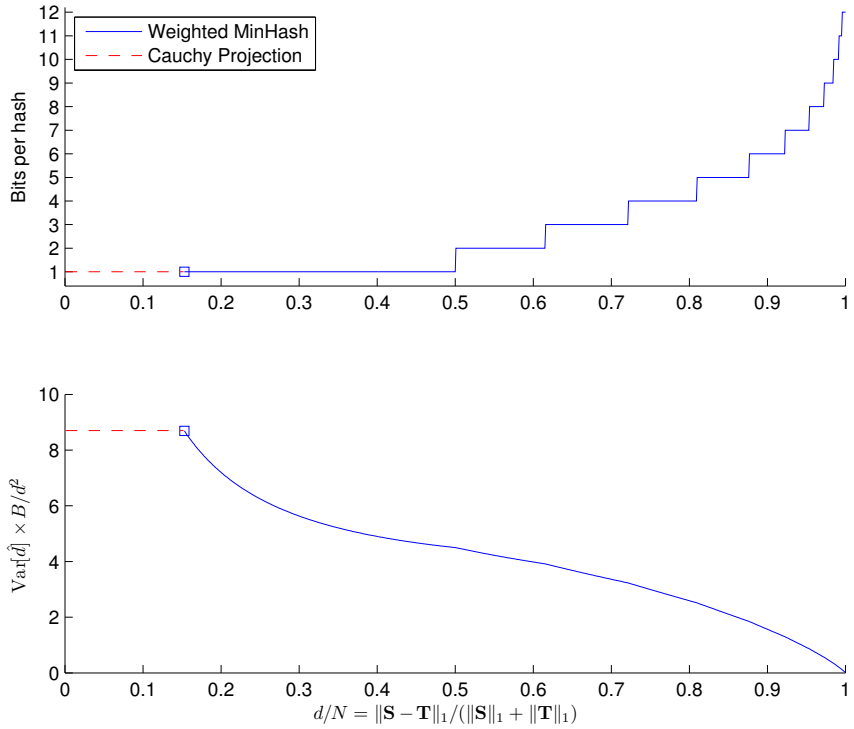

Figure 4. *Comparing Weighted Minhash with Cauchy Random Projections for estimating the distance $d = \|\mathbf{S} - \mathbf{T}\|_1$. In both cases, each hash is mapped to a b-bit code, and we minimize the variance of the distance estimate given that we are allowed B bits total. For random projections, the minimum variance $\mathrm{Var}[\hat{d}] \approx 8.7d^2/B$ is obtained for $b = 1$. For Weighted Minhash, we explore different values of b. We normalize out the effects of B and d by plotting $\mathrm{Var}[\hat{d}] \times B/d^2$ against $d/N = \|\mathbf{S} - \mathbf{T}\|_1/(\|\mathbf{S}\|_1 + \|\mathbf{T}\|_1)$.* (top) *The optimal number of bits b to which each Weighted Minhash value should be mapped to minimize $\mathrm{Var}[\hat{d}]$. For a near-duplicate scenario where $d < 0.153N$, it is better to use random projections, at one bit per projection. Otherwise, Weighted Minhash performs better.* (bottom) *The optimal normalized variance. For Weighted Minhash, we benefit from minimizing the vector norms, e.g. by centering each dimension at its median.*

number generator using $(h, \mathrm{WMH}_h(\mathbf{S}))$ as the seed. Then sample an integer b-bit value $\mathrm{WMH}_h^{(b)}$ uniformly from $\{0 \dots 2^b - 1\}$.

Note that if the original hashes are equal then the b-bit hashes are too. For different original hashes, on the other hand, the b-bit hashes collide with probability 2^{-b}. Therefore,

$$\Pr[\mathrm{WMH}_h^{(b)}(\mathbf{S}) = \mathrm{WMH}_h^{(b)}(\mathbf{T})] = J + (1 - J) \times 2^{-b} = J^{(b)}$$

and, denoting by $\mathrm{HashSim}_H^{(b)}$ the Hamming similarity of H b-bit hashes, we can approximate

$$J = \frac{J^{(b)} - 2^{-b}}{1 - 2^{-b}} \approx \frac{\mathrm{HashSim}_H^{(b)} - 2^{-b}}{1 - 2^{-b}}$$

Similar to (4), we obtain that the variance of the estimate $\hat{d}^{(b)}$ of the distance $d = \|\mathbf{S} - \mathbf{T}\|_1$ depends on H and b as follows:

$$\mathrm{Var}[\hat{d}^{(b)}] = \frac{J^{(b)}(1 - J^{(b)})}{H} \times \left(\frac{\partial J^{(b)}}{\partial d}\right)^{-2}$$

Denoting by B the total number of bits available, we have $H = B/b$, and

$$\mathrm{Var}[\hat{d}^{(b)}] = \frac{d(N + d)^2(N - d(1 - 2^{1-b}))\, b}{2N^2(1 - 2^{-b})B} \qquad (6)$$

The distance estimate variance computed in (6) allows us to reason about the optimal number of bits b to represent each hash, assuming that the pairs of vectors we care about have similar distances d and sums of norms N. As an alternative, we may choose b empirically to obtain the most accurate estimate $\hat{d}$ for the vector pairs of interest.

