Proofs and Additional Experiments for FastLSH

No Author Given

No Institute Given

1 Proofs

1.1 Proof of Theorem 1

Proof. Let $f_{|\tilde{s}X|}(t)$ represent the PDF of the absolute value of $\tilde{s}X$. For given bucket width $\tilde{w}$, the probability $p(|\tilde{s}X| < t)$ for any pair $(\boldsymbol{v}, \boldsymbol{u})$ is computed as $p(|\tilde{s}X| < t) = \int_0^{\tilde{w}} f_{|\tilde{s}X|}(t)dt$, where $t \in [0, \tilde{w}]$. Recall that $\tilde{b}$ follows the uniform distribution $U(0, \tilde{w})$, the probability $p(\tilde{b} < \tilde{w} - t)$ is thus $(1 - \frac{t}{\tilde{w}})$. This means that after random projection, $(|\tilde{s}X| + \tilde{b})$ is also within the same bucket, and the collision probability $p(s, \sigma)$ is the product of $p(|\tilde{s}X| < t)$ and $p(\tilde{b} < \tilde{w} - t)$. Hence we prove this Theorem.

1.2 Proof of Lemma 2

Proof. For the characteristic function of W, we can write:

$$
\begin{aligned}
\varphi_W(x) &= E_W\{\exp(ixW)\} \\
&= E_{XY}\{\exp(ixXY)\} \\
&= E_Y\{E_{X|Y}\{\exp(ixXY)|Y\}\} \\
&= \int_{-\infty}^{+\infty} E_{X|Y}\{\exp(-ixXY)|Y\} f(Y)dY \\
&= \int_{-\infty}^{+\infty} (\int_{-\infty}^{+\infty} \exp(-ixXY) f(X|Y)dX) f(Y)dY \\
&= \int_{-\infty}^{+\infty} \exp(-\frac{x^2Y^2}{2}) f(Y)dY \\
&= E_Y\{\exp(-\frac{x^2Y^2}{2})\}
\end{aligned}
$$

where standard normal random variable X is eliminated by its characteristic function. We prove this Lemma.

1.3 Proof of Lemma 3

Proof. According to Eqn.(8), we know the PDF of $\tilde{s}$. By applying Lemma 2, we have the following result:

$$\begin{aligned}
\varphi_{\tilde{s}X}(x) &= \frac{1}{\sqrt{2\pi}\tilde{\sigma}}\int_0^{+\infty}\frac{2y}{\Phi(a_2;\tilde{\mu},\tilde{\sigma}^2)-\Phi(a_1;\tilde{\mu},\tilde{\sigma}^2)}\exp(\frac{-x^2y^2}{2}-\frac{(y^2-\tilde{\mu})^2}{2\tilde{\sigma}^2})dy \\
&= \frac{1}{\sqrt{2\pi}\tilde{\sigma}}\int_0^{+\infty}\frac{1}{\Phi(a_2;\tilde{\mu},\tilde{\sigma}^2)-\Phi(a_1;\tilde{\mu},\tilde{\sigma}^2)}\exp(-\frac{x^2y^2}{2}-\frac{(y^2-\tilde{\mu})^2}{2\tilde{\sigma}^2})dy^2 \\
&= \frac{1}{\sqrt{2\pi}\tilde{\sigma}}\int_0^{+\infty}\frac{1}{\Phi(a_2;\tilde{\mu},\tilde{\sigma}^2)-\Phi(a_1;\tilde{\mu},\tilde{\sigma}^2)}\exp(\frac{-(y^2-(\tilde{\mu}-\frac{1}{2}x^2\tilde{\sigma}^2))^2-\tilde{\mu}x^2\tilde{\sigma}^2+\frac{1}{4}x^4\tilde{\sigma}^4)}{2\tilde{\sigma}^2})dy^2 \\
&= \frac{1}{\sqrt{2\pi}\tilde{\sigma}}\int_0^{+\infty}\frac{1}{\Phi(a_2;\tilde{\mu},\tilde{\sigma}^2)-\Phi(a_1;\tilde{\mu},\tilde{\sigma}^2)}\exp(\frac{1}{8}x^4\tilde{\sigma}^2-\frac{1}{2}\tilde{\mu}x^2)\exp(\frac{-(y^2-(\tilde{\mu}-\frac{1}{2}x^2\tilde{\sigma}^2))^2)}{2\tilde{\sigma}^2})dy^2 \\
&= \frac{1}{2(\Phi(\frac{a_2-\tilde{\mu}}{\tilde{\sigma}})-\Phi(\frac{a_1-\tilde{\mu}}{\tilde{\sigma}}))}\exp(\frac{1}{8}x^4\tilde{\sigma}^2-\frac{1}{2}\tilde{\mu}x^2)\,\mathrm{erfc}(\frac{\frac{1}{2}x^2\tilde{\sigma}^2-\tilde{\mu}}{\sqrt{2}\tilde{\sigma}}) \\
&= \frac{1}{2(1-\Phi(\frac{-\tilde{\mu}}{\tilde{\sigma}}))}\exp(\frac{1}{8}x^4\tilde{\sigma}^2-\frac{1}{2}\tilde{\mu}x^2)\,\mathrm{erfc}(\frac{\frac{1}{2}x^2\tilde{\sigma}^2-\tilde{\mu}}{\sqrt{2}\tilde{\sigma}})
\end{aligned}$$

where $\tilde{\mu}=\frac{ms^2}{n}$, $\tilde{\sigma}^2=m\sigma^2$, $a_2=\infty$ and $a_1=0$. Hence we prove this Lemma.

1.4 Proof of Theorem 2

Proof. Recall that $\tilde{\mu}=\frac{ms^2}{n}$ and $\tilde{\sigma}=m\sigma^2$. $\varphi_{\tilde{s}X}(x)$ can be written as follows:

$$\varphi_{\tilde{s}X}(x)=\frac{1}{2(1-\Phi(\frac{-\sqrt{m}s^2}{n\sigma}))}\exp(\frac{mx^4\sigma^2}{8}-\frac{ms^2x^2}{2n})\,\mathrm{erfc}(\frac{\sqrt{m}(nx^2\sigma^2-2s^2)}{2\sqrt{2}n\sigma})\quad(-\infty<x<+\infty)$$

Let $a=\frac{s^2}{\sqrt{2}n\sigma}>0$ and $b=\frac{\sigma}{2\sqrt{2}}>0\Rightarrow b^2=\frac{\sigma^2}{8}$. According to the fact $\Phi(x)=\frac{1}{2}\mathrm{erfc}(-\frac{x}{\sqrt{2}})$, $\varphi_{\tilde{s}X}(x)$ is simplified as:

$$\varphi_{\tilde{s}X}(x)=\exp(b^2mx^4-\frac{1}{2}m\mu x^2)\cdot\frac{\mathrm{erfc}(b\sqrt{m}x^2-a\sqrt{m})}{2-\mathrm{erfc}(a\sqrt{m})}$$

Let $g(x)=\frac{\varphi_{\tilde{s}X}(x)}{\varphi_{sX}(x)}$, where $\varphi_{sX}(x)=\exp(-\frac{ms^2x^2}{2n})$ is the characteristic function of $\mathcal{N}(0,\frac{ms^2}{n})$. Then $g(x)$ is denoted as:

$$g(x)=\exp(b^2mx^4)\cdot\frac{\mathrm{erfc}(b\sqrt{m}x^2-a\sqrt{m})}{2-\mathrm{erfc}(a\sqrt{m})}$$

To prove $\varphi_{\tilde{s}X}(x)=\varphi_{sX}(x)$, we convert to prove whether $g(x)=1$ as $m\to\infty$. Obviously $x^2\le O(m^{-1})$. Then $\sqrt{m}x^2\le O(m^{-1/2})$ and $mx^4\le O(m^{-1})$. It is easy to derive:

$$\lim_{m\to+\infty}\exp(b^2mx^4)=1$$

It holds for any fixed $b \in \mathbb{R}^+$. On the other hand, we have:

$$\operatorname{erfc}(b\sqrt{m}x^2 - a\sqrt{m}) \sim \operatorname{erfc}(-a\sqrt{m}), \quad m \to +\infty$$

Actually using the fact $b\sqrt{m}x^2 - a\sqrt{m} \sim -a\sqrt{m}$ as $m \to +\infty$ and $\operatorname{erfc}(-x) = 2 - \frac{\exp(-x^2)}{\sqrt{\pi}x}$ as $x \to +\infty$, we have:

$$\lim_{m\to+\infty} \frac{\operatorname{erfc}(b\sqrt{m}x^2 - a\sqrt{m})}{2 - \operatorname{erfc}(a\sqrt{m})} = \frac{\lim_{m\to+\infty} \operatorname{erfc}(b\sqrt{m}x^2 - a\sqrt{m})}{\lim_{m\to+\infty} \operatorname{erfc}(-a\sqrt{m})} = 1$$

Hence we have:

$$g(x) = \lim_{m\to+\infty} \exp(b^2mx^4) \cdot \lim_{m\to+\infty} \frac{\operatorname{erfc}(b\sqrt{m}x^2 - a\sqrt{m})}{\operatorname{erfc}(-a\sqrt{m})} = 1.$$

We prove this Theorem.

1.5 Proof of Lemma 4

Proof. If the characteristic function of a random variable Z exists, it provides a way to compute its various moments. Specifically, the r-th moment of Z denoted by $E(Z^r)$ can be expressed as the r-th derivative of the characteristic function evaluated at zero [9], i.e.,

$$E(Z^r) = (i)^{-r} \frac{d^r}{dt^r} \varphi_Z(t) \mid_{t=0} \tag{1}$$

where $\varphi_Z(t)$ denotes the characteristic function of Z. If we know the characteristic function, then all of the moments of the random variable Z can be obtained.

