Arrays of (locality-sensitive) Count Estimators (ACE): Anomaly Detection on the Edge

Chen Luo
Rice University
Houston, Texas
cl67@rice.edu

Anshumali Shrivastava
Rice University
Houston, Texas
anshumali@rice.edu

ABSTRACT

Anomaly detection is one of the frequent and important subroutines deployed in large-scale data processing applications. Even being a well-studied topic, existing techniques for unsupervised anomaly detection require storing significant amounts of data, which is prohibitive from memory, latency and privacy perspectives, especially for small mobile devices which has ultra-low memory budget and limited computational power. In this paper, we propose ACE (Arrays of (locality-sensitive) Count Estimators) algorithm that can be 60x faster than most state-of-the-art unsupervised anomaly detection algorithms. In addition, ACE has appealing privacy properties. Our experiments show that ACE algorithm has significantly smaller memory footprints ($< 4MB$ in our experiments) which can exploit Level 3 cache of any modern processor. At the core of the ACE algorithm, there is a novel statistical estimator which is derived from the sampling view of Locality Sensitive Hashing (LSH). This view is significantly different and efficient than the widely popular view of LSH for near-neighbor search. We show the superiority of ACE algorithm over 11 popular baselines on 3 benchmark datasets, including the KDD-Cup99 data which is the largest available public benchmark comprising of more than half a million entries with ground truth anomaly labels.

1 INTRODUCTION

The problem of anomaly (or outlier) detections is the task of identifying instances (or patterns) in data that do not conform to the expected behavior [9]. These non-conforming examples are popularly referred to as anomalies, or outliers, sometimes interchangeably.

Anomaly detection can be either supervised [28] or unsupervised [20]. Supervised anomaly detection leverages machine learning algorithms, such as classification, over datasets labeled as anomalous or non-anomalous. However, there are three major issues with supervised anomaly detection algorithms: 1) In most applications, label information about anomalies is not available; 2) Anomalies are rare, and hence there is a huge class imbalance, and 3) Supervised algorithms need to be re-trained for drifting data distributions with new label information. Drifting data distribution is quite common in big-data systems, where supervised re-training is prohibitive. Therefore, we are interested in unsupervised anomaly detection which does not require any label information, and which can automatically deal with changes in data distributions over time. We briefly describe some of the modern challenges for unsupervised anomaly detection that we will address in this work.

Challenge 1: High-Speed Drifting Data: Many applications demand fast-response and real-time inference from dynamic and drifting high volumes of sensor data over time. Most anomaly detection applications, for example over the web-network servers, require dealing with unprecedented amounts of data in a fraction of seconds. The data distribution is constantly changing, and it is often bursty [25, 29]. Detecting anomaly events in real-time, such as DDoS (Distributed Denial of Service) attacks, network failures, *etc.*, is highly beneficial in monitoring network performance.

Challenge 2: Ultra-Low Memory Budget: In many high-speed streaming applications, such as High Energy Physics (HEP) and network servers, any algorithm requiring to store and process a significant fraction of data is prohibitive. Another critical pushing need for ultra-low memory algorithm is anomaly detection on mobile phones or smart sensors. Algorithms which require significant resources are prohibitive for low-resource platforms.

Challenge 3: Anomaly Detection on the Edge (Mobile Devices): Anomaly detection on portable devices or mobile devices often requires dealing with high-speed drifting data, low-memory, and in addition ultra-low power. A modern smartphone usually only have a relatively small memory capacity (1,2 Gigabytes) and limited computational power. Battery life is a significant concern, and transmitting data to cloud for analysis has privacy risks as well as are not sustainable due to their energy demands. With the accelerated adoption of 4G (or 5G) technologies including WiMAX and LTE, cellular devices will become the primary means of broadband Internet access for many users. According to the report from Cisco, Global mobile data traffic reached 7.2 exabytes per month at the end of 2016 [14]. Thus, the traffic data that is monitored, or generated, by the mobile devices are extremely high-speed and enormous, and it is hopeless to rely on anomaly detection methods which require consulting a significant fraction of the data. Unfortunately, most unsupervised anomaly detection techniques are near-neighbor based and require querying, and hence the prohibitive storage of the historical data.

Challenge 4: Privacy: As the IoT becomes more widespread, consumers must demand better security and privacy protections that do not leave them vulnerable to corporate surveillance and data breaches. Thus, storing a significant fraction of data for finding anomalous behavior is prohibitive. Privacy-preserving anomaly detection is a challenging problem in itself [7].

The significance and the impact of the above challenges have put high-speed data mining among the top-10 big-data challenges [46], which will also be the focus of the present work.

Popular Approaches for Unsupervised Anomaly Detection: There are numerous methods for unsupervised anomaly detection in literature. We review and compare with 11 of these popular methods in our experiments. Unsupervised anomaly detection can be broadly categorized into two categories: 1) Near-Neighbor (NN) Based and 2) Aggregate Statistics (or score) based. NN based approaches typically define the outlier score of a point q based on the difference between q's own behavior and the behavior of q's near-neighbors. The first category is the most common category.

Aggregate statistics based methods, on the other hand, define the outlier score of a point q based on the expected behavior of a global function $S(q, \mathcal{D})$ of the data $\mathcal{D}$, relative to q. A notable method among them is ABOD (Angle-based outlier detection) [32]. ABOD computes the variance of the angle formed by different pairs of points, in the dataset, incident on the point of interest q. It is expected that the outliers will have a small variance [32].

There are several implementations of existing anomaly detection algorithms. A notable among them is the *ELKI* package [1] which is currently one of the most efficient and popular packages for outlier detection because of highly optimized implementations.

Both categories of anomaly detection algorithms require storing the complete dataset to either compute near-neighbor or the desired statistics from the data. The bottleneck computational cost is at least one pass over the data to either calculate the near-neighbor or the statistics. Thus, these methods have poor computational and memory requirements. Furthermore, change in the distribution of data requires storing and processing a larger set of observations.

Sampling and Fast Near-Neighbors: To work around the computational requirements it is natural to resort to fast alternatives [8]. There are plenty of techniques which exploits efficient near-neighbor capabilities to speed up NN. However, they still require storing the data in the memory. Even with the computational speedups, the methodologies are still slow for ultra-high speed data mining, as an accurate near-neighbor search over large data is costly.

Relying on random sampling and projections of the data to estimate the aggregate statistics efficiently is not new [9]. For example, recently, [32] showed that using smart random sampling and hashing algorithms, we can speed up the anomaly detection and also reduce the memory requirement. Instead of storing all the data points, we only need few random samples and their quantized projections. They proposed FastVOA which uses a modified ABOD statistics that can be estimated in near-constat time and is as good as ABOD for anomaly detection.

However, these approximation methods still require storing a significant number of data samples, which makes the algorithm slow and prohibitive from privacy perspective. FastVOA involves various computation of medians and other costly statistics. Our experiments show that the sampling based FastVOA approach is significantly slower than fast NN based alternatives.

There is a third category of anomaly detection algorithms over a sliding window in data streams [44]. The notions of anomalies in these algorithms are confined to a given fixed-size window over time. Not surprisingly, if the size of sliding window is increased to take into account large amounts of data, we again observe the same memory and latency issues. The focus of this paper is on unsupervised outlier detection, where the notion of anomaly is with respect to the compete data and not constrained to a limited sliding window.

Our Contributions: We propose a family of statistics which provides a “sweet" spot between the ability to discriminate anomalies and the resource efficiency. These special statistics, due to their form, can be efficiently computed in ultra-low memory, and they do not require storing even a single data sample. Furthermore, any updates to the data can be incorporated on the fly making our proposal ideal for high-speed data applications. The proposed algorithm, in addition, has strong privacy properties making it ideal for IoT (Internet of Things) setting.