4.4. Weighted Minhash or Random Projections?

Let us now consider another commonly used Locality Sensitive Hashing method, based on Stable Distributions

[2]. Here, each hash is a quantized dot product between the vector $\mathbf{S}$ and a vector of independent Cauchy variates. The method has a single parameter r, the quantization granularity. For given $\mathbf{S}$, $\mathbf{T}$ and r, let us define $u = \frac{r}{\|\mathbf{S}-\mathbf{T}\|_1}$. Then, the probability of hash collision is

$$p = \frac{2\tan^{-1} u}{\pi} - \frac{\ln(1+u^2)}{\pi u} \tag{7}$$

We can estimate p as the fraction of colliding hashes, and recover $d = \|\mathbf{S}-\mathbf{T}\|_1$ by inverting $p(u)$. For H hashes, the variance of the estimate is

$$\mathrm{Var}[\hat{d}] = \frac{p(1-p)}{H} \times \left(\frac{\partial p}{\partial d}\right)^{-2} \tag{8}$$

To represent the hash in a fixed number b of bits, we can consider several schemes. One is the scheme described above, where each hash is mapped to a random b-bit sequence. Alternatively, we propose to minimize spurious collisions by making the regions represented with the same b bits as separated as possible, as follows:

$$\mathrm{RP}^{(b)}(\mathbf{S}) = \mod\left(\left\lfloor \frac{\mathbf{S}\cdot\mathbf{c}}{r} + \beta \right\rfloor, 2^b\right) \tag{9}$$

where $\mathbf{c}$ is the vector of Cauchy variates and β is the random offset. Numerical minimization with respect to r and b shows that for a given distance $d = \|\mathbf{S}-\mathbf{T}\|_1$ and available number of bits B the smallest

$$\mathrm{Var}[\hat{d}_{\mathrm{RP}}] \approx \frac{8.7d^2}{B} \tag{10}$$

is achieved by the scheme in (9) with $r \approx 4.5d$ and $b = 1$ — that is, the random projections are quantized, and the quantization bins are alternately assigned to bits 0 and 1 (for comparison, random assignment of each bin to a single bit yields the estimate variance of $11.6d^2/B$). Note that the in this analysis we are being optimistic, assuming that for each d we can select the corresponding r value, which of course is not true in practice.

In addition to the hashing methods, we can consider storing non-quantized Cauchy projections, and estimating the $\|\mathbf{S}-\mathbf{T}\|_1$ as the maximum likelihood estimate of the scale of a Cauchy distribution. Such a method was proposed in [13], and yields

$$\mathrm{Var}[\hat{d}] = \frac{2d^2}{k} + O(k^{-2})$$

where k is the number of projections. However, even if we can very aggressively compress the projections to 5 bits with no loss in distance estimation accuracy (so that $k = B/5$), the resulting variance exceeds that in (10).

To compare our sketching method with that given by Cauchy random projections with quantization and remapping to bits, we observe that $\mathrm{Var}[\hat{d}]$ scales inversely with B and quadratically with d, with the caveat for our method that N/d remains constant. Therefore, to compare the methods as well as different values of b for Weighted Minhash, in Fig. 4 we plot $\mathrm{Var}[\hat{d}]/d^2$ against d/N.

From the plots we can make several observations. One is that for a fixed $d = \|\mathbf{S}-\mathbf{T}\|_1$ we obtain smaller variance $\mathrm{Var}[\hat{d}]$ if $d/(\|\mathbf{S}\|_1 + \|\mathbf{T}\|_1)$ is larger. Therefore, it is advantageous to make the ℓ_1 norms of the inputs as small as possible, while preserving distances. We propose to minimize the sum of ℓ_1 norms of the training data by determining, for each dimension k, the median weight $\mu_k = \mathrm{median}\{S_k\}$ over the training data. Then, we subtract the medians from the weights and replace each weight with a pair to ensure non-negativity:

$$S_k \rightarrow (\max(0, S_k - \mu_k), \max(0, \mu_k - S_k)) \tag{11}$$

It is easy to see that this transformation preserves the ℓ_1 distances, while minimizing the total ℓ_1 norm of the data.

The second observation from Fig. 4 is that Weighted Minhash achieves better variance of the distance estimate than random projections with remapping to 1 bit, as long as the vectors are sufficiently different, i.e. $\|\mathbf{S}-\mathbf{T}\|_1 > 0.153\,(\|\mathbf{S}\|_1 + \|\mathbf{T}\|_1)$. On the other hand, in the near-duplicate scenario, where $d/N < 0.153$, we are better off using random projections. One explanation for this is that the norms of the inputs $\|\mathbf{S}\|_1$ which we store alongside the Weighted Minhashes help us obtain better distance estimates when these distances are of the same order of magnitude as the norms. The norms become less useful, however, if the distances are much smaller – at that point they only amplify the variances.