From Lemma 3, we can easily compute the first-order derivative of characteristic function with respect to x, which is as follows:

$$\varphi'_{\tilde{s}X}(x) = \frac{\exp(\frac{1}{8}x^4\tilde{\sigma}^2 - \frac{1}{2}\tilde{\mu}x^2)}{2(1 - \Phi(\frac{-\tilde{\mu}}{\tilde{\sigma}}))} \left[(\frac{1}{2}x^3\tilde{\sigma}^2 - \tilde{\mu}x) \operatorname{erfc}(\frac{\frac{1}{2}x^2\tilde{\sigma}^2 - \tilde{\mu}}{\sqrt{2}\tilde{\sigma}}) - \frac{2\tilde{\sigma}x}{\sqrt{2\pi}} \exp(-(\frac{\frac{1}{2}x^2\tilde{\sigma}^2 - \tilde{\mu}}{\sqrt{2}\tilde{\sigma}})^2) \right] \tag{2}$$

Then the second-order derivative is

$$\begin{aligned}\varphi''_{\tilde{s}X}(x) =& \frac{\exp(\frac{1}{8}x^4\tilde{\sigma}^2 - \frac{1}{2}\tilde{\mu}x^2)}{2(1 - \Phi(\frac{-\tilde{\mu}}{\tilde{\sigma}}))} \Bigg[((\frac{1}{2}x^3\tilde{\sigma}^2 - ux)^2 + \frac{3}{2}x^2\tilde{\sigma}^2 - \tilde{\mu}) \operatorname{erfc}(\frac{\frac{1}{2}x^2\tilde{\sigma}^2 - \tilde{\mu}}{\sqrt{2}\tilde{\sigma}}) \\ & - \frac{2}{\sqrt{2\pi}}(\tilde{\sigma} + \frac{3}{2}x^4\tilde{\sigma}^3 - \frac{3}{2}\tilde{\mu}\tilde{\sigma}x^2) \exp(-(\frac{\frac{1}{2}x^2\tilde{\sigma}^2 - \tilde{\mu}}{\sqrt{2}\tilde{\sigma}})^2) \Bigg]\end{aligned} \tag{3}$$

The third-order derivative is

$$\begin{aligned}\varphi'''(x) =& \frac{\exp(\frac{1}{8}x^4\tilde{\sigma}^2 - \frac{1}{2}\tilde{\mu}x^2)}{2(1 - \Phi(\frac{-\tilde{\mu}}{\tilde{\sigma}}))} \Bigg[(\frac{1}{8}\tilde{\sigma}^6x^9 - \frac{3}{4}\tilde{\mu}\tilde{\sigma}^4x^7 + (\frac{2}{3}\tilde{\mu}^2\tilde{\sigma}^2 + \frac{9}{4}\tilde{\sigma}^4)x^5 - (6\tilde{\mu}\tilde{\sigma}^2 + \tilde{\mu}^3)x^3 + 3(\tilde{\mu}^2 + \tilde{\sigma}^2)x) \\ & \operatorname{erfc}(\frac{\frac{1}{2}\tilde{\sigma}^2x^2 - \tilde{\mu}}{\sqrt{2}\tilde{\sigma}}) - \frac{2}{\sqrt{2\pi}}(\frac{1}{4}\tilde{\sigma}^5x^7 - \tilde{\mu}\tilde{\sigma}^3x^5 + (\tilde{\mu}^2\tilde{\sigma} + \frac{7}{2}\tilde{\sigma}^3)x^3 - 3\tilde{\mu}\tilde{\sigma}x) \exp(-(\frac{\frac{1}{2}\tilde{\sigma}^2x^2 - \tilde{\mu}}{\sqrt{2}\tilde{\sigma}})^2) \Bigg]\end{aligned} \tag{4}$$

The fourth-order derivative is

$$\varphi''''(x) = \frac{\exp(\frac{1}{8}x^4\tilde{\sigma}^2 - \frac{1}{2}\tilde{\mu}x^2)}{2(1-\Phi(\frac{-\tilde{\mu}}{\tilde{\sigma}}))}\Big[(\frac{1}{16}\tilde{\sigma}^8x^{12} - \frac{1}{2}\tilde{\mu}\tilde{\sigma}^6x^{10} + (\frac{3}{2}\tilde{\mu}^2\tilde{\sigma}^4 + \frac{9}{4}\tilde{\sigma}^4)x^8 - (2\tilde{\mu}^3\tilde{\sigma}^2 + \frac{21}{2}\tilde{\mu}\tilde{\sigma}^4)x^6$$
$$+(\tilde{\mu}^4 + \frac{51}{4}\tilde{\sigma}^4 + 9\tilde{\mu}^2\tilde{\sigma}^2)x^4 - (6\tilde{\mu}^3 + 21\tilde{\mu}\tilde{\sigma}^2)x^2 + 3(\tilde{\mu}^2 + \tilde{\sigma}^2))\,\mathrm{erfc}(\frac{\frac{1}{2}\tilde{\sigma}^2x^2 - \tilde{\mu}}{\sqrt{2}\tilde{\sigma}})$$
$$-\frac{2}{\sqrt{2\pi}}(\frac{1}{8}\tilde{\sigma}^7x^{10} - \frac{3}{4}\tilde{\mu}\tilde{\sigma}^5x^8 + (\frac{3}{2}\tilde{\mu}^2\tilde{\sigma}^3 + 4\tilde{\sigma}^5)x^6 - (6\tilde{\mu}^2\tilde{\sigma} + \frac{13}{2}\tilde{\sigma}^3)x^2 - 3\tilde{\mu}\tilde{\sigma})\exp(-(\frac{\frac{1}{2}\tilde{\sigma}^2x^2 - \tilde{\mu}}{\sqrt{2}\tilde{\sigma}})^2)\Big] \tag{5}$$

Let $E(\tilde{s}X - E(\tilde{s}X))^i$ for $i \in \{1,2,3,4\}$ denote the first four central moments. According to Eqn.(1), we know that $E(\tilde{s}X) = \frac{\varphi'(0)}{i} = 0$, then it is easy to derive $E(\tilde{s}X - E(\tilde{s}X))^i = E((\tilde{s}X)^i)$. To this end, by Eqn.(2) -(5), we have the following results:

$$\begin{cases} E(\tilde{s}X) \quad = 0 \\ E((\tilde{s}X)^2) = \frac{\tilde{\mu}\,\mathrm{erfc}(\frac{-\tilde{\mu}}{\sqrt{2}\tilde{\sigma}}) + \frac{2\tilde{\sigma}}{\sqrt{2\pi}}\exp(\frac{-\tilde{\mu}^2}{2\tilde{\sigma}^2})}{2(1-\Phi(\frac{-\tilde{\mu}}{\tilde{\sigma}}))} \\ E((\tilde{s}X)^3) = 0 \\ E((\tilde{s}X)^4) = \frac{3(\tilde{\mu}^2+\tilde{\sigma}^2)\,\mathrm{erfc}(\frac{-\tilde{\mu}}{\sqrt{2}\tilde{\sigma}}) + \frac{6\tilde{\mu}\tilde{\sigma}}{\sqrt{2\pi}}\exp(\frac{-\tilde{\mu}^2}{2\tilde{\sigma}^2})}{2(1-\Phi(\frac{-\tilde{\mu}}{\tilde{\sigma}}))} \end{cases} \tag{6}$$

where $E(\tilde{s}X)$ is the expectation; $E((\tilde{s}X)^2)$ is the variance; $\frac{E((\tilde{s}X)^3)}{(E((\tilde{s}X)^2))^{\frac{3}{2}}}$ is the skewness; $\frac{E((\tilde{s}X)^4)}{(E((\tilde{s}X)^2))^2}$ is the kurtosis. Since

$$\frac{\mathrm{erfc}(\frac{-\tilde{\mu}}{\sqrt{2}\tilde{\sigma}})}{2(1-\Phi(\frac{-\tilde{\mu}}{\tilde{\sigma}}))} = \frac{\frac{2}{\sqrt{\pi}}\int_{\frac{-\tilde{\mu}}{\sqrt{2}\tilde{\sigma}}}^{+\infty}\exp(-t_1^2)dt_1}{2(1-\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\frac{-\tilde{\mu}}{\tilde{\sigma}}}\exp(\frac{-t_2^2}{2})dt_2)} = \frac{\int_{\frac{-\tilde{\mu}}{\sqrt{2}\tilde{\sigma}}}^{+\infty}\exp(-t_1^2)dt_1}{\sqrt{2}\int_{\frac{-\tilde{\mu}}{\tilde{\sigma}}}^{+\infty}\exp(\frac{-t_2^2}{2})dt_2} = \frac{\int_{\frac{-\tilde{\mu}}{\sqrt{2}\tilde{\sigma}}}^{+\infty}\exp(-t_1^2)dt_1}{\int_{\frac{-\tilde{\mu}}{\sqrt{2}\tilde{\sigma}}}^{+\infty}\exp(-t^2)dt} = 1$$

Therefore, Eqn.(6) can be rewritten as below

$$\begin{cases} E(\tilde{s}X) \quad = 0 \\ E((\tilde{s}X)^2) = \tilde{\mu}(1+\epsilon) \\ E((\tilde{s}X)^3) = 0 \\ E((\tilde{s}X)^4) = 3\tilde{\mu}^2(1+\lambda) \end{cases}$$

where $\epsilon = \frac{\tilde{\sigma}\exp(\frac{-\tilde{\mu}^2}{2\tilde{\sigma}^2})}{\sqrt{2\pi}\tilde{\mu}(1-\Phi(\frac{-\tilde{\mu}}{\tilde{\sigma}}))}$, $\lambda = \frac{\tilde{\sigma}^2}{\tilde{\mu}^2} + \epsilon$, $\tilde{\mu} = \frac{ms^2}{n}$ and $\tilde{\sigma}^2 = m\sigma^2$. We prove this Lemma.

2 Extension to Maximum Inner Product

[3,19] shows that there exists two transformation functions, by which the maximum inner product search (MIPS) problem can be converted into solve the near neighbor search problem. More specifically, the two transformation functions are