Our proposed family of statistics are derived from the collision probability of locality sensitive hashing (LSH) functions. We show that these classes of statistics have strong discriminative property for identifying outliers and most importantly, it can be accurately estimated using Arrays of Count Estimators (ACE), a novel and tiny LSH based data structure. Designing these estimators requires using the sampling view of LSH rather than the widely popular near-neighbor search view. To the best of our knowledge, this is the first use of LSH counts as unbiased estimators of outlierness.

We demonstrate, empirically and theoretically, that the proposed LSH based count estimators are significantly more accurate than random sampling approaches. Our ACE algorithm only requires computing few locality sensitive hashes of the data and a small set of count array lookups to estimate the proposed statistics sharply. Our approach does not require even a single distance computation. The theory and the class of estimators presented in the paper, could of independent interest in itself.

We demonstrate rigorous experimental evidence on three public outlier detection benchmarks including the largest publicly available benchmark dataset KDD-cup99 HTTP dataset having more than half a million labeled instances. Empirically, our algorithm only requires around $4MB$ of memory and near-constant amount of computations, for all the three benchmark datasets. Thus, we can exploit fast L3 caches (Level 3 caches), which can be significantly faster than dealing with main memory.

We provide a comparison of our algorithm with 11 different methodologies, which include some of the fastest and most popular anomaly detection algorithms. Our experiment shows that we are around at least 60x faster than of the best performing competitor on the largest benchmark KDD-cup99 HTTP dataset. This disruptive speedup is not surprising given the computational simplicity of our algorithm and ultra-low memory print.

2 BACKGROUND: LOCALITY SENSITIVE HASHING

Locality-Sensitive Hashing (LSH) [22] is a popular technique for efficient approximate nearest-neighbor search. LSH is a family of functions, such that a function uniformly sampled from this hash family has the property that, under the hash mapping, similar points have a high probability of having the same hash value. More precisely, consider $\mathcal{H}$ a family of hash functions mapping $\mathbb{R}^D$ to a discrete set $[0, R-1]$.

Definition 2.1. **Locality Sensitive Hashing (LSH) Family** A family $\mathcal{H}$ is called (S_0, cS_0, u_1, u_2)-sensitive if for any two points $x, y \in \mathbb{R}^d$ and h chosen uniformly from $\mathcal{H}$ satisfies the following:

- if $Sim(x, y) \geq S_0$ then $Pr_{\mathcal{H}}(h(x) = h(y)) \geq u_1$

- if $Sim(x, y) \leq cS_0$ then $Pr_{\mathcal{H}}(h(x) = h(y)) \leq u_2$

A collision occurs when the hash values for two data vectors are equal, meaning that $h(x) = h(y)$.

LSH is a very well studied topic in computer science theory and database literature. There are a number of well-known LSH families in the literature. Please refer [17] for details. The most popular one is Signed Random Projections [11].

Signed Random Projections(SRP) is an LSH for the cosine similarity measure, which originates from the concept of **randomized rounding (SRP)** [11, 18, 30]. Given a vector x, SRP utilizes a random w vector with each component generated from i.i.d. normal, *i.e.*, $w_i \sim N(0, 1)$, and only stores the sign of the projection. Formally SRP family is given by

$$h_w(x) = sign(w^T x). \tag{1}$$

It was shown in the seminal work [18] that collision under SRP satisfies the following equation:

$$Pr_w(h_w(x) = h_w(y)) = 1 - \frac{\theta}{\pi}, \tag{2}$$

where $\theta = cos^{-1}\left(\frac{x^T y}{||x||_2||y||_2}\right)$. $\frac{x^T y}{||x||_2||y||_2}$, is the cosine similarity.

If we generate K independent SRP bits, by sampling w independently k times, and use the generated K-bit number as the new hash function H, then the new collision probability is

$$Pr(H(x) = H(y)) = (1 - \frac{\theta}{\pi})^K \tag{3}$$

by the simple multiplicative law of probability. We will be using this observation heavily in our work.

Over the last decade, there has been a significant advancement in reducing the amortized computational and memory requirements for computing several LSH signatures of the data vector. For random projections based LSH, of which signed random projection is a special case, we can calculate m LSH hashes of the data vector, with dimensions d, in time $O(d \log d + m)$, a significant improvement over $O(dm)$. This speedup is possible due to the theory of Fast-Johnson-Lindenstrauss transformation [3, 15]. On the orthogonal side, even better speedup of $O(d + m)$ has been obtained with permutation-based LSH, such as minwise hashing, using ideas of densification [37–40]. These drastic reductions in hashing time have been instrumental in making LSH based algorithms more appealing and practical.

3 OUR PROPOSAL

Denote the dataset with $\mathcal{D} = \{x_i | i \in [1, n]\}$, where n is the number of data points in $\mathcal{D}$. By definition, outliers are significantly separated from an average data point. Therefore, any reasonable statistics of x_i with respect to all other $x_j \in \mathcal{D}$ will deviate significantly for outliers compared to a normal data point. Even an average distance of x_i with all other elements of $\mathcal{D}$ is a reasonably good statistics [33]. However, as noted before, computing these statistics requires storing the complete data $\mathcal{D}$. In general, calculating every single $S(x_i, \mathcal{D})$ requires one complete pass over the dataset $\mathcal{D}$. Besides, our experiments show that alternative estimations based on random sampling and random projections still lead to significant computational overheads.

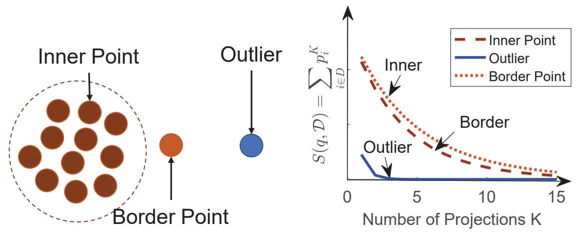

Figure 1: Discriminative power of $S(q, \mathcal{D})$: We can see from the figure that the value of $S(q, \mathcal{D})$ for an Outlier is significantly lower (different) compared to that of non-outliers.

We instead focus on classes of scoring functions $S(., .)$ over the dataset that can be estimated efficiently using a tiny (memory efficient) data structure that can easily fit fast processor cache. Furthermore, we also want to update the data structure on the fly. In particular, any change in data from $\mathcal{D}$ to $\mathcal{D}'$ requires no change, and the estimates get dynamically adjusted.

We show that a class of scoring functions of the following form have the aforementioned property:

$$S(q, \mathcal{D}) = \sum_{x_i \in \mathcal{D}} p(q, x_i)^K, \tag{4}$$

where p is the collision probability of any LSH family and $K \geq 1$ is an integer.

The analysis of this paper extends naturally to any LSH scheme. For this work, we will focus on the popular signed random projections (SRP) as the LSH because of its simplicity. Furthermore, advances in fast SRP have lead to some very lightweight hashing variants. With SRP, the collision probability $p(q, x_i)$ is given by:

$$p(q, x_i) = 1 - \frac{1}{\pi} \cos^{-1}\left(\frac{q^T x_i}{\|q\| \, \|x_i\|}\right)$$

which will also be the value of $p(q, x_i)$ for the rest of the paper.

3.1 Can it Discriminate Outliers?

To demonstrate the discriminative power of the scoring function in Equation 4, we do a simulation experiment similar to the one performed in [32]. We first generate a simple dataset with an outlier point. Figure 1 (left) shows the snapshot of the data. There are two sets of data points. The outlier and the general data points. For the general data points, in addition, we make a distinction between the border points and inner points as illustrated in the figure.