Finally, we can see that as the distances we are interested in become larger compared to the norms, we benefit from using more bits per hash, and need fewer hashes.

5. Experimental Results

In our evaluations, we have concentrated on sketching, and analyzed how well the variances of the distance estimate obtained from sketches agree with the theoretical predictions.

Our inputs are image descriptors containing different histograms of color, texture and other image properties, for a total of about 5×10^5 dimensions. The descriptors are somewhat sparse, with about 50000 non-zero weights per image. Some of the constituent histograms are normalized and some are not. We preprocessed the descriptors by subtracting from each dimension the median value of that dimension, computed over a separate set of images, ensuring non-negativity as in (11).

The average norm of a descriptor is $\|\mathbf{S}\|_1 \approx 14.2$, and the average distance $\|\mathbf{S}-\mathbf{T}\|_1 \approx 19.9$. Looking up these values in Fig. 4, we find that for $d/N \approx 19.9/(2 \cdot 14.2) \approx 0.7$, we should use Weighted Minhash, with 3 bits per

hash. We computed the hashes for different values of b, with the budget of 3000 bytes per image ($B = 24000$), and found that $b = 3$ in fact gives the best agreement between the distances and their estimates. Turning now to (6), we expect that $\mathrm{Var}[\hat{d}] \approx 0.0556$. In our experiment, we obtained almost exactly this value: averaging over many pairs of images, we got $\mathrm{E}(\hat{d} - d)^2 = 0.0559$.

The time required to compute the $B/b = 8000$ hashes, with about 50000 non-zero weights per input, was approximately 7 seconds. Following Sec. 3.4, we selected for each hash the 200 most-frequently selected input dimensions (as evaluated on a separate set of inputs). Now each hash looks only at the dimensions *among the 200* which have non-zero weights; the sets of dimensions considered by different hashes may or may not overlap. With this optimization, the variance $\mathrm{Var}[\hat{d}]$ increases slightly from 0.0559 to 0.0572. However, the running time is reduced from 7 seconds to 47 milliseconds per image, i.e. a speedup factor of almost 150.

6. Conclusion

We presented a new consistent sampling scheme, which improves on the best existing such scheme by making it worst-case constant-time per sample per non-zero input weight (compared to expected constant-time). Another advantage of our method is that all the random variates used for hash computation can be computed offline, independently of the inputs. The time required to compute the samples was reduced by a factor of 150 by analyzing which samples are affected by which input dimensions. The consistent samples can be used as generalization of Minhash to weighted inputs, with the collision probability given by the Jaccard similarity. We exploit the relationship between Jaccard similarity and ℓ_1 distance to present a hash-based approximate near-neighbor retrieval scheme with performance characteristics superior to the previously existing ones. We discussed the use of Weighted Minhash for sketching that allows ℓ_1 distance approximation, and studied the accuracy of this approximation for a given sketch size. We showed how to map Weighted Minhash, or any other hash, to a fixed number of bits, and investigated the optimal trade-off between the number of hashes and the number of bits per hash. Our experimental results agree well with the theoretical predictions.

References

[1] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, New York, NY, USA, 1998, pp. 604–613.

[2] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *SCG '04: Proceedings of the twentieth annual symposium on Computational geometry*, New York, NY, USA, 2004, pp. 253–262.

[3] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher, "Min-wise independent permutations," *J. Comput. Syst. Sci.*, vol. 60, no. 3, pp. 630–659, 2000.

[4] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," in *FOCS '06: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, Washington, DC, USA, 2006, pp. 459–468.

[5] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *STOC '02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, New York, NY, USA, 2002, pp. 380–388.

[6] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *Proc. Neural Information Processing Systems 2008*, 2008.

[7] R. Salakhutdinov and G. Hinton, "Semantic hashing," in *IRGM 2007 workshop at the SIGIR 2007 conference*, 2007.

[8] O. Chum, J. Philbin, and A. Zisserman, "Near duplicate image detection: min-hash and tf-idf weighting," in *Proceedings of the British Machine Vision Conference*, 2008.

[9] J. Kleinberg and E. Tardos, "Approximation algorithms for classification problems with pairwise relationships: Metric labeling and markov random fields," in *FOCS '99: Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, Washington, DC, USA, 1999, p. 14.

[10] M. Manasse, F. McSherry, and K. Talwar, "Consistent weighted sampling," MSR, Tech. Rep. MSR-TR-2010-73, 2010.

[11] P. Li and C. König, "b-bit minwise hashing," in *WWW '10*, 2010.

[12] G. Marsaglia and W. W. Tsang, "The ziggurat method for generating random variables," *Journal of Statistical Software*, vol. 5, no. 8, pp. 1–7, 2000.

[13] P. Li, T. J. Hastie, and K. W. Church, "Nonlinear estimators and tail bounds for dimension reduction in l1 using cauchy random projections," *J. Mach. Learn. Res.*, vol. 8, 2007.