$P(\boldsymbol{v}) = (\sqrt{\kappa^2 - \|\boldsymbol{v}\|_2^2}, \boldsymbol{v})$ for data processing and $Q(\boldsymbol{u}) = (0, \boldsymbol{u})$ for query processing respectively, where $\kappa = max(\|\boldsymbol{v}_i\|_2)$ $(i \in \{1, 2, \dots, N\})$. Then the relationship between maximum inner product and l_2 norm for any vector pair $(\boldsymbol{v}_i, \boldsymbol{u})$ is denoted as $\underset{i}{argmax}(\frac{P(\boldsymbol{v}_i)Q(\boldsymbol{u})}{\|P(\boldsymbol{v}_i)\|_2\|Q(\boldsymbol{u})\|_2}) = \underset{i}{argmin}(\|P(\boldsymbol{v}_i) - Q(\boldsymbol{u})\|_2)$ for $\|\boldsymbol{u}\|_2 = 1$. To make FastLSH applicable for MIPS, we first apply the sample operator $S(\cdot)$ defined earlier to vector pairs $(\boldsymbol{v}_i, \boldsymbol{u})$ for yielding $\tilde{\boldsymbol{v}}_i = S(\boldsymbol{v}_i)$ and $\tilde{\boldsymbol{u}} = S(\boldsymbol{u})$, and then obtain $\tilde{P}(\boldsymbol{v}) = (\sqrt{\tilde{\kappa}^2 - \|S(\boldsymbol{v})\|_2^2}, S(\boldsymbol{v}))$ and $\tilde{Q}(\boldsymbol{u}) = (0, S(\boldsymbol{u}))$, where $\tilde{\kappa} = max(\|S(\boldsymbol{v}_i)\|_2)$ is a constant. Then $\underset{i}{argmax}(\frac{\tilde{P}(\boldsymbol{v}_i)\tilde{Q}(\boldsymbol{u})}{\|\tilde{P}(\boldsymbol{v}_i)\|_2\|\tilde{Q}(\boldsymbol{u})\|_2}) = \underset{i}{argmin}(\|\tilde{P}(\boldsymbol{v}_i) - \tilde{Q}(\boldsymbol{u})\|_2)$ for $\|S(\boldsymbol{u})\|_2 = 1$. Let $\triangle = \tilde{\kappa}^2 - \|S(\boldsymbol{v})\|_2^2$. After random projection, $\boldsymbol{a}^T\tilde{P}(\boldsymbol{v}) - \boldsymbol{a}^T\tilde{Q}(\boldsymbol{u})$ is distributed as $(\sqrt{\tilde{s}^2 + \triangle})X$. Since $\tilde{s}^2 \sim \mathcal{N}(m\mu, m\sigma^2)$, then $(\tilde{s}^2 + \triangle) \sim \mathcal{N}(m\mu + \triangle, m\sigma^2)$. Let $\sqrt{\tilde{s}^2 + \triangle}$ be the random variable $\mathcal{I}$. Similar to Eqn.(8), the PDF of $\mathcal{I}$ represented by $f_{\mathcal{I}}$ is yielded as follow:

$$f_{\mathcal{I}}(t) = 2t\psi(t^2; m\mu + \triangle, m\sigma^2, 0, +\infty)$$

By applying Lemma 2, the characteristic function of $\mathcal{I}X$ is as follows:

$$\varphi_{\mathcal{I}X}(x) = \frac{1}{2(1 - \Phi(\frac{-ms^2 - n\triangle}{\sqrt{m}\sigma\triangle}))}\exp(\frac{mx^4\sigma^2}{8} - \frac{(ms^2 + n\triangle)x^2}{2n})\,\mathrm{erfc}(\frac{mnx^2\sigma^2 - 2(ms^2 + n\triangle)}{2\sqrt{2mn}\sigma})(-\infty < x < +\infty)$$

Then the PDF of $\mathcal{I}X$ is obtained by $\varphi_{\mathcal{I}X}(x)$:

$$f_{\mathcal{I}X}(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty}\varphi_{\mathcal{I}X}(x)exp(-itx)dx$$

It is easy to derive the collision probability of any pair $(\boldsymbol{v}, \boldsymbol{u})$ by $f_{\mathcal{I}X}(t)$, which is as follows:

$$p(s) = \int_0^{\tilde{w}'} f_{|\mathcal{I}X|}(t)(1 - \frac{t}{\tilde{w}'})dx$$

where $f_{|\mathcal{I}X|}(t)$ denotes the PDF of the absolute value of $\mathcal{I}X$. $\tilde{w}'$ is the bucket width.

3 Additional Experiments

In this section we present more description about datasets, parameter setting and additional experiments for three machine learning tasks. All experiments for nearest neighbor search and outlier detection are carried out on a server with six-cores Intel(R), i7-8750H @ 2.20GHz CPU and 32 GB RAM, in Ubuntu 20.04. Experiments for neural network training are carried out on a server with fourteen-cores Intel(R), i7-12700H @ 2.30GHz CPU and 64 GB RAM, in Ubuntu 20.04.

Table 1. Statistics of the datasets

Datasets	# of Instances	# of Outliers	Dimension
Statlog Shuttle	34,987	879	9
a9a	48,842	7841	123
Musk	6,598	97	166

3.1 Outlier Detection

Baseline: Anomaly detection is a critical task in data analysis, which aims to identify instances or patterns that deviate from expected behavior. For anomaly detection task, there are two kinds of methods, i.e., supervised and unsupervised methods. Unsupervised methods are more preferred in practice due to their ability to adapt to changing data distributions without requiring label information. Existing unsupervised anomaly detection approaches often require storing the entire dataset, leading to poor computational and memory requirements, particularly as data volume increases. To overcome these limitations, researchers propose Arrays of (locality-sensitive) Count Estimators (ACE) in [14], a novel anomaly detection algorithm, for high-speed streaming data and constrained memory environments.

ACE is an efficient anomaly detection data structure, which is composed of multiple locality-sensitive hash tables. These hash tables are used to estimate counts of collision and detect anomalies by performing hash lookup. Specifically, a hash code $H(\boldsymbol{v}) = (h_1(\boldsymbol{v}), h_2(\boldsymbol{v}), \ldots, h_k(\boldsymbol{v})$ of k bits is computed using k independent LSH functions. Then L groups of hash functions $H_i(\cdot), i = \{1, \cdots, L\}$ are drawn independently and uniformly at random from the LSH family. Instead of constructing L hash tables, ACE constructs L short arrays, A_i, of size 2^k each initialized with zeros. Given any observed element $\boldsymbol{v} \in \mathbf{D}$, ACE increments the count of the corresponding counter $H_i(\boldsymbol{v})$ in array A_i for all i.

To decide if $\boldsymbol{u}$ is an outlier, ACE computes the average of all the counters for $\forall i \in \{1, 2, \ldots L\}$, i.e., $\hat{S}(\boldsymbol{u}, \mathbf{D}) = \frac{1}{L}\sum_{i=1}^{L} A_i[H_i(\boldsymbol{u})]$. $\boldsymbol{u}$ is reported as anomaly if the estimated score $\hat{S}(\boldsymbol{u}, \mathbf{D})$ is less than $\mu - \sigma$, where $\mu = \frac{1}{N}\sum_{i=1}^{N} \hat{S}(\boldsymbol{v}_i, \mathbf{D})$ is the mean of the scores over all $\boldsymbol{v} \in \mathbf{D}$ and σ is the standard deviation. To evalute FastLSH and ACHash in the outlier detection task, we only need to replace the hash functions used in ACE with FastLSH and ACHash. The corresponding methods are termed as FastACE (FastLSH + ACE) and ACHashACE (ACHash +ACE), respectively.

Evaluation Metrics: Similar to [14], the following five performance measures are used: 1) outliers reported, i.e., the number of outliers detected by the algorithm. 2) correctly reported, i.e., the number of truth outliers in the outliers reported. 3) outlier missed, i.e., the remaining number of truth outliers in all truth outliers after outliers reported having been detected. 4) the execution time taken to report the outliers. 5) the speedup over ACE.

Datasets: We choose three widely used real-world benchmark datasets for anomaly detection, the statistics of which are presented in Table 1.

Table 2. Statistics of Datasets

Datasets	Feature Dim	Feature Sparsity	Label Dim	Training Size	Testing Size
Delicious-200K	782,585	0.038%	205,443	196,606	100,095
Amazon-670K	135,909	0.055%	670,091	490,449	153,025

Statlog Shuttle[1]: It is the dataset of radiator positions in a NASA space shuttle with 9 attributes designed for supervised anomaly detection. The dataset includes 34987 instances with 879 anomalies.

a9a[2]: It is obtained from UCI Adult dataset, which predicts whether income exceeds 50K dollars per year. The dataset contains 48842 instances with each having 123 features. There are two classes of labels denoted as -1 and 1, where 1 is the income exceeding 50K, while the other is lower than 50K.

Musk[3]: It is the set of 102 molecules of which 39 molecules are judged by human experts to be musks and the remaining 93 molecules are non-musks. These molecules are to generate 6598 conformations with each conformation contains 166 features.

Parameter Settings: We use $k = 15$ and $L = 50$ for ACE, ACHashACE and FastACE as in [14]. For FastACE, m is set to 3 for Statlog Shuttle, and $m = 30$ for the other two datasets. For ACHashACE, the sampling ratio is set to the default 0.25 [7].

3.2 Neural Network Training

Baseline: Deep Learning (DL) has drawn a lot of attention in recent years, transforming fields such as computer vision, natural language processing and speech recognition. Training a DL mode from scratch demands massive computing resources. While dedicated hardware offers accelerated performance in matrix multiplication, it comes with risks and limitations, including substantial investment requirements and the possibility of becoming obsolete with advancements in algorithms. To this end, SLIDE (Sub-LInear Deep learning Engine) [6] is proposed to train DL models using only commodity CPUs by exploiting adaptive sparsity in neural networks, especially in large fully connected neural networks.

SLIDE is a novel deep learning engine that integrates randomized algorithms (LSH) and multi-core parallelism. It achieves training speeds faster than using Tensorflow on GPU, exclusively utilizing a CPU on large-scale recommendation datasets [5,16,17]. Briefly SLIDE works as follows. In a fully connected neural network, each layer consists of a list of neurons and a set of LSH sampling hash tables. During network initialization, the weights are randomly initialized, and $k \times L$ LSH hash functions are set up along with L hash tables for each layer. These hash functions compute hash codes $h_l(w_l^a)$ for the weight vectors

[1] https://archive.ics.uci.edu/ml/datasets/Statlog+(Shuttle)
[2] https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/binary.html
[3] https://archive.ics.uci.edu/dataset/75/musk+version+2

Table 3. Statistics of Datasets

Datasets	# of Points	# of Queries	Dimension
Sun	69106	200	512
Cifar	50000	200	512
Audio	53387	200	192
Trevi	99000	200	4096
Notre	333000	200	128
Sift	1000000	200	128
Gist	1000000	200	960
Deep	1000000	200	256
Ukbench	1000000	200	128
Glove	1192514	200	100
ImageNet	2340000	200	150
Random	1000000	200	100

of neurons in each layer, where h_l represents the hash function in layer l and w_l^a is the weights for the a^{th} neuron in layer l. The neuron's id a is stored in the hash buckets determined by the LSH function $h_l(w_l^a)$. In SLIDE, rather than calculating all activations in each layer, the input to each layer v_l is passed through hash functions to compute $h_l(v_l)$. These hash codes act as queries to retrieve the ids of active (or sampled) neurons from corresponding buckets in the hash tables. For each training data instance, the neuron backpropagates partial gradients (using error propagation) exclusively to active neurons in previous layers through the connected weights.

Evaluation Metrics: We report the classification accuracy, the end-to-end training time and the number of iterations as in [6].

Datasets: We employ two large real datasets, Delicious-200K and Amazon-670K, from the Extreme Classification Repository [5], and the statistics of the two datasets are presented in Table 2. Delicious-200K dataset is a subsampled dataset generated from a vast corpus of almost 150 million bookmarks from Social Bookmarking Systems. Amazon-670K dataset is a product to product recommendation dataset with 670K labels.