In Figure 1 (right), we plot the value of our proposed statistics $\frac{1}{n}S(q, \mathcal{D})$, given by Equation 4, for different sets of data points as a function of K. We can see from the figure the value of $\frac{1}{n}S(q, \mathcal{D})$ for an outlier point is near zero. In particular, it is significantly lower compared to the values of the same statistics for inner points and even border points. This behavior is expected. Note that our statistics is a sum of collision probabilities of the LSH mapping over all the data points $x_i \in \mathcal{D}$. From the theory of LSH, the collision probability $p(q, x_i)$ indicates the level of similarity between q and x_i. If q is an outlier, $p(q, x_i)$ is expected to be significantly low. We will further demonstrate the usefulness of these statistics in the experiments section.

Algorithm 1 Arrays of (locality-sensitive) Count Estimator(ACE) Algorithm

```
1:  Input: Dataset $D$, Number of Hashes $K$, Number of Hash tables $L$, $\alpha$
2:  Hash Initialize: Generate $L$ $H_j(.)$ using $K$ independent SRPs each.
3:  for $i = 1$ to $L$ do
4:      $A_j = new\ short[2^K](0)$ (Short Arrays)
5:      $\mu = 0, n = 0$
6:  end for

7:  Online Addition Phase
8:  for $x_i \in \mathcal{D}$ do
9:      $\mu_{incre} = 0$
10:     for $j = 1$ to $L$ do
11:         $A_j[H_j(x_i)] + +$
12:         $\mu_{incre} = \mu_{incre} + \frac{2A_j[H_j(x)]+1}{L}$
13:     end for
14:     $\mu = \frac{1}{n+1}\left(n\mu + \mu_{incre}\right)$
15:     n++;
16: end for

17: Query (Detection) Phase: Given query $q$
18: $\widehat{S(q,\mathcal{D})} = 0$
19: for $j = 1$ to $L$ do
20:     $\widehat{S(q,\mathcal{D})} = \widehat{S(q,\mathcal{D})} + \frac{1}{L}A_j[H_j(x_i)]$
21: end for
22: if $\widehat{S(q,\mathcal{D})} \leq \mu - \alpha$ then
23:     report $q$
24: end if
```

3.2 ACE (Arrays of (locality-sensitive) Counts Estimator) Algorithm

For the ease of explanation, we first describe the procedure of our proposed ACE algorithm. We later show that this procedure is an efficient statistical estimator of our proposed outlier score $S(q,\mathcal{D})$ defined by Equation 4.

The overall process of ACE is summarized in Algorithm 1. Our ACE algorithm, uses $K \times L$ independent SRP hash functions h_i, each given by Equation 1. K and L are hyperparameters that are pre-specified. Note, this is analogous to the traditional (K, L) parameterized LSH algorithm for near-neighbor search. However, we do not perform any retrieval which requires heavy hash tables with buckets of candidates for each hash index. For near-neighbor, we further need to compute the distances of these candidates to identify the best.

On the contrary, our algorithm does not require a single distance computation. Our method only needs to check the value of a simple counter at each index. We only need arrays of counters. The process is significantly efficient, both in memory and speed, compared to a single LSH near-neighbor query.

We use Signed Random Projections(SRP) h^{sim} (Equations 1) which gives one-bit output. Using these 1-bit outputs, we then generate L different meta-hash functions given by

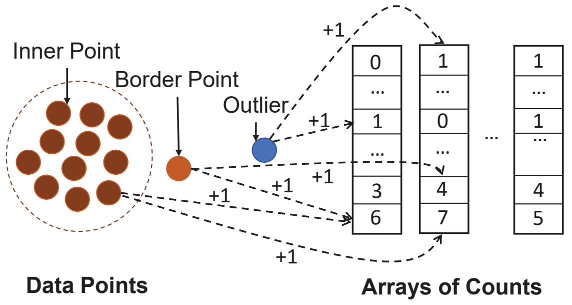

Figure 2: We use the LSH hash of the data points to increment corresponding counters into different (independent) hash arrays. We do not save anything, we only increase the value by 1 for each bucket and then forget the data.

$H_j(x) = [h_{j1}(x); h_{j2}(x); ...; h_{jK}(x)]$ of K bits each. The K bits are generated by concatenating the individual bits. Here h_{ij}, $i \in \{1, 2, ..., K\}$ and $j \in \{1, 2, ..., K\}$, are $K \times L$ independent evaluations of the SRP.

The overall algorithm works in the following two phases:
1) **Counting Phase:** We construct L short arrays, $A_j, j = \{1,\ 2, .., \ L\}$, of size 2^K each initialized with zeros. Given any observed element $x \in \mathcal{D}$, we increment the count of the corresponding counter $H_j(x)$ in array A_j, for all js. Thus, every counter keeps the total count of the number of hits to that particular index (See Figure 2). The total cost of updating the data structure for any given x is KL SRP computations followed by L increments.
Mean Update on Fly: For each $x_i \in \mathcal{D}$ our estimated score is

$$\widehat{S(x_i,\mathcal{D})} = \frac{1}{L}\sum_{j=1}^{L} A_j[H_j(x_i)].$$

We compute the mean behavior μ of the scores over all the element in dataset $x \in \mathcal{D}$.

$$\mu = \frac{1}{n}\sum_{i=1}^{n}\widehat{S(x_i,\mathcal{D})}.$$

Deviation from this mean will indicate outlierness. It turns out that we can dynamically update the mean μ on fly, as we read (or observe) the new data as shown in the algorithm. See Section 3.4.1 for details.
2) **Real-time (query) Phase:** Given a query q, for which we want to compute the score, we report the average of all the counters $A_j[H_j(q)]$ $\forall j \in \{1, 2, ..., L\}$, *i.e.*, $\widehat{S(q,\mathcal{D})} = \frac{1}{L}\sum_{j=1}^{L} A_j[H_j(q)]$. We report q as an anomaly if the estimated score $\widehat{S(q,\mathcal{D})}$ is less than $\mu - \alpha$, where α is some preselected hyperparameter. The overall cost for querying is KL SRP computations and L lookups followed by a simple average calculation.

3.3 Theory: Analysis and Superiority over Random Sampling

We first define few notations needed for analysis. Given a query point q. For convenience, we will denote $p(q, x_i)$, the collision probability of the SRP of q with that of $x_i \in \mathcal{D}$, by p_i. Due to space limitations the proofs are omitted.

Intuition: LSH as Samplers LSH is widely accepted as a black box algorithm for near neighbor search. We take an alternative adaptive sampling view of LSH which has emerged very recently [10, 12, 13, 41, 42]. As argued in Section 2, for a given query q and K-bit SRP hash function H_j, the probability that any element x_i increments the count of location $H_j(q)$ (the location of query) in array A_j is precisely $p(q, x_i)^K$. Using this observation, we will show that the count of the number of elements, from $\mathcal{D}$, hitting the bucket of query $H_j(q)$ is an unbiased estimator of the $S(q, C) = \sum_{i=1}^{n} p(q, x_i)^K$. This novel use of LSH as efficient data structure for statistical estimation could be of independent interest in itself.