Parameter Settings: For both datasets, a standard fully connected neural network with one hidden layer of size 128 and a batch size of 128 is employed. All algorithms are executed until convergence. To evaluate the performance of SLIDE, FastSLIDE (FastLSH + SLIDE) and ACHashSLIDE (ACHash + SLIDE), we utilize the same optimizer, Adam, while adjusting the initial step size from $1e^{-5}$ to $1e^{-3}$ to ensure better convergence across all experiments. In SLIDE, we particularly focus on maintaining hash tables for the last layer, which is often a computational bottleneck in the models. In terms of LSH settings, we set $k = 9$ and $L = 50$ for Decilious-200K and $k = 8$ and $L = 50$ for Amazon-670K. The hash tables are updated initially every $N_0 = 50$ iterations and then exponentially decayed. The sampling ratio of FastSLIDE is set to 0.15 and 0.07

for Delicious and Amazon, respectively. For ACHashSLIDE, the sampling ratio is set to the default 0.25 [7].

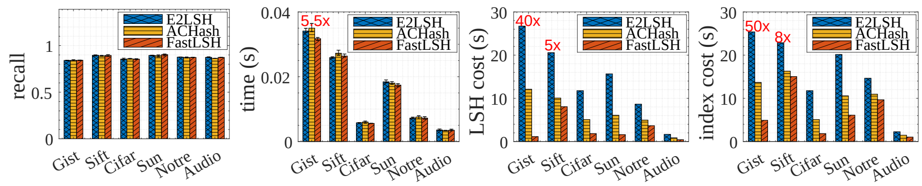
3.3 FastLSH vs. E2LSH for Nearest Neighbor Search

Baseline: Nearest neighbor search (NNS) is an essential problem in machine learning, which has numerous applications such as face recognition, information retrieval and duplicate detection. The purpose of nearest neighbor search is to find the point in the dataset $\mathbf{D} = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_N\}$ that is most similar (has minimal distance) to the given query $\boldsymbol{u}$. For low dimensional spaces (<10), popular tree-based index structures such as KD-tree [4], SR-tree [11], etc. deliver exact answers and provide satisfactory performance. For high dimensional spaces, however, these index structures suffer from the well-known curse of dimensionality, that is, their performance is even worse than that of linear scans [18]. To address this issue, one feasible way is to use approximate nearest neighbor (ANN) search by trading accuracy for efficiency [12]. Locality-sensitive hashing (LSH) is an effective randomized technique in machine learning, which is originally proposed to solve the problem of approximate nearest neighbor (ANN) search in high dimensional space [10,8,2]. The basic idea of LSH is to map high dimensional points into buckets in low dimensional space using random hash functions, by which similar points have higher probability to end up in the same bucket than dissimilar points.

The canonical LSH index structure (E2LSH) for ANN search is built as follows. A hash code $H(\boldsymbol{v}) = (h_1(\boldsymbol{v}), h_2(\boldsymbol{v}), \ldots, h_k(\boldsymbol{v})$ is computed using k independent LSH functions (i.e., $H(\boldsymbol{v})$ is the concatenation of k elementary LSH codes). Then a hash table is constructed by adding the 2-tuple $\langle H(\boldsymbol{v}), id\ of\ \boldsymbol{v}\rangle$ into corresponding bucket. To boost accuracy, L groups of hash functions $H_i(\cdot), i = 1, \cdots, L$ are drawn independently and uniformly at random from the LSH family, resulting in L hash tables.

To answer a query $\boldsymbol{u}$, one need to first compute $H_1(\boldsymbol{u}), \cdots, H_L(\boldsymbol{u})$ and then search all these L buckets to obtain the combined set of candidates. Then, all points in the candidate set are evaluated against the query and the most similar points are returned. There exists two ways (approximate and exact) to process these candidates. In the approximate version, no more than $3L$ points in the candidate set are evaluated. The LSH theory ensures that the (c, R)-NN is found with a constant probability. In practice, however, the exact one is widely used since it offers better accuracy at the cost of evaluating all points in the candidate set [8]. The search time consists of both the hashing time and the time taken to prune the candidate set [8,1]. In many cases, nearest neighbor search is just one component of a larger application that involves other approximations. As a result, using approximate neighbors instead of exact ones often leads to minimal performance loss. Therefore, we use the exact method to process a query similar to [8,1].

Evaluation Metrics: To evaluate the performance of FastLSH and baselines, we present the following metrics: 1) recall, i.e., the fraction of near neighbors that are actually returned; 2) the average running time to report the near



(a) Recall (b) Query Time (c) Hashing Cost (d) Indexing Time

Fig. 1. Comparison of recall, average query time, LSH computation and index construction time.

neighbors for each query; 3) the time taken to compute hash functions; 4) the end-to-end index construction time.

Datasets: 11 publicly available high-dimensional real datasets and one synthetic dataset are experimented with [13], the statistics of which are listed in Table 3. Sun[4] is the set of containing about 70k GIST features of images. Cifar[5] is denoted as the set of 50k vectors extracted from TinyImage. Audio[6] is the set of about 50k vectors extracted from DARPA TIMIT. Trevi[7] is the set of containing around 100k features of bitmap images. Notre[8] is the set of features that are Flickr images. Sift[9] is the set of containing 1 million SIFT vectors. Gist[10] is the set that is consist of 1 million image vectors. Deep[11] is the set of 1 million vectors that are deep neural codes of natural images obtained by convolutional neural network. Ukbench[12] is the set of vectors containing 1 million features of images. Glove[13] is the set of about 1.2 million feature vectors extracted from Tweets. ImageNet[14] is the set of data points containing about 2.4 million dense SIFT features. Random is the synthetic dataset containing 1 million randomly selected vectors in a unit hypersphere.

Parameter Settings: For the same dataset and target recall, we use identical k (number of hash functions per table) and L (number of hash tables) for fairness. Thus, three algorithms take the same space occupations. m is set to 30 throughout all experiments for FastLSH. To achieve optimal performance, the sampling ratio for ACHash is set to the defaulst 0.25 [7]. Table 4 reports the parameters (k, L and bucket width w) for different target recall illustrated in Fig. 3, where w is the bucket width of E2LSH, $\tilde{w}$ and w' are those of FastLSH and ACHash respectively.

[4] http://groups.csail.mit.edu/vision/SUN/
[5] http://www.cs.toronto.edu/ kriz/cifar.html
[6] http://www.cs.princeton.edu/cass/audio.tar.gz
[7] http://phototour.cs.washington.edu/patches/default.htm
[8] http://phototour.cs.washington.edu/datasets/
[9] http://corpus-texmex.irisa.fr
[10] https://github.com/aaalgo/kgraph
[11] https://yadi.sk/d/I_yaFVqchJmoc
[12] http://vis.uky.edu/ stewe/ukbench/
[13] http://nlp.stanford.edu/projects/glove/
[14] http://cloudcv.org/objdetect/

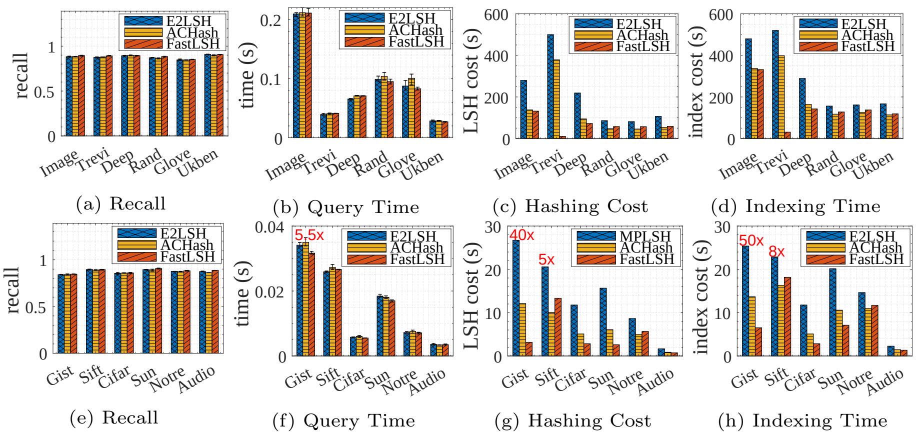


Fig. 2. Comparison of recall, average query time, LSH computation and index construction time for $m = 45/60$.

Results and Discussion: In this set of experiments, we are intended to show that FastLSH can reduce the index construction time significantly and achieve almost the same recall and query time as other LSH-based algorithms in the meantime.

We first compare the performance among E2LSH, ACHash and FastLSH for target recall around 0.9. The recall, average query time and LSH computation time for Gist, Sift, Cifar, Sun, Notre and Audio are illustrated in Fig. 1 (a), (b) and (c). It is easy to see that FastLSH and E2LSH achieve comparable query performance and answer accuracy as plotted in Figure 1 (a) and (b). Due to lack of theoretical guarantee, ACHash performs slightly worse than FastLSH and E2LSH in most cases w.r.t query efficiency. As shown in Fig. 1 (c), the LSH computation time of FastLSH is significantly superior to E2LSH and ACHash. For example, FastLSH obtains around 24x speedup over E2LSH and runs 11x times faster than ACHash on *Gist*. For ACHash, the fixed sampling ratio and overhead in Hadamard transform make it inferior to FastLSH. Note that because the query time, hashing cost and index construction time for different datasets varies greatly among datasets, we use 40x and etc. in the plots to indicate that the actual time is 40 times as much as the one shown in the plots.

We also plot the recall v.s. average query time curves by varying target recalls to obtain a complete picture of FastLSH in Fig. 3. The empirical results demonstrate that FastLSH performs almost the same in terms of answer accuracy, query efficiency and space occupation as E2LSH. Again, ACHash is slightly inferior to the others in most cases.