We define indicator variable $\mathbb{I}_{x_i \in B_q}$ as

$$\mathbb{I}_{x_i \in B_q} = \begin{cases} 1, & \text{if } x_i \text{ is in the bucket of } q \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

Here, $\mathbb{I}_{x_i \in B_q}$ is an indicator for the event that data element x_i and the query q are in the same bucket. It should be noted that

$$Pr(\mathbb{I}_{x_i \in B_q} = 1) = p(q, x_i)^K = p_i^K \tag{6}$$

Note that, $\mathbb{I}_{x_i \in B_q}$ and $\mathbb{I}_{x_j \in B_q}$ are correlated. If x_i and x_j are "similar" then $\mathbb{I}_{x_i \in B_q} = 1$ is likely to imply $\mathbb{I}_{x_j \in B_q} = 1$. In other words, high similarity indicates positive correlation. Due to correlations, we may have both the cases:

$$\mathbb{E}[\mathbb{I}_{x_i \in B_q} \mathbb{I}_{x_j \in B_q}] \begin{cases} \geq p_i^K p_j^K, & \text{(positive correlations)} \\ \leq p_i^K p_j^K, & \text{(negative correlation).} \end{cases} \tag{7}$$

Here, $\mathbb{E}$ is the expectation.

Using the above notations we can show that, for a given query q, $\widehat{S(q, \mathcal{D})}$, computed in Algorithm 1, is an unbiased estimator of $S(q, \mathcal{D})$ with variance given by:

Theorem 3.1.

$$\mathbb{E}[\widehat{S(q, \mathcal{D})}] = \sum_{x_i \in \mathcal{D}} p_i^K = S(q, \mathcal{D})$$

$$Var(\widehat{S(q, \mathcal{D})}) = \frac{1}{L}\left(\sum_{i=1}^{n} p_i^K (1 - p_i^K) + \sum_{i \neq j} \left[\mathbb{E}[\mathbb{I}_{x_i \in B_q} \mathbb{I}_{x_j \in B_q}] - p_i^K p_j^K \right] \right)$$

The variance of $\widehat{S(q, \mathcal{D})}$ is dependent on the data distribution. There are two terms in the variance $\frac{1}{L}\left(\sum_{i=1}^{n} p_i^K (1 - p_i^K) \right)$ and $\frac{1}{L} \sum_{i \neq j} \left[\mathbb{E}[\mathbb{I}_{x_i \in B_q} \mathbb{I}_{x_j \in B_q}] - p_i^K p_j^K \right]$. The terms inside summation is precisely the covariance between $\mathbb{I}_{x_i \in B_q}$ and $\mathbb{I}_{x_j \in B_q}$

$$\mathbb{E}[\mathbb{I}_{x_i \in B_q} \mathbb{I}_{x_j \in B_q}] - p_i^K p_j^K = \mathbb{E}[\mathbb{I}_{x_i \in B_q} \mathbb{I}_{x_j \in B_q}] \tag{8}$$

$$- \mathbb{E}[\mathbb{I}_{x_i \in B_q}] \mathbb{E}[\mathbb{I}_{x_j \in B_q}] \tag{9}$$

$$= Cov(\mathbb{I}_{x_i \in B_q}, \mathbb{I}_{x_j \in B_q}) \tag{10}$$

There are $n(n-1)$ covariance terms in the second term of variance, $\frac{1}{L} \sum_{i \neq j} \left[\mathbb{E}[\mathbb{I}_{x_i \in B_q} \mathbb{I}_{x_j \in B_q}] - p_i^K p_j^K \right]$. To see why almost all of them will be negative, let m be the number of elements in the buckets of the query. So only pairs x_i and x_j in the bucket ($O(m^2)$ pairs) of query $H_j(q)$ will contribute $1 - p_i^K p_j^K \geq 0$ to the summation (product of indicators is 1 $\iff$ both are 1). Rest all pairs ($O((n-m)^2)$) will contribute negative terms $-p_i^K p_j^K$. Thus, if we choose K large enough then the expected number of elements in the bucket m is quite small. Hence, we can expect the variance to be significantly smaller than $\left(\sum_{i=1}^{n} p_i^K \frac{(1 - p_i^K)}{L} \right)$. We observe in our experiments that $K = 15$ is a good recommended constant value.

As noted the variance is dependent on the data distribution. If we have all exact duplicates, then all the covariances are positive. However, for real datasets, for any randomly chosen pair x_i, x_j, the covariance $Cov(\mathbb{I}_{x_i \in B_q}, \mathbb{I}_{x_j \in B_q})$ will be almost always be negative.

An alternative way of estimating $S(q, \mathcal{D})$ is to use the random sampling. The idea is to uniformly sample a subset $\mathcal{S} \subseteq \mathcal{D}$ of size L and report the random sampling estimator $RSE(q, \mathcal{D})$ as:

$$RSE(q, \mathcal{D}) = \frac{n}{L} \Big[\sum_{x_i \in \mathcal{S}} p_i^K \Big] \tag{11}$$

From the theory of random sampling this estimator is also unbiased and has the following variance:

Theorem 3.2.

$$\mathbb{E}[RSE(q, \mathcal{D})] = \sum_{x_i \in \mathcal{D}} p_i^K = S(q, \mathcal{D})$$

$$Var(RSE(q, \mathcal{D})) = \sum_{i=1}^{n} p_i^K \left(\left[\frac{n}{L} - 1 \right] p_i^K \right)$$

Both $RSE(q, \mathcal{D})$ and $\widehat{S(q, \mathcal{D})}$ are unbiased. For the same number of samples, the estimator with smaller variance is superior.

We can get some insights from the leading terms $\sum_{i=1}^{n} p_i^K \left(\left[\frac{n}{L} - 1 \right] p_i^K \right)$ and $\frac{1}{L}\left(\sum_{i=1}^{n} p_i^K (1 - p_i^K) \right)$. Generally, for any i and large enough n, we always have $\left[\frac{n}{L} - 1 \right] \geq \frac{1}{L}\left(\frac{1 - p_i^K}{p_i^K} \right)$. Thus, for large enough n,

$$Var(RSE(q, \mathcal{D})) > \frac{1}{L}\left(\sum_{i=1}^{n} p_i^K (1 - p_i^K) \right)$$

As argued before for real data we can expect $\frac{1}{L}\left(\sum_{i=1}^{n} p_i^K (1 - p_i^K) \right) > Var(\widehat{S(q, \mathcal{D})})$. Precise mathematical comparison between the variances of these two estimators is fairly challenging due to data-dependent correlation.

Empirical Comparison: As argued, we expect that for real datasets the ACE estimator to be more accurate (less variance) compared to the random sampling estimator. To validate our arguments empirically, we compare these estimators on the three benchmark anomaly detection datasets. These are the same datasets used in the experiment sections (see section 5.1 for details). For all the three datasets, we randomly chose 50 queries and estimate their $S(q, \mathcal{D})$ using the two competing estimators. We use $K = 15$ which is the fixed value used in all our experiments.

We plot the mean square error of the estimates, computed using the actual and the estimated values on three real anomaly detection datasets, in Figure 3. We vary the number of samples for random sampling estimator $RSE(q, \mathcal{D})$ and the number of arrays for $\widehat{S(q, \mathcal{D})}$.

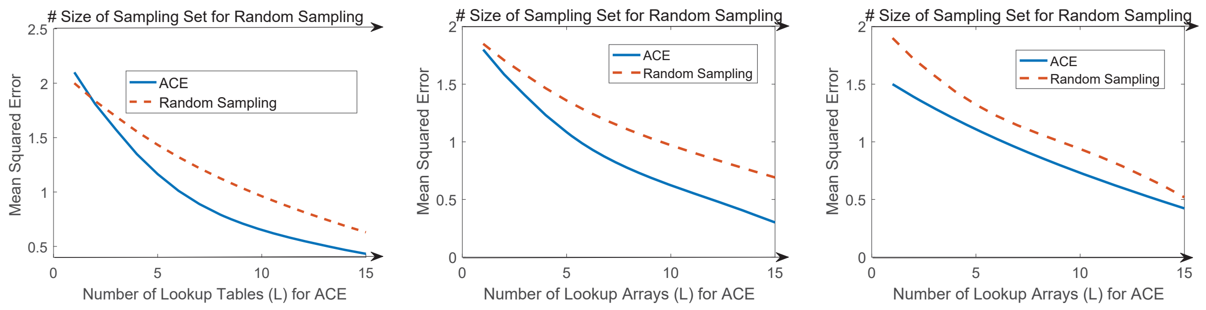

Figure 3: Comparison of ACE estimator with random sampling estimator on three datasets. The x-axis denotes the number to arrays and size of samples for ACE estimator and random sampling estimator respectively. ACE estimator is not only more accurate but also cheaper compared to random sampling estimators from the computational perspective.

From the plots, it is clear that on all the three real datasets, as expected from our analysis, our ACE estimator $\widehat{S(q, \mathcal{D})}$ consistently outperforms the random sampling estimator $RSE(q, \mathcal{D})$ at the same level of L. Note, these estimators are unbiased and hence mean square error value is also the theoretical variance. These experiments indicate that the variance of our ACE estimator is superior for estimating $S(q, \mathcal{D})$ over random sampling.

In addition to providing sharper estimates, in the next section, we show that our ACE algorithm only needs $O(d \log d + KL)$ computations to calculate the score. Here, d is the dimensions of the dataset. For the same number of samples L, random sampling estimator requires $O(Ld)$ computations. Given that $K = 15$ is a fixed constant. For high dimensional datasets, we will have $d > K$. Thus, our estimator is not only more accurate but also cheaper compared to random sampling estimators from the computational perspective.

3.4 Implementation Details, Running Time, Cache Utilization and Memory

Running Time: From Algorithm 1, it is not difficult to see that for a query q, we need to compute KL hashes of the data followed by a simple addition of size L. The costliest step is the computations of KL hashes, which for d dimensional data can be accomplished in $O(d \log d + KL)$ computations using advances in fast random projections (Section 2). If instead, we are using minwise hashing as the LSH then it can be done in mere $O(d + KL)$ using fast minwise hashes. However, minwise hashing is limited to binary datasets only.

Note that computing the original score $S(q, \mathcal{D})$ via naive calculation requires $O(nd)$ computations. It further require to store all the data for outlier detection, which for large and high dimensional datasets can be prohibitive.

In all our experiments, we use $K = 15$ and $L = 50$ for all the three datasets (see Section 6). Thus, with these small constant values, our scoring time negligible compared to the other algorithms which requires one pass over the full dataset $\mathcal{D}$. In the experiments, we see that even with such minuscule computation, our method provides competitive accuracy while being orders of magnitude faster than 11 state-of-the-art methods.

Memory: Since we have 2^K counters, it is unlikely that the counters will get too many hits. To save memory by a factor of two, we can use short integers (16 bits) instead of integer counters. The total amount of memory required by L counter arrays is 2^K bytes each if we use short counters. The total space needed for the arrays is $L \times 2^K \times 2$ bytes. For $K = 15$ and $L = 50$, the total space required by the ACE algorithm is around $3.2MB$. In addition, we need to compute $KL = 750$ hashes, which requires storing 750 random seeds (integers) from which we can generate hashes on the fly. 750 integers require negligible space compared to $3.2MB$. In the worst case, even if we decide to store the full random projections, we only need $750 \times d \times 8$ bytes (approx $6d$ kilobytes).

L3 Cache Utilizations: For all of our experiments, the total memory requirement of the ACE algorithm is $\leq 4MB$ for all the datasets. Our query data structure, the arrays, can easily fit into L3 cache of any modern processor, where the memory access can be anywhere from 2-10x faster than the main memory (DRAM) access. Detecting anomaly requires scoring which only needs reading count from the arrays. Due to all these unique favorable properties, our algorithm is orders of magnitude faster than the fastest available packages for unsupervised anomaly detection.

3.4.1 *Dynamic Updates.*

One of the appealing features of the ACE algorithm is that data can be dynamically updated. It is straightforward to increment the counters if we decide to add any data x. However, we will lose all the data information. We only store a set of count arrays, so it is not clear how we update the global mean μ of counts. Updating μ is an important part of Algorithm 1. Note that the updated mean, μ', should be the average of all the estimated score of all the data in $\mathcal{D}' = \mathcal{D} + x$.

It turns out that we can exactly compute the new value of μ' from the existing count arrays. To simplify, let us convert old mean μ to sum by multiplying it by the size of dataset $n = |\mathcal{D}|$. It is easy to keep track of the sum

$$n\mu = \sum_{x_i \in \mathcal{D}} \frac{1}{L} \sum_{j=1}^{L} A_j[H_j(x_i)].$$

Observe that, if the new x goes to location $H_j(x)$ in array A_j for any j. The count of location H_j will be increment by 1. This will also lead to an increment in the scores of all the elements which maps to $H_j(x)$ in j^{th} array by exactly $\frac{1}{L}$. Since we already know the count value of $A_j[H_j(x_i)]$, the total increment to the sum would be $\frac{A_j[H_j(x_i)]}{L}$. In addition, the new data x will add an extra $\frac{A_j[H_j(x_i)]+1}{L}$ for its own count. Thus, we can precisely compute the increment in the sum. The new mean μ', for an addition of data x, can be computed as

$$\mu' = \frac{1}{n+1}\left(n\mu + \sum_{j=1}^{L} \frac{2A_j[H_j(x)] + 1}{L}\right). \quad (12)$$

4 DISCUSSIONS: PRIVACY PRESERVING ANOMALY DETECTION

Privacy is becoming one of the sought after directions in data mining and machine learning. Privacy preserving anomaly detection is of broad interest in the big-data and IoT (Internet of Things) community [45]. In many setting, we do want to detect anomalies in the data. However, it also desirable that the attribute information remains private and secure. It turns out that our proposed ACE algorithm has ideal properties for privacy preserving anomaly detection.

ACE does not require storing any data attributes, and the complete algorithm works only over aggregated counts generated from hashed data. If the hashes are not invertible, then the algorithm is safe. We can exploit advances in the secure computation to design protocols which hide the hashing mechanism [19].

Obtaining differential privacy [5, 31] with ACE is quite appealing and neat. Since ACE algorithm relies on random projections to compute hashes, instead of original data, we can make ACE algorithm differentially private by adding only Gaussian noise instead of heavy-tailed Laplacian noise. [24] shows a way to release user information in a privacy-preserving way for near-neighbor search. The paper showed that adding Gaussian noise $N(0, \sigma^2)$ after the random projection preserves differential privacy. Any function of differentially private object it also differentially private. Thus, to compute a private variant of SRP (Signed random projection), we used the sign of the differentially private random projections (generated by adding Gaussian noise to usual projection) as suggested in [24].

The final algorithm is very simple. The data is never revealed to anyone. At the source itself, the sign of differentially private random projections of data is used instead of usual SRP. All other process remains the same. Now since, we are only perturbing our algorithm with Gaussian noise, instead of Laplacian, we can expect a minimal loss in utility (or change in output).

Note, that privacy is significantly harder with other state-of-the-art anomaly detection algorithms that store the actual data or even samples. Making such algorithms private requires perturbing the system with heavy-tailed Laplacian noise, which can significantly hurt the outcome of the algorithm.

5 EXPERIMENTAL EVALUATIONS

5.1 Datasets

We choose three real-world benchmark datasets for anomaly detection: 1) **Statlog Shuttle**, 2) **Object Images (ALOI)**, and 3) **KDD-Cup99 HTTP**. These datasets are labeled and hence can be used for quantifying the effectiveness of anomaly detection measure. These three datasets also cover a broad spectrum of applications of unsupervised anomaly detection.