The end-to-end speedup in the index construction time is shown in Fig. 1 (d). Thanks to the significant reduction in hashing cost, the time spent in building the index decreases by up to a factor of 20. Besides hashing, the procedure of index construction consists of some other operations such as hash table initialization

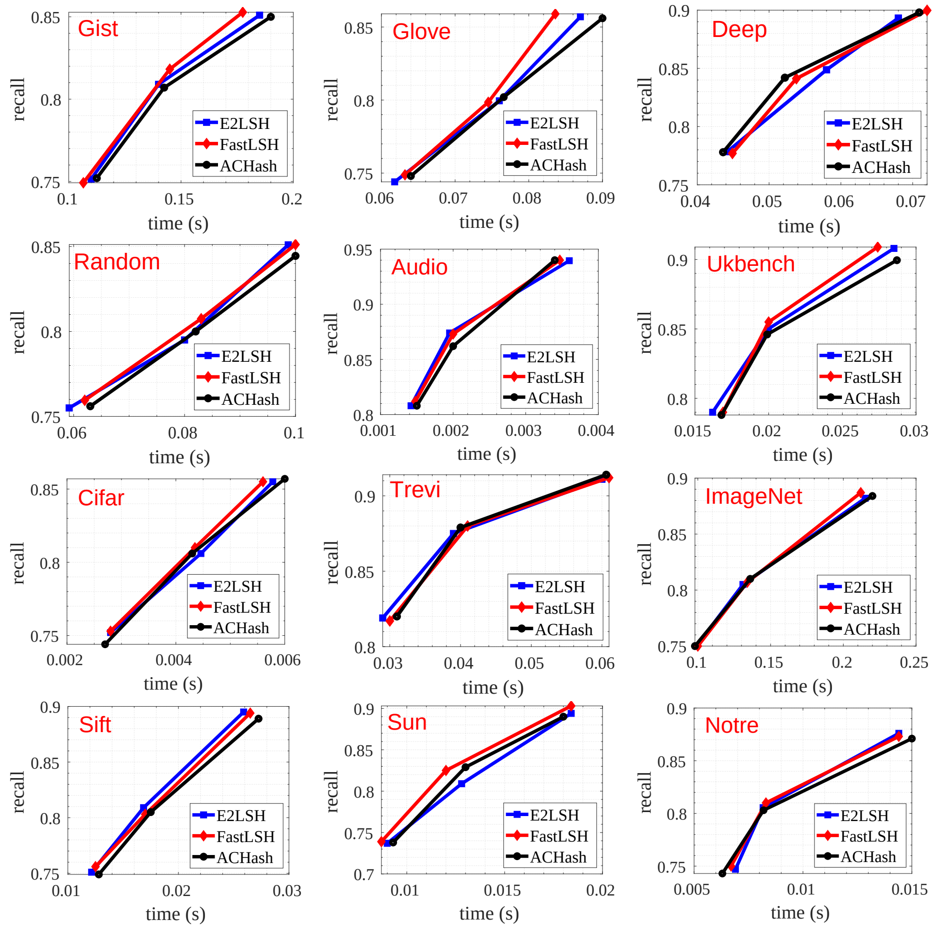


Fig. 3. Recall vs. Average query time

and linked list maintainance, which cannot be accelarated. Thus, the end-to-end latency in index construction decreases not as much as the hashing cost.

To evaluate the efficiency of FastLSH, we increased the value of m and presented the results in Fig. 2. For datasets with dimensions exceeding 500, we set $m = 60$, while for other datasets $m = 45$. Note that no changes were made to the parameters of E2LSH and ACHash. As shown in Figure 2, the query performance of FastLSH remains consistent with that in Fig. 1. However, both the LSH computation time and the end-to-end index construction time have significantly increased. This is primarily due to the higher computational cost of hashing as m increases. Therefore, we recommend setting $m = 30$ as the default sampling factor to mitigate the hashing cost and improve overall efficiency.

In sum, FastLSH is on par with E2LSH (with provable LSH property) in terms of answer accuracy and query efficiency and marginally superior to ACHash

Table 4. Parameters of E2LSH, FastLSH and ACHash

Datasets	recall	k	L	w	$\tilde{w}$	w'	Datasets	recall	k	L	w	$\tilde{w}$	w'
Cifar	lowest	10	40	0.0175	0.0041	0.138	Glove	lowest	8	79	0.84	0.455	3
	median	10	40	0.0185	0.0042	0.15		median	8	105	0.84	0.455	3
	highest	10	40	0.0195	0.00428	0.156		highest	9	150	0.91	0.495	3
Sun	lowest	8	45	4200	980	34000	Sift	lowest	8	45	36	17	145
	median	8	45	4550	1050	36000		median	8	45	39.5	19	160
	highest	8	45	5050	1120	39000		highest	8	45	42	20	165
Gist	lowest	10	105	2.5	0.435	0.45	Deep	lowest	8	60	0.0565	0.0189	0.45
	median	10	105	2.75	0.45	0.48		median	8	80	0.0565	0.0189	0.45
	highest	10	105	2.9	0.5	0.52		highest	8	105	0.0565	0.0189	0.45
Trevi	lowest	8	105	11.8	1	370	Random	lowest	10	108	0.5	0.275	1.85
	median	8	105	12.8	1.1	400		median	10	108	0.54	0.285	2
	highest	8	105	14	1.2	435		highest	10	108	0.6	0.295	2.2
Audio	lowest	6	25	7000	2680	47500	Ukbench	lowest	8	55	20.5	9.5	72
	median	6	25	7700	2900	50000		median	8	75	20.5	9.5	72
	highest	6	25	8700	3398	61000		highest	8	105	20.5	9.5	80
Notre	lowest	6	35	29	13.8	116	ImageNet	lowest	8	105	0.36	0.162	0.159
	median	6	35	31.5	15	125		median	8	105	0.4	0.175	0.17
	highest	6	35	35	16.6	134		highest	8	105	0.465	0.203	0.2

(without provable LSH property), while significantly reducing the cost of hashing.

3.4 FastLSH vs. Multi-Probe LSH in Nearest Neighbor Search

Baseline: MPLSH[15] [15] is a variant of vanilla LSH (E2LSH) designed for ANN search, which uses a heuristic probe sequence to search multiple buckets that contain the NNs of given query with high probability. Compared with E2LSH, MPLSH provides better space and time efficiency, i.e., it achieves the reduction in memory usage by using less hash functions ($k \times L$) and the same recall with lower query response time.

Metrics and Datasets: The same as those in evaluating FastLSH and E2LSH.

Parameter Settings: For fairness, we set the same parameters k, L and probe sequence for FastLSH and MPLSH to obtain the target recall. The parameters are presented in Table 5, where w is the bucket width of MPLSH, $\tilde{w}$ and w' are that of FastLSH and ACHash respectively; m is the number of sampled dimensions for FastLSH and the sampling ratio for ACHash is set to the defaulst 0.25 as in [7].

Results and Discussion: In this set of experiments, we achieve around 0.9 target recall for MPLSH, ACHash and FastLSH over all tested datasets. The actual recall, query time, hashing time and speedup in hashing are shown in

[15] https://lshkit.sourceforge.net/index.html

Table 5. Parameters of MPLSH, FastLSH and ACHash

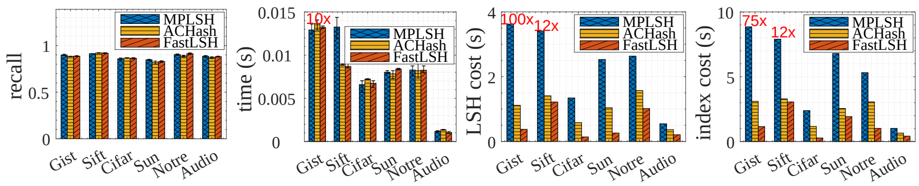
Datasets	recall	k	L	w	$\tilde{w}$	w'	m
Cifar	0.85	10	5	0.65	0.15	0.16	30
Sun	0.8	6	15	99370	19500	18000	30
Gist	0.9	15	25	3.9	1.07	0.355	90
Trevi	0.9	10	25	3500	1100	225	512
Audio	0.88	10	5	166000	80000	37000	60
Notre	0.88	6	10	420	110	93	30
Glove	0.88	15	25	16	8.1	5.2	50
Sift	0.9	15	25	1000	256	230	30
Deep	0.9	15	25	2.5	0.57	0.375	30
Random	0.9	15	25	8	4.5	4.3	50
Ukbench	0.87	6	10	220	60	52	30
ImageNet	0.9	10	25	0.65	0.28	0.18	75

Fig. 4. We can observe the same trend for MPLSH as with E2LSH, as shown in Fig. 1. Particularly, FastLSH achieves around the same recall for each dataset with the same or even much better query time than MPLSH and ACHash. For half of the datasets, $m = 30$ offers nice performance for FastLSH, whereas the others need much larger m to obtain the target recall. This might be attributed to the data-dependent nature of FastLSH and the probing heuristics used by MPLSH. The performance of ACHash is inferior to FastLSH due to lack of provable LSH property. The experimental results indicate that FastLSH performs well for all datasets and obtains up to 12x and 3x speedup in hashing over MPLSH and ACHash, respectively. The end-to-end index construction time is presented in Fig. 4 (d). Likewise, efficient hashing translates to reduced indexing construction time, leading to up to 10x speedup in building the index.

3.5 FastLSH Handles Sparse Data

Actually, FastLSH does work on sparse vector. The reasons and empirical evidence are as follows.

Recall that FastLSH consists of random sampling and random projection. FastLSH first performs m random sampling operations. The probability of selecting at least one non-zero (significant) element during the process of random sampling is $p_1 = 1 - (1-p)^m$, where p is the proportion of non-zero elements in the n-dimensional vector. And then FastLSH performs random projection. Since the hash code of FastLSH is a fingerprint concatenated from k hashes, the probability of selecting at least one non-zero (significant) element under k hashes is $p_r = 1 - (1-p_1)^k$. As a quick example, if $p = 0.01$, $m = 30$ and $k = 8$, then $p_r = 0.95$, meaning that the hash code contains the information of the significant elements with a high probability of 0.95. Therefore, FastLSH can effectively handle sparse data and is also well-suited for dense data, particularly in cases where many coordinate values are nearly identical.



(a) Recall (b) Query Time (c) Hashing Cost (d) Indexing Time

Fig. 4. Comparison of recall, average query time, LSH computation and index construction time.

We chose the MNIST[16] dataset to verify this claim. The dimension of MNIST is 784, with only around 2.6% non-zero elements. When we set $m = 30$ for FastLSH, the experimental results of FastLSH, E2LSH and ACHash under different k and L are shown in Table 6. From the table, it can be seen that FastLSH and E2LSH obtain comparable query accuracy, meaning it works very well for sparse datasets. While for ACHash, the fixed sampling ratio and overhead in Hadamard transform make it inferior to FastLSH. At the mean time, FastLSH can achieve up to 14.6 and 7.7 times faster LSH computation compared with E2LSH and ACHash, respectively. As a result, FastLSH can significantly reduce the end-to-end index construction time.

Note that for E2LSH package that we used [1], ultrahigh-dimensional vectors are assumed dense by default and all elements, regardless of whether it is zero or not, have to be computed. That is why such high speedup in hashing was achieved for FastLSH. In addition, for sparse ultrahigh-dimensional vectors, if the number of non-zero elements is still high after data preprocessing, FastLSH can be used to handle these non-zero elements; otherwise, we use the following Fast Johnson-Lindenstrauss (JL) transform.