The first dataset we use is the shuttle dataset [1]. This dataset describes radiator positions in a NASA space shuttle with 9 attributes. It was designed for supervised anomaly detection. In the original datasets, about 20% of the data regarded as anomaly. The entire dataset contains 34, 987 instances with 879 anomalies.

The second dataset is Object Images (ALOI) datasets[2]. The aloi dataset is derived from the "Amsterdam Library of Object Images" collection [16]. It contains about 110 images of 1000 small objects taken under different light conditions and viewing angles. From the original images, a 27 dimensional feature vector was extracted using HSB color histograms [35]. Some objects were chosen as anomalies, and the data was down-sampled such that the resulting dataset contains 50, 000 instances including 1508 anomalies.

The third dataset is KDD-Cup99 HTTP. [3]. KDD-Cup99 HTTP dataset [28] is the largest benchmark for unsupervised anomaly detection evaluation. It contains simulated normal and attack traffic on an IP level in a computer network environment in order to test intrusion detection systems. There are total of 36 dimensions. The dataset contains 596, 853 instances with 1055 labeled anomalies.

The statistics of these datasets are shown in Table. 1.

Table 1: The statistics of the three datasets.

Dataset	Instances	Outliers	Dimension
Statlog Shuttle	34, 987	879	9
Object Images (ALOI)	50, 000	1508	27
KDD-Cup99	596, 853	1055	36

Table 2: Comparison Algorithms and Their Parameter Values Recommended for These Benchmark Datasets.

Method	Shuttle	Image Object	KDD-CUP 99
ACE	$K = 15, L = 50$	$K = 15, L = 50$	$K = 15, L = 50$
LOF	$k = 5$	$k = 5$	$k = 10$
kNN	$k = 5$	$k = 5$	$k = 10$
kNNW	$k = 5$	$k = 5$	$k = 10$
LoOP	$k_r = k_c = 5$ $\lambda = 0.2$	$k_r = k_c = 5$ $\lambda = 0.2$	$k_r = k_c = 10$ $\lambda = 0.2$
LDOF	$k = 5$	$k = 5$	$k = 10$
ODIN	$k = 5$	$k = 5$	$k = 10$
KDEOS	$k = 5$ $B = 5, s = 0.2$ Gaussian Kernel	$k = 5$ $B = 5, s = 0.2$ Gaussian Kernel	$k = 10$ $B = 5, s = 0.2$ Gaussian Kernel
COF	$k = 5$	$k = 5$	$k = 10$
LDF	$h = 1, c = 0.1$ Gaussian Kernel	$h = 1, c = 0.1$ Gaussian Kernel	$h = 1, c = 0.1$ Gaussian Kernel
INFLO	$k = 5, m = 0.5$	$k = 5, m = 0.5$	$k = 10, m = 0.5$
FastVOA	$k = 5, \lvert S_2\rvert = 2, \lvert S_1\rvert = 320$	$k = 5, \lvert S_2\rvert = 2, \lvert S_1\rvert = 320$	$k = 10, \lvert S_2\rvert = 2, \lvert S_1\rvert = 320$

5.2 Baselines

We use 11 different state-of-the-art methodologies to compare with ACE. These methodologies cover the whole spectrum of unsupervised anomaly detection techniques with all sorts of variations developed over the years. Our baselines cover very recent scoring mechanisms based on simple to sophisticated strategies which include near-neighbor, kernel density estimation, graph connectedness, *etc.* The competing methodologies are: **ACE** (Proposed),

[1]https://archive.ics.uci.edu/ml/datasets/Statlog+(Shuttle)

[2]http://aloi.science.uva.nl/

[3]http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

Table 3: Comparison Results

Dateset	Method	Reported	Correct	Missed	F1-score	F1-Rank	Time (s)	Speed-up with ACE
Statlog Shuttle	ACE	6763	273	606	0.071	5	**0.81s**	1x
	LOF	4356	381	498	0.145	3	14.12s	17.4x
	kNN	4897	493	386	0.170	2	12.35s	15.2x
	kNNW	5264	610	269	0.199	1	13.54s	16.7x
	LoOP	6145	201	678	0.057	8	14.51s	17.9x
	LDOF	6433	330	549	0.090	4	16.42s	20.3x
	ODIN	9775	375	504	0.071	6	12.21s	15.1x
	KDEOS	12630	314	565	0.046	12	11.73s	14.5x
	COF	9133	280	599	0.056	11	13.45s	16.6x
	LDF	9809	375	504	0.070	7	19.93s	24.6x
	INFLO	4488	183	696	0.068	8	14.03	17.3x
	FastVOA	8532	271	608	0.057	10	235.10s	290.2x
Image Object	ACE	7216	340	1168	0.078	5	**1.26s**	1x
	LOF	4476	519	989	0.1735	1	72.31s	57.4x
	kNN	5428	447	1061	0.1289	2	63.27s	50.2x
	kNNW	5558	329	1508	0.089	4	89.96s	71.4x
	LoOP	5121	253	1179	0.077	6	59.97s	47.6x
	LDOF	7501	470	1038	0.1043	3	60.39s	47.9x
	ODIN	10110	162	1346	0.028	12	72.69s	57.6x
	KDEOS	9515	404	1104	0.073	7	55.89s	44.36x
	COF	8746	284	1224	0.055	11	81.74s	64.9x
	LDF	9133	301	1207	0.056	10	60.51s	48.0x
	INFLO	10328	420	1088	0.071	8	72.13s	57.2x
	FastVOA	8931	319	1189	0.061	9	291.10s	231.0x
KDD-CUP 99	ACE	15160	406	649	0.051	5	**23.33s**	1x
	LOF	13260	523	532	0.073	1	1813.63s	77.7x
	kNN	15432	365	690	0.044	7	1483.54s	63.5x
	kNNW	14328	460	595	0.059	2	2125.43s	91.1x
	LoOP	16578	396	659	0.045	6	1594.54s	68.3x
	LDOF	16579	496	559	0.056	3	1674.43s	71.7x
	ODIN	18054	365	690	0.038	10	1918.34s	82.2x
	KDEOS	21095	469	586	0.042	8	1428.32s	61.2x
	COF	20658	584	471	0.054	4	2043.43s	87.5x
	LDF	19574	368	687	0.036	11	1485.85s	63.7x
	INFLO	25704	565	490	0.042	9	1684.47s	72.2x
	FastVOA	29316	354	701	0.023	12	3510.26s	150.4x

LOF (Local Outlier Factor) [6], **FastVOA** (Fast Variance of Angles) [32], **kNN** (KNNOutlier) [33], **KNNW** (KNNWeightOutlier) [4], **LoOP** (Local Outlier probability) [26], **LDOF** (Local Distance based Outlier Factor) [47], **ODIN** (Outlier Detection using Indegree Number) [21], **LDF** (Local density factor) [27], **KDEOS** (Kernel Density Estimation Outlier Score) [36], **COF** (Connectivity-based Outlier Factor) [43] and **INFLO** (Influenced Outlierness) [23].

We use the highly optimized recent *ELKI* (Environment for Developing KDD-Applications Supported by Index-Structures) package [34] which is the most advanced set of anomaly detection algorithms noted for its efficient implementations. 10 of our baselines methodologies are implemented in this package. For FastVOA, a state-of-the-art randomized algorithm for variance of angle computation, we use the C++ package provided by the authors.

It should be noted that ACE and FastVOA are implemented in C++, while *ELKI* is a java package. A direct wall clock comparison is not fair. However, given the simplicity of our algorithm (Algorithm 1) which only requires simple hashing, use of primitive arrays, and simple summations. We do need any complex object other than arrays of short integers (primitives only). All other operations are primitive multiplications and summations. Thus, we expect that the difference between Java and C++ implementation would not be any significant for ACE. Furthermore, we show a significant speedup which cannot be explained by the difference in platforms.

Parameter Settings: Almost all of our baseline algorithms need hyper-parameters. We use most of the default settings of the parameters as implemented. For the baseline algorithms, the *ELKI* package has the recommended settings of parameters for these benchmark datasets. To avoid complications, we directly use those recommended settings. For the sake of reproducibility, we provide the precise recommended settings of the parameters for different methods and datasets used in the paper in Table 2. It should be noted that for ACE we use the fixed value of $K = 15$ and $L = 50$ for all the datasets. ACE does not need the near neighbor parameter k (small). Variations in parameter K and L are discussed in Section 6.

System and Platform Details: Experiments were conducted on a 3.50 GHz core Xeon Windows platform with 16GB of RAM. We use g++ (version 5.4.0) as the C++ compiler for ACE and fastVOA. For *ELKI* package, we use OpenJDK 64bits version 1.8.0.

5.3 Methodology and Results

All of these 12 algorithms associate a score with every element in the data. After association, a significantly lower score from the mean indicates an anomaly. In order to convert these scores into an anomaly detections algorithm, there are many reasonable strategies. We can rank each candidate, based on scores, and report bottom-k as the anomalies, but such rankings are not realistic. In real-time applications, ranking all the seen records is artificial. A more practical approach is to use a threshold strategy to report anomalies. We compute the mean μ and the standard deviations σ of the scores on the dataset of interest and report any element with the associated score less than $\mu - \sigma$ as an anomaly.

With the above introduced anomaly detection strategies, we run all the 12 algorithms on the three datasets. We report seven different numbers separately for each of the three datasets: (1) Number of outliers reported; (2) Number of outliers correctly reported; (3) Number of Outliers missed; (4) F1-score; (5) Ranking of F1-score; (6) the CPU execution time for the different methods, and (7) Relative speed with ACE. The CPU executing time is the end to end time of the complete run of the algorithm, which includes data reading, preprocessing (if any), scoring every data instance, and reporting outliers. Relative speedup reports the ratio of the time required by a given algorithm to the time required by ACE algorithm. The results of each datasets are shown in Table 3.

Accuracy Comparison. We report the F1-scores[2] of each method. F1-score is a widely used method for evaluating the performance of anomaly detection methods, for the detailed definition of F1-score please refer [2]. Based on the F1-scores of each method, we rank the different methods. From the results, we can see LOF seems to be consistently more accurate than others. ACE is ranked consistently among the top-5 ranked methods on all the datasets. The number of anomalies reported correctly (true positives) with ACE is similar to other algorithms. ACE, however, seems to report slightly more anomalies (high false positives) than other algorithms. This is not a major concern though. Few extra false positives are easy to deal with because we can always further filter them using a more sophisticated algorithm, so long as they are small. Overall, our proposed new scoring scheme $S(q, \mathcal{D})$ and the corresponding estimator performs very competitively, in terms of accuracy, in comparison with many successful algorithms.

Running Time Comparison. The most exciting part is the computational savings with ACE. From the result, we observe that ACE is significantly faster than any other alternatives irrespective of the choice of dataset. ACE algorithm is at least around 15x, 45x and 60x faster than the best competitor on Statlog Shuttle, Object Images (ALOI), and KDD-Cup99 HTTP datasets respectively. Most of the algorithms, based on near-neighbors except FastVOA, have similar speeds. This could be because almost all of them requires computation of the order of the data. FastVOA is consistently very slow, which we suspect is because the estimators used in FastVOA is computationally very expensive. FastVOA estimators require multiple sorting and frequently computing costly medians. See [32] for details. ACE is around 150-300x faster than FastVOA.

Memory Analysis. The results are even more exciting if we start considering the memory requirements. With $K = 15$ and $L = 50$, our methodology requires less than $4MB$ of operating memory for the complete run of the algorithm. Since we use the same K and L across all datasets, this $4MB$ requirement is unaltered. We never keep any data in the memory. On the other hand, all other methods except FastVOA require storing complete data in the memory. In our case, the KDD-Cup99 HTTP dataset itself is around 165MB to store. Although KDD-Cup99 HTTP dataset is the largest labeled benchmark, it is still tiny from big-data perspective. The disruptive performance of ACE is not surprising given the simplicity of the process. However, as argued, the process is a statistically sound procedure for estimating the proposed score $S(q, \mathcal{D})$.

6 DISCUSSION: EFFECTS OF K AND L

Parameters K and L determine the memory and also the running time of the ACE algorithm. Note L is also the number of independent samples used for averages. Therefore, a reasonably large L is good enough, after which increasing L does not give significant accuracy but hurts the performance. K cannot be too small because locations in arrays should distinguish anomalies with everything else. However, too large K is not needed either. Ideally, if K is $\log n$, then under random assignments all data will go to the single bucket. Beyond this K, the performance is lost for no gain in accuracy.

To stress test, we ran ACE for with different values of $K = \{2, 5, \ldots, 20\}$ and $L = \{10, 20, \ldots, 100\}$. For the Image dataset, the minimum reasonable result appears at K=8, L=30. For the shuttle dataset, K=11, L=10 is fine, and for the KDDCUP dataset, the minimum fair result appears at K=9, L=30. These parameters give similar results as shown with fixed $K = 15$ and $L = 50$. With these parameters, the ACE took mere 0.5, 0.2 and 11.5 seconds on the Image, shuttle and KDDCUP dataset for a negligible loss in accuracy compared to what is shown in Table 3. The results degrade if we decrease K and L beyond these numbers. Increasing K and L values, significantly beyond $K = 15$ and $L = 50$, does not increase the accuracies significantly but, as expected, hurts the performance.

7 CONCLUSION

Statistical measures for popular learning and data mining problems, such as anomaly detection, were designed without taking into account the computational complexity of the estimation process. When faced with current big-data challenges, most of these estimation process fail to address tight resources constraints. In this paper, we showed that for the problem of unsupervised anomaly detection, we could leverage advances in probabilistic indexing and redesign a significantly efficient statistical measure.

We proposed ACE algorithm, for unsupervised anomaly detection, which is 60-300x faster than existing approaches with competing accuracy. Our algorithm requires mere $4MB$ of memory which can utilize L3 caches of modern processors leading to fast-lookups. We believe ACE will replace existing unsupervised anomaly detection algorithms deployed in resource-frugal environments.

ACKNOWLEDGMENTS

This work was supported by National Science Foundation IIS-1652131, RI-1718478, and a GPU grant from NVIDIA

REFERENCES

[1] Elke Achtert, Thomas Bernecker, Hans-Peter Kriegel, Erich Schubert, and Arthur Zimek. 2009. ELKI in Time: ELKI 0.2 for the Performance Evaluation of Distance Measures for Time Series. In *Advances in Spatial and Temporal Databases, 11th International Symposium, SSTD 2009, Aalborg, Denmark, July 8-10, 2009, Proceedings*. 436–440.

[2] Charu C Aggarwal and Philip S Yu. 2001. Outlier detection for high dimensional data. In *ACM Sigmod Record*, Vol. 30. ACM, 37–46.

[3] Nir Ailon and Bernard Chazelle. 2006. Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*. ACM, 557–563.

[4] Fabrizio Angiulli and Clara Pizzuti. 2005. Outlier mining in large high-dimensional data sets. *IEEE transactions on Knowledge and Data engineering* 17, 2 (2005), 203–215.

[5] Kanishka Bhaduri, Mark D Stefanski, and Ashok N Srivastava. 2011. Privacy-preserving outlier detection through random nonlinear data distortion. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 41, 1 (2011), 260–272.