Actually, for most practical applications, FastLSH is sufficient to handle sparse data as we have shown with the MINIST dataset. However, in the case of datasets being pathologically sparse, we can first apply the Fast JL transform to make the data dense similar to ACHash [7]. Here the Fast JL transform refers to the Hadamard transform, which is a unitary transformation, meaning that it preserves the nearest neighbor relationships of data points after transformation. Then FastLSH performs random sampling and random projection for the dense data. Although these operations are similar to ACHash, FastLSH retains the provable LSH property, which ACHash does not.

3.6 More Results for Comparison of Probability Density Curves

Lemma 4 and Fact 3 provide a principled approach to quantitatively analyze how m affects the difference between FastLSH and the classic LSH in terms of ϵ and λ, given the dataset characteristics (the variance in the squared distances of coordinates for a pair of data items). By using this analytical tool, it is easy

[16] http://yann.lecun.com/exdb/mnist/

Table 6. Results of E2LSH, FastLSH and ACHash over MNIST

Methods	Recall	Time (s)	Hashing (s)	Indexing (s)	k	L	w
E2LSH	0.844	0.00541	9.818	11.302	8	50	0.18
ACHash	0.853	0.00547	5.151	6.683	8	50	92.5
FastLSH	0.8545	0.00482	**0.671**	**2.301**	8	50	0.0325
E2LSH	0.799	0.00460	7.703	8.875	8	40	0.18
ACHash	0.809	0.00465	3.911	5.063	8	40	92.5
FastLSH	0.816	0.00439	**0.542**	**1.828**	8	40	0.0325
E2LSH	0.741	0.00352	6.365	7.307	8	30	0.18
ACHash	0.750	0.00360	3.328	4.339	8	30	92.5
FastLSH	0.748	0.00315	**0.417**	**1.369**	8	30	0.0325

for practitioners to determine the trade-off between hashing time (how much m is) and desired performance level (how close FastLSH is to the standard LSH).

To visualize the similarity, we plot $f_{\tilde{s}X}(t)$ for different m under the maximum and minimum σ, and the PDF of $\mathcal{N}(0, \frac{ms^2}{n})$ in Fig. 5 for all 12 datasets. The observations can be made from these figures: (1) the distribution of $\tilde{s}X$ matches very well with $\mathcal{N}(0, \frac{ms^2}{n})$ for small σ; (2) for large σ, $f_{\tilde{s}X}(t)$ differs only slightly from the PDF of $\mathcal{N}(0, \frac{ms^2}{n})$ for all m, indicating that s is the dominating factor in $p(s, \sigma)$; (3) greater m results in higher similarity between $f_{\tilde{s}X}(t)$ and $\mathcal{N}(0, \frac{ms^2}{n})$, implying that FastLSH can always achieve almost the same performance as E2LSH by choosing m appropriately.

3.7 Comparison of ρ Curves

To further validate the LSH property of FastLSH, we compare the important parameter ρ for FastLSH and E2LSH in the case of $m = 30$. ρ is defined as the function of the approximation ratio c, i.e., $\rho(c) = log(1/p(s_1))/log(1/p(s_2))$, where $s_1 = 1$ and $s_2 = c$. Note that ρ affects both the space and time efficiency of LSH algorithms. For c in the range $[1, 20]$ (with increments of 0.1), we calculate ρ using *Matlab*, where the minimal and maximal σ are collected for different c (s). Plots of $\rho(c)$ under different bucket widths for 12 datesets are given in Fig. 6 and Fig. 7. Clearly, the $\rho(c)$ curve of FastLSH matches very well with that of E2LSH, verifying that FastLSH maintains almost the same LSH property with E2LSH even when m is relatively small.

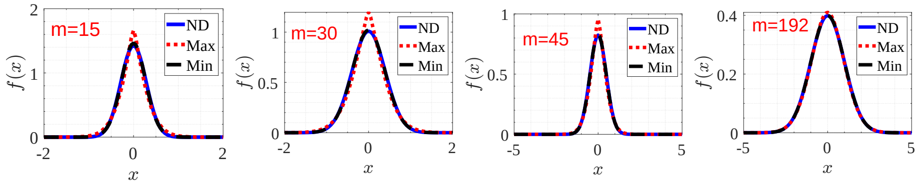
3.8 Effects of ϵ and λ

We list the values of ϵ and λ for different m over 12 datasets in Table 7, where ϵ and λ are calculated using the maximum, mean and minimum σ, respectively. Recall that smaller ϵ and λ are, FastLSH is more similar to E2LSH. As shown in Table 7, ϵ and λ decrease as m increases. Take *Trevi* as an example, ϵ is equal to 0 and λ is very tiny (0.0001-0.000729), manifesting the equivalence between $f_{\tilde{s}X}(t)$ and the PDF of $\mathcal{N}(0, \frac{ms^2}{n})$.

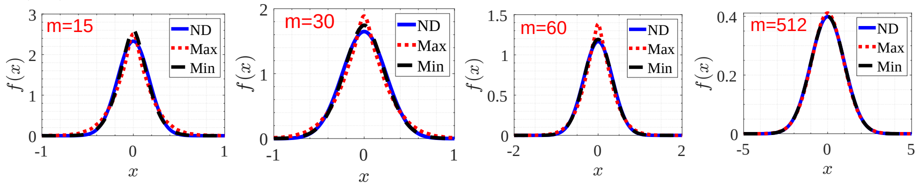
References

1. Alexandr Andoni. E2lsh-0.1 user manual. http://web.mit.edu/andoni/www/LSH/index.html, 2005.
2. Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*, 51(1):117–122, 2008.
3. Yoram Bachrach, Yehuda Finkelstein, Ran Gilad-Bachrach, Liran Katzir, Noam Koenigstein, Nir Nice, and Ulrich Paquet. Speeding up the xbox recommender system using a euclidean transformation for inner-product spaces. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 257–264, 2014.
4. Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.
5. K. Bhatia, K. Dahiya, H. Jain, P. Kar, A. Mittal, Y. Prabhu, and M. Varma. The extreme classification repository: Multi-label datasets and code, 2016.
6. Beidi Chen, Tharun Medini, James Farwell, Charlie Tai, Anshumali Shrivastava, et al. Slide: In defense of smart algorithms over hardware acceleration for large-scale deep learning systems. *Proceedings of Machine Learning and Systems*, 2:291–306, 2020.
7. Anirban Dasgupta, Ravi Kumar, and Tamás Sarlós. Fast locality-sensitive hashing. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1073–1081, 2011.
8. Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262, 2004.
9. Paul Gerhard Hoel, Sidney C Port, and Charles Joel Stone. *Introduction to probability theory*. Houghtion Mifflin, 1971.
10. Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, 1998.
11. Norio Katayama and Shin'ichi Satoh. The sr-tree: An index structure for high-dimensional nearest neighbor queries. *ACM Sigmod Record*, 26(2):369–380, 1997.
12. Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 614–623, 1998.
13. Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering*, 32(8):1475–1488, 2019.
14. Chen Luo and Anshumali Shrivastava. Arrays of (locality-sensitive) count estimators (ace) anomaly detection on the edge. In *Proceedings of the 2018 World Wide Web Conference*, pages 1439–1448, 2018.
15. Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. Multi-probe lsh: efficient indexing for high-dimensional similarity search. In *Proceedings of the 33rd international conference on Very large data bases*, pages 950–961, 2007.
16. Anshul Mittal, Kunal Dahiya, Shreya Malani, Janani Ramaswamy, Seba Kuruvilla, Jitendra Ajmera, Keng-hao Chang, Sumeet Agarwal, Purushottam Kar, and Manik Varma. Multi-modal extreme classification. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12393–12402, 2022.

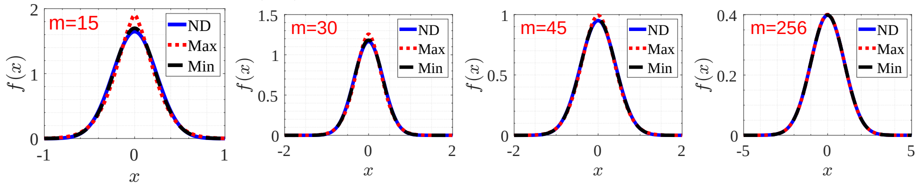
17. Anshul Mittal, Noveen Sachdeva, Sheshansh Agrawal, Sumeet Agarwal, Purushottam Kar, and Manik Varma. Eclare: Extreme classification with label graph correlations. In *Proceedings of the Web Conference 2021*, pages 3721–3732, 2021.
18. Hanan Samet. *Foundations of multidimensional and metric data structures*. Morgan Kaufmann, 2006.
19. Anshumali Shrivastava and Ping Li. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). 27:2321–2329, 2014.



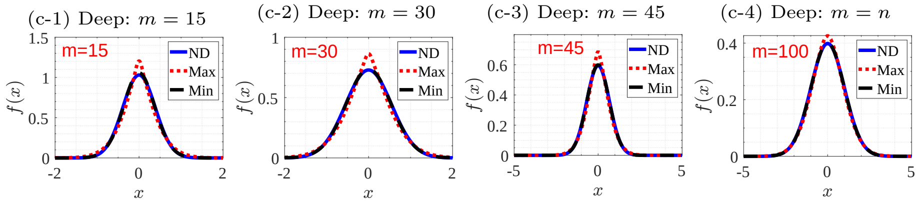
(a-1) Audio: $m = 15$ (a-2) Audio: $m = 30$ (a-3) Audio: $m = 45$ (a-4) Audio: $m = n$



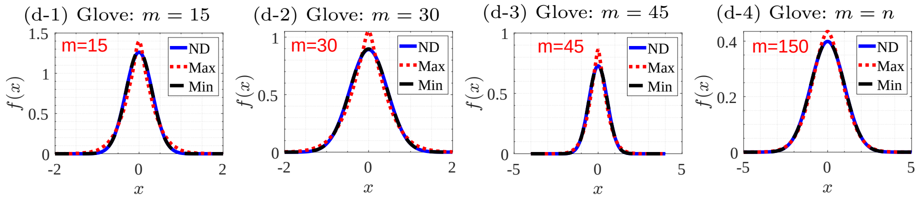
(b-1) Cifar: $m = 15$ (b-2) Cifar: $m = 30$ (b-3) Cifar: $m = 60$ (b-4) Cifar: $m = n$



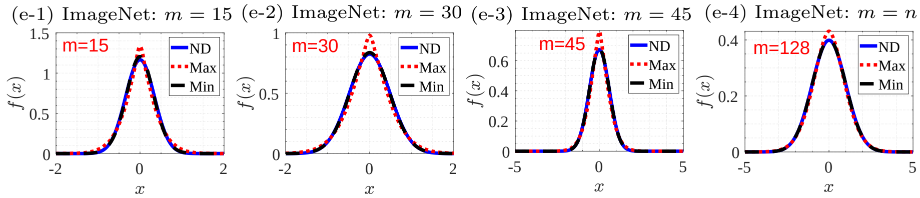
(c-1) Deep: $m = 15$ (c-2) Deep: $m = 30$ (c-3) Deep: $m = 45$ (c-4) Deep: $m = n$