[6] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. 2000. LOF: identifying density-based local outliers. In *ACM sigmod record*, Vol. 29. ACM, 93–104.

[7] Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas Dimitropoulos. 2010. SEPIA: Privacy-preserving aggregation of multi-domain network events and statistics. *Network* 1, 101101 (2010).

[8] Aniket Chakrabarti, Venu Satuluri, Atreya Srivathsan, and Srinivasan Parthasarathy. 2015. A bayesian perspective on locality sensitive hashing with extensions for kernel methods. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 10, 2 (2015), 19.

[9] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM computing surveys (CSUR)* 41, 3 (2009), 15.

[10] Moses Charikar and Paris Siminelakis. [n. d.]. Hashing-based-estimators for kernel density in high dimensions.

[11] Moses S Charikar. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. ACM, 380–388.

[12] Beidi Chen, Anshumali Shrivastava, and Rebecca C Steorts. 2017. Unique Entity Estimation with Application to the Syrian Conflict. *arXiv preprint arXiv:1710.02690* (2017).

[13] Beidi Chen, Yingchen Xu, and Anshumali Shrivastava. 2018. LSH-SAMPLING BREAKS THE COMPUTATIONAL CHICKEN-AND-EGG LOOP IN ADAPTIVE STOCHASTIC GRADIENT ESTIMATION. (2018). https://openreview.net/forum?id=SyVOjfbRb

[14] Cisco Visual Networking Index Cisco. 2014. Global mobile data traffic forecast update, 2013–2018. *white paper* (2014).

[15] Anirban Dasgupta, Ravi Kumar, and Tamás Sarlós. 2011. Fast locality-sensitive hashing. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1073–1081.

[16] Jan-Mark Geusebroek, Gertjan J Burghouts, and Arnold WM Smeulders. 2005. The Amsterdam library of object images. *International Journal of Computer Vision* 61, 1 (2005), 103–112.

[17] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. 1999. Similarity search in high dimensions via hashing. In *VLDB*, Vol. 99. 518–529.

[18] Michel X Goemans and David P Williamson. 1994. .879-approximation algorithms for max cut and max 2sat. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*. ACM, 422–431.

[19] Oded Goldreich. 1998. Secure multi-party computation. *Manuscript. Preliminary version* (1998), 86–97.

[20] Nico Görnitz, Marius Micha Kloft, Konrad Rieck, and Ulf Brefeld. 2013. Toward supervised anomaly detection. *Journal of Artificial Intelligence Research* (2013).

[21] Ville Hautamaki, Ismo Karkkainen, and Pasi Franti. 2004. Outlier detection using k-nearest neighbour graph. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, Vol. 3. IEEE, 430–433.

[22] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. ACM, 604–613.

[23] Wen Jin, Anthony KH Tung, Jiawei Han, and Wei Wang. 2006. Ranking outliers using symmetric neighborhood relationship. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 577–593.

[24] Krishnaram Kenthapadi, Aleksandra Korolova, Ilya Mironov, and Nina Mishra. 2012. Privacy via the johnson-lindenstrauss transform. *arXiv preprint arXiv:1204.2606* (2012).

[25] Jon Kleinberg. 2002. Bursty and hierarchical structure in streams. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 91–101.

[26] Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. 2009. LoOP: local outlier probabilities. In *Proceedings of the 18th ACM conference on Information and knowledge management*. ACM, 1649–1652.

[27] Longin Jan Latecki, Aleksandar Lazarevic, and Dragoljub Pokrajac. 2007. Outlier detection with kernel density functions. In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*. Springer, 61–75.

[28] Kingsly Leung and Christopher Leckie. 2005. Unsupervised anomaly detection in network intrusion detection using clusters. In *Proceedings of the Twenty-eighth Australasian conference on Computer Science-Volume 38*. Australian Computer Society, Inc., 333–342.

[29] Chen Luo, Jian-Guang Lou, Qingwei Lin, Qiang Fu, Rui Ding, Dongmei Zhang, and Zhe Wang. 2014. Correlating events with time series for incident diagnosis. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1583–1592.

[30] Chen Luo and Anshumali Shrivastava. 2017. SSH (Sketch, Shingle, & Hash) for Indexing Massive-Scale Time Series. In *NIPS 2016 Time Series Workshop*. 38–58.

[31] Ashwin Machanavajjhala, Daniel Kifer, John Abowd, Johannes Gehrke, and Lars Vilhuber. 2008. Privacy: Theory meets practice on the map. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*. IEEE, 277–286.

[32] Ninh Pham and Rasmus Pagh. 2012. A near-linear time approximation algorithm for angle-based outlier detection in high-dimensional data. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 877–885.

[33] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. 2000. Efficient algorithms for mining outliers from large data sets. In *ACM Sigmod Record*, Vol. 29. ACM, 427–438.

[34] Erich Schubert, Alexander Koos, Tobias Emrich, Andreas Züfle, Klaus Arthur Schmid, and Arthur Zimek. 2015. A framework for clustering uncertain data. *Proceedings of the VLDB Endowment* 8, 12 (2015), 1976–1979.

[35] Erich Schubert, Remigius Wojdanowski, Arthur Zimek, and Hans-Peter Kriegel. 2012. On evaluation of outlier rankings and outlier scores. In *Proceedings of the 2012 SIAM International Conference on Data Mining*. SIAM, 1047–1058.

[36] Erich Schubert, Arthur Zimek, and Hans-Peter Kriegel. 2014. Generalized outlier detection with flexible kernel density estimates. In *Proceedings of the 2014 SIAM International Conference on Data Mining*. SIAM, 542–550.

[37] Anshumali Shrivastava. 2016. Simple and efficient weighted minwise hashing. In *Advances in Neural Information Processing Systems*. 1498–1506.

[38] Anshumali Shrivastava. 2017. Optimal Densification for Fast and Accurate Minwise Hashing.. In *ICML*.

[39] Anshumali Shrivastava and Ping Li. 2014. Densifying One Permutation Hashing via Rotation for Fast Near Neighbor Search.. In *ICML*. 557–565.

[40] Anshumali Shrivastava and Ping Li. 2014. Improved Densification of One Permutation Hashing.. In *UAI*.

[41] Ryan Spring and Anshumali Shrivastava. 2017. A New Unbiased and Efficient Class of LSH-Based Samplers and Estimators for Partition Function Computation in Log-Linear Models. *ArXiv e-prints* (2017). arXiv:1703.05160

[42] Ryan Spring and Anshumali Shrivastava. 2017. Scalable and sustainable deep learning via randomized hashing. In *KDD*.

[43] Jian Tang, Zhixiang Chen, Ada Wai-Chee Fu, and David W Cheung. 2002. Enhancing effectiveness of outlier detections for low density patterns. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 535–548.

[44] Luan Tran, Liyue Fan, and Cyrus Shahabi. 2016. Distance-based outlier detection in data streams. *Proceedings of the VLDB Endowment* 9, 12 (2016), 1089–1100.

[45] Jaideep Vaidya and Chris Clifton. 2004. Privacy-preserving outlier detection. In *Data Mining, 2004. ICDM'04. Fourth IEEE International Conference on*. IEEE, 233–240.

[46] Xindong Wu, Vipin Kumar, J Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J McLachlan, Angus Ng, Bing Liu, S Yu Philip, et al. 2008. Top 10 algorithms in data mining. *Knowledge and information systems* 14, 1 (2008), 1–37.

[47] Ke Zhang, Marcus Hutter, and Huidong Jin. 2009. A new local distance-based outlier detection approach for scattered real-world data. *Advances in knowledge discovery and data mining* (2009), 813–822.