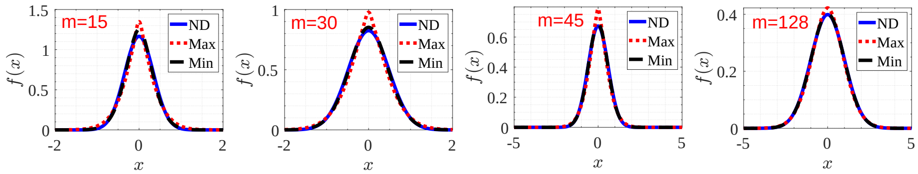
(d-1) Glove: $m = 15$ (d-2) Glove: $m = 30$ (d-3) Glove: $m = 45$ (d-4) Glove: $m = n$



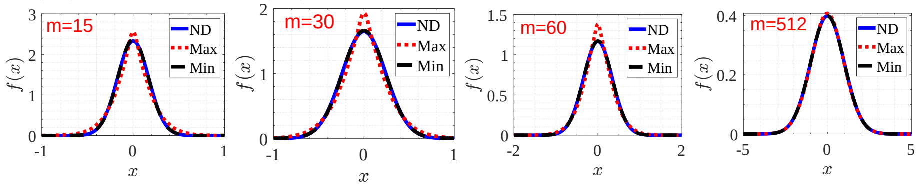
(e-1) ImageNet: $m = 15$ (e-2) ImageNet: $m = 30$ (e-3) ImageNet: $m = 45$ (e-4) ImageNet: $m = n$



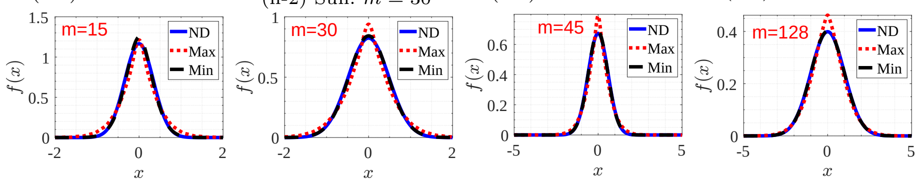
(f-1) Notre: $m = 15$ (f-2) Notre: $m = 30$ (f-3) Notre: $m = 45$ (f-4) Notre: $m = n$



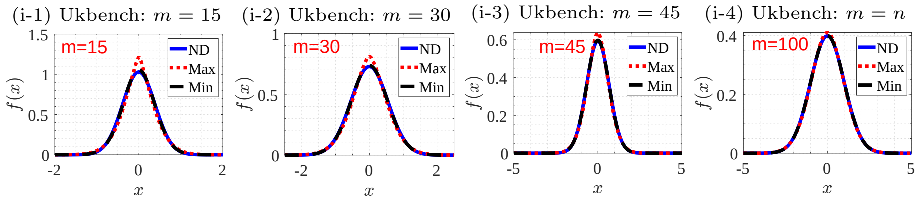
(g-1) Sift: $m = 15$ (g-2) Sift: $m = 30$ (g-3) Sift: $m = 45$ (g-4) Sift: $m = n$



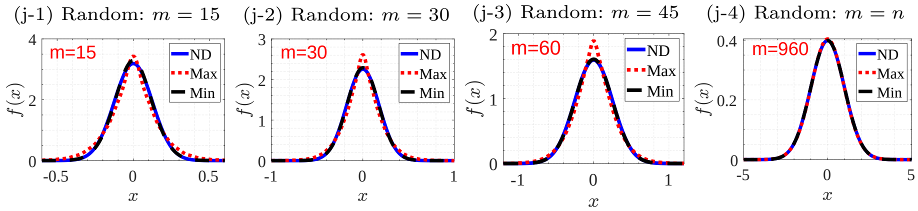
(h-1) Sun: $m = 15$ (h-2) Sun: $m = 30$ (h-3) Sun: $m = 60$ (h-4) Sun: $m = n$



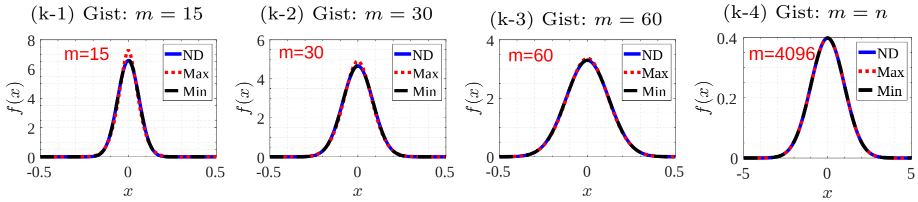
(i-1) Ukbench: $m = 15$ (i-2) Ukbench: $m = 30$ (i-3) Ukbench: $m = 45$ (i-4) Ukbench: $m = n$



(j-1) Random: $m = 15$ (j-2) Random: $m = 30$ (j-3) Random: $m = 45$ (j-4) Random: $m = n$

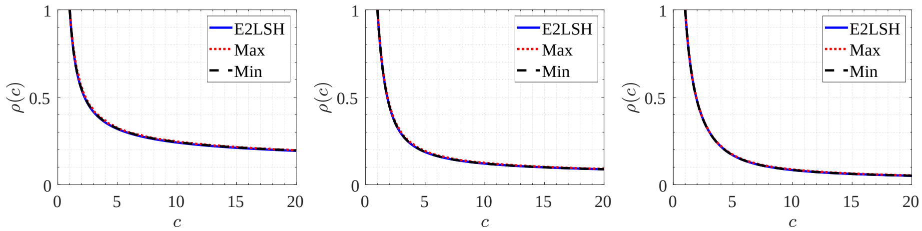


(k-1) Gist: $m = 15$ (k-2) Gist: $m = 30$ (k-3) Gist: $m = 60$ (k-4) Gist: $m = n$

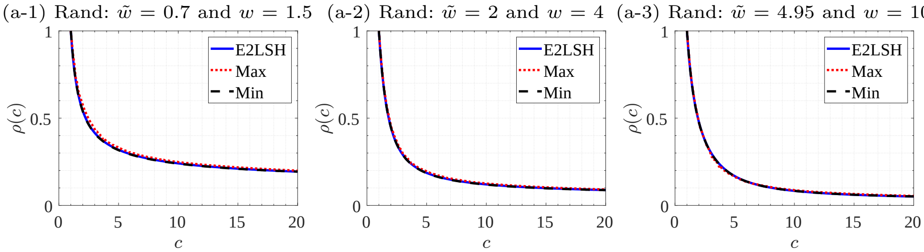


(l-1) Trevi: $m = 15$ (l-2) Trevi: $m = 30$ (l-3) Trevi: $m = 60$ (l-4) Trevi: $m = n$

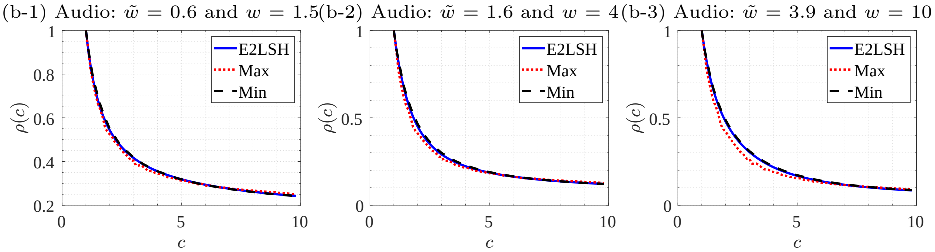
Fig. 5. Comparison of probability density curves of $\mathcal{N}(0, \frac{ms^2}{n})$ (ND) and $\tilde{s}X$ under different m.



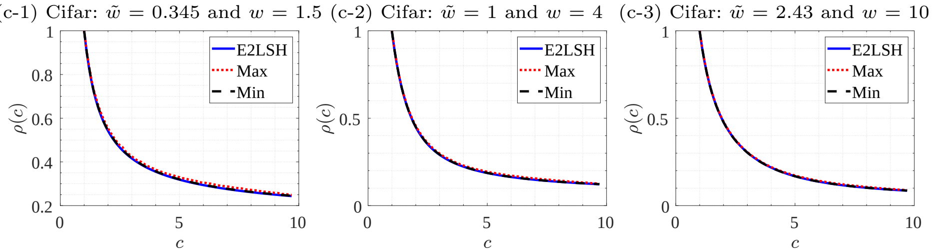
(a-1) Rand: $\tilde{w} = 0.7$ and $w = 1.5$ (a-2) Rand: $\tilde{w} = 2$ and $w = 4$ (a-3) Rand: $\tilde{w} = 4.95$ and $w = 10$



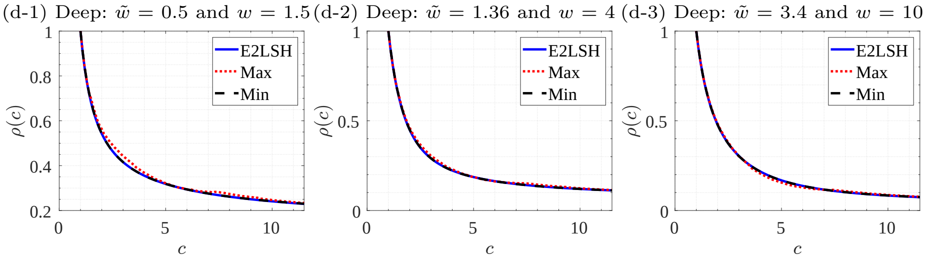
(b-1) Audio: $\tilde{w} = 0.6$ and $w = 1.5$ (b-2) Audio: $\tilde{w} = 1.6$ and $w = 4$ (b-3) Audio: $\tilde{w} = 3.9$ and $w = 10$



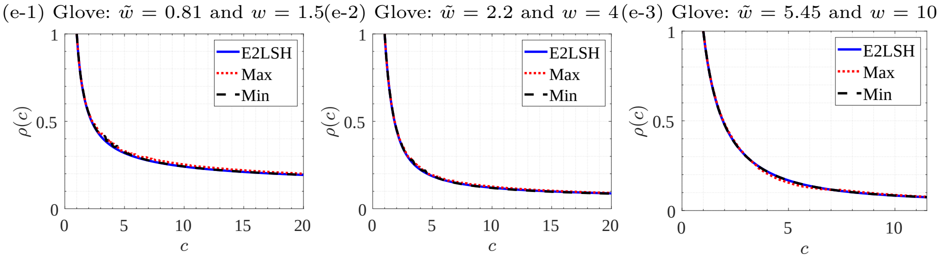
(c-1) Cifar: $\tilde{w} = 0.345$ and $w = 1.5$ (c-2) Cifar: $\tilde{w} = 1$ and $w = 4$ (c-3) Cifar: $\tilde{w} = 2.43$ and $w = 10$



(d-1) Deep: $\tilde{w} = 0.5$ and $w = 1.5$ (d-2) Deep: $\tilde{w} = 1.36$ and $w = 4$ (d-3) Deep: $\tilde{w} = 3.4$ and $w = 10$

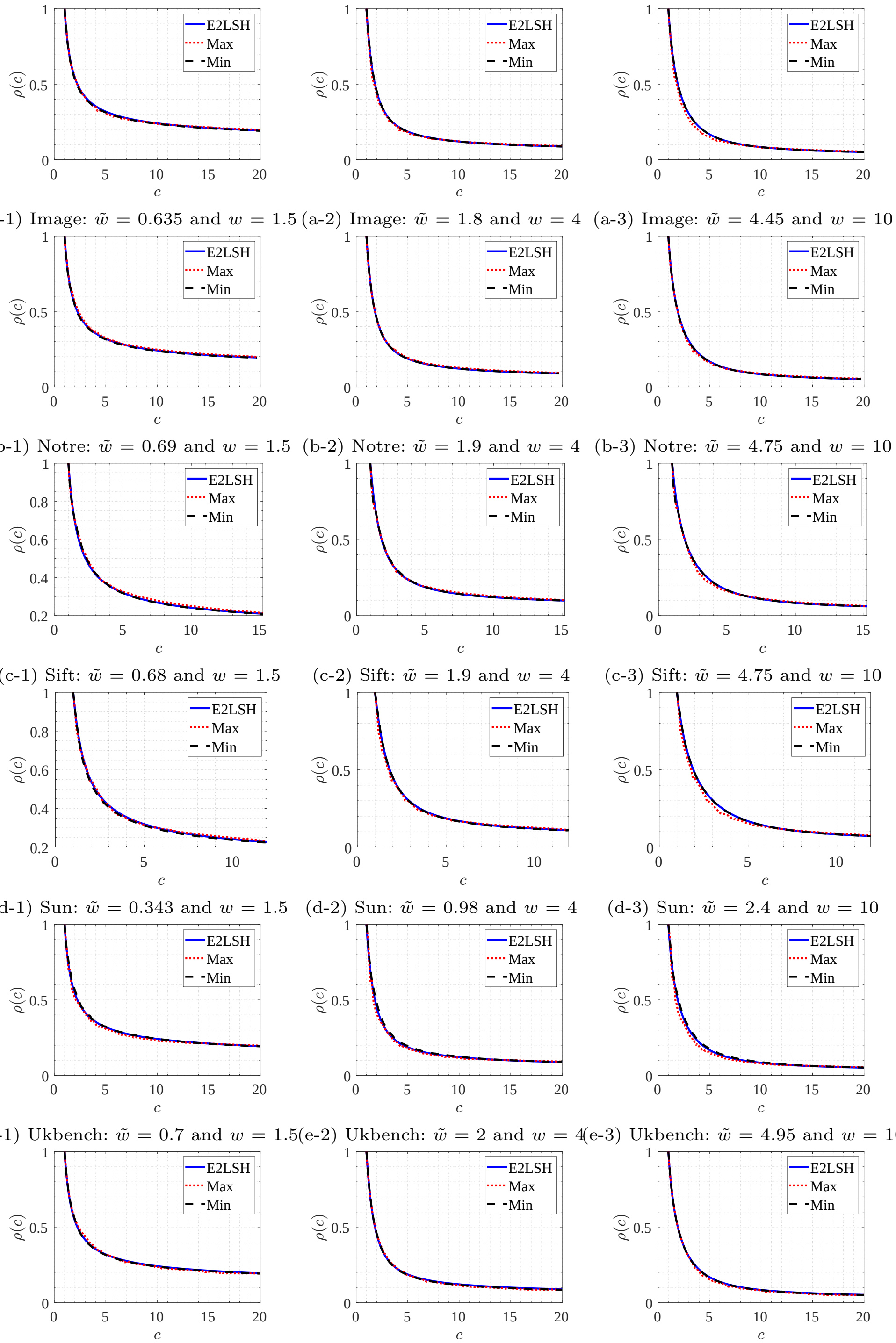


(e-1) Glove: $\tilde{w} = 0.81$ and $w = 1.5$ (e-2) Glove: $\tilde{w} = 2.2$ and $w = 4$ (e-3) Glove: $\tilde{w} = 5.45$ and $w = 10$



(f-1) Trevi: $\tilde{w} = 0.13$ and $w = 1.5$ (f-2) Trevi: $\tilde{w} = 0.35$ and $w = 4$ (f-3) Trevi: $\tilde{w} = 0.86$ and $w = 10$

Fig. 6. ρ curves under different bucket widths over datasets *Random*, *Audio* and *Cifar*, *Deep*, *Glove* and *Trevi*.



(a-1) Image: $\tilde{w} = 0.635$ and $w = 1.5$ (a-2) Image: $\tilde{w} = 1.8$ and $w = 4$ (a-3) Image: $\tilde{w} = 4.45$ and $w = 10$

(b-1) Notre: $\tilde{w} = 0.69$ and $w = 1.5$ (b-2) Notre: $\tilde{w} = 1.9$ and $w = 4$ (b-3) Notre: $\tilde{w} = 4.75$ and $w = 10$

(c-1) Sift: $\tilde{w} = 0.68$ and $w = 1.5$ (c-2) Sift: $\tilde{w} = 1.9$ and $w = 4$ (c-3) Sift: $\tilde{w} = 4.75$ and $w = 10$

(d-1) Sun: $\tilde{w} = 0.343$ and $w = 1.5$ (d-2) Sun: $\tilde{w} = 0.98$ and $w = 4$ (d-3) Sun: $\tilde{w} = 2.4$ and $w = 10$

(e-1) Ukbench: $\tilde{w} = 0.7$ and $w = 1.5$ (e-2) Ukbench: $\tilde{w} = 2$ and $w = 4$ (e-3) Ukbench: $\tilde{w} = 4.95$ and $w = 10$

(f-1) Gist: $\tilde{w} = 0.255$ and $w = 1.5$ (f-2) Gist: $\tilde{w} = 0.73$ and $w = 4$ (f-3) Gist: $\tilde{w} = 1.8$ and $w = 10$

Fig. 7. ρ curves under different bucket widths over datasets *ImageNet*, *Notre*, *Sift*, *Sun*, *Ukbench* and *Gist*.

Table 7. ϵ and λ for different m

Datasets		$m = 15$		$m = 30$		$m = 45/\mathbf{60}$		$m = n$	
		ϵ	λ	ϵ	λ	ϵ	λ	ϵ	λ
Cifar	max	0.567575	2.527575	0.287600	1.287600	**0.114125**	**0.618225**	0.000064	0.067664
	mean	0.102716	0.575372	0.027539	0.277239	**0.001994**	**0.120123**	0	0.014617
	min	0.019616	0.239671	0.001702	0.116014	**0.000011**	**0.055001**	0	0.006691
Sun	max	0.502360	2.218460	0.236626	1.084867	**0.095099**	**0.546683**	0.000006	0.051535
	mean	0.022783	0.255107	0.001849	0.118130	**0.000018**	**0.058099**	0	0.006724
	min	0	0.040401	0	0.020736	**0**	**0.010000**	0	0.001156
Gist	max	0.633640	2.853740	0.274502	1.234902	**0.129940**	**0.677540**	0	0.032400
	mean	0.042122	0.340238	0.005183	0.152639	**0.000133**	**0.074662**	0	0.004624
	min	0.000064	0.067664	0	0.038025	**0**	**0.016926**	0	0.001089
Trevi	max	0.013420	0.207020	0.001105	0.106081	**0.000003**	**0.049287**	0	0.000729
	mean	0.000011	0.055236	0	0.029241	**0**	**0.013924**	0	0.000196
	min	0	0.015876	0	0.008100	**0**	**0.003969**	0	0.000100
Audio	max	0.268000	1.208900	0.099004	0.561404	0.043520	0.346020	0.000033	0.062533
	mean	0.033344	0.303017	0.003637	0.138767	0.000480	0.091021	0	0.020967
	min	0.000022	0.059705	0	0.030765	0	0.021874	0	0.005329
Notre	max	0.341109	1.507509	0.143598	0.728823	0.074226	0.467355	0.004017	0.142401
	mean	0.022783	0.255107	0.001902	0.118866	0.000172	0.077456	0	0.027225
	min	0.000101	0.071925	0	0.036481	0	0.023716	0	0.008281
Glove	max	0.261530	1.183130	0.094131	0.543031	0.040065	0.331665	0.002362	0.124862
	mean	0.003190	0.134234	0.000047	0.065072	0.000001	0.043265	0	0.019853
	min	0.000025	0.060541	0	0.029929	0	0.020335	0	0.009409
Sift	max	0.294194	1.314294	0.098513	0.559554	0.057014	0.400410	0.001296	0.109537
	mean	0.054265	0.389506	0.007185	0.167986	0.001509	0.113065	0	0.036864
	min	0.003755	0.139916	0.000107	0.072468	0.000001	0.045370	0	0.016129
Deep	max	0.036744	0.317644	0.003841	0.140741	0.000463	0.090463	0	0.014400
	mean	0.003047	0.132719	0.000044	0.064560	0.000001	0.043306	0	0.007569
	min	0.000199	0.079160	0	0.039204	0	0.026374	0	0.004638
Random	max	0.077344	0.479300	0.015195	0.216796	0.003505	0.137461	0.000041	0.064050
	mean	0.002824	0.130273	0.000038	0.063542	0.000001	0.042437	0	0.019044
	min	0.000024	0.060049	0	0.029929	0.	0.019881	0	0.008649
Ukbench	max	0.692955	3.157855	0.361581	1.593681	0.217238	1.009338	0.039727	0.330248
	mean	0.139183	0.712232	0.034502	0.308031	0.012672	0.202768	0.000056	0.066620
	min	0.002895	0.131059	0.000041	0.064050	0.000001	0.042437	0	0.015376
ImageNet	max	0.466568	2.054168	0.217238	1.009338	0.119323	0.637723	0.004773	0.149173
	mean	0.019125	0.237214	0.001336	0.110236	0.000107	0.072468	0	0.022201
	min	0	0.033489	0	0.016641	0	0.011025	0	0.003481