University of Rhode Island

DigitalCommons@URI

11-24-2021

CLUSTERED HIERARCHICAL ANOMALY AND OUTLIER DETECTION ALGORITHMS

Najib Ishaq
University of Rhode Island

Thomas J. Howard III
University of Rhode Island

Noah M. Daniels
University of Rhode Island, noah_daniels@uri.edu

Follow this and additional works at: https://digitalcommons.uri.edu/cs_facpubs

Terms of Use

Citation/Publisher Attribution

CLUSTERED HIERARCHICAL ANOMALY AND OUTLIER DETECTION ALGORITHMS

Clustered Hierarchical Anomaly and Outlier Detection Algorithms

Najib Ishaq
Department of Computer Science and Statistics
University of Rhode Island
Kingston, RI
najib_ishaq@uri.edu

Thomas J. Howard III
Department of Computer Science and Statistics
University of Rhode Island
Kingston, RI
thoward27@uri.edu

Noah M. Daniels
Department of Computer Science and Statistics
University of Rhode Island
Kingston, RI
noah_daniels@uri.edu

November 24, 2021

ABSTRACT

Anomaly and outlier detection is a long-standing problem in machine learning. In some cases, anomaly detection is easy, such as when data are drawn from well-characterized distributions such as the Gaussian. However, when data occupy high-dimensional spaces, anomaly detection becomes more difficult. We present CLAM (Clustered Learning of Approximate Manifolds), a manifold mapping technique in any metric space. CLAM begins with a fast hierarchical clustering technique and then induces a graph from the cluster tree, based on overlapping clusters as selected using several geometric and topological features. Using these graphs, we implement CHAODA (Clustered Hierarchical Anomaly and Outlier Detection Algorithms), exploring various properties of the graphs and their constituent clusters to find outliers. CHAODA employs a form of transfer learning based on a training set of datasets, and applies this knowledge to a separate test set of datasets of different cardinalities, dimensionalities, and domains. On 24 publicly available datasets, we compare CHAODA (by measure of ROC AUC) to a variety of state-of-the-art unsupervised anomaly-detection algorithms. Six of the datasets are used for training. CHAODA outperforms other approaches on 16 of the remaining 18 datasets. CLAM and CHAODA scale to large, high-dimensional "big data" anomaly-detection problems, and generalize across datasets and distance functions. Source code to CLAM and CHAODA are freely available on GitHub[1].

1 Introduction and Related Work

Detecting anomalies and outliers from data is a well-studied problem in machine learning. When data occupy easily-characterizable distributions, such as the Gaussian, the task is relatively easy: one need only identify when a datum is sufficiently far from the mean. However, in "big data" scenarios, where data can occupy high-dimensional spaces, anomalous behavior becomes harder to quantify. If the data happen to be uniformly distributed, one can conceive of simple mechanisms, such as a one-class SVM, that would be effective in any number of dimensions. However, real-world data are rarely distributed uniformly. Instead, data often obey the "manifold hypothesis" [16], occupying a low-dimensional manifold in a high-dimensional embedding space, similar to how a 2-d sheet of paper, once crumpled, occupies a 3-dimensional space. Detecting anomalies in such a landscape is not easy. Imagine trying to identify if a

---

[1]https://github.com/URI-ABD/clam

point within the sphere of crumpled paper were anomalous; identifying whether or not the point sits on the sheet or within a gap of the lower-dimensional manifold presents a challenge.

Anomalies (data that do not belong to a distribution) and outliers (data which represent the extrema of a distribution) are found in datasets for a variety of reasons. In this manuscript, we treat anomalies and outliers interchangeably. Errors in the measurement of data create anomalous records, novel instances of data emerge over time, or normal behavior may evolve towards the outliers. Moreover, even if data collection is executed perfectly and shifts in behavior are accounted for, there is still the risk of adversarial attacks causing undesired behavior [13].

Modern algorithms designed to detect anomalous behavior fail for a variety of reasons, in particular when anomalies live close to, but not on, a complex manifold in high-dimensional space. Our approach is designed to map these complex manifolds. Here we briefly survey contemporary approaches to anomaly detection in order to provide the context needed to understand how our approach differs.

1.1 Clustering-based Approaches

Clustering refers to techniques for grouping points in a way that provides value. This is generally done by assigning *similar* points to the same cluster. Given a clustering and a new point, one can estimate the anomalousness of that new point by measuring its distance to its nearest cluster(s).

There have been few advancements in clustering techniques over the past decade [47]. This may be explained by the poor performance, thus far, of clustering in high-dimensional space [51].

Distance-based clustering relies on some distance measure to partition data into several clusters. Within this approach, the numbers or sizes of clusters are often predetermined: either user-specified, or randomly chosen [47]. Examples of distance-based clustering algorithms are K-Means [32], PAM [24], CLARANS [33] and CLARA [24].

Hierarchical clustering methods build a tree, where points are allocated into leaves [47]. These trees can be created via bottom-up (agglomerative) or top-down (divisive) [1] clustering. The high cost of all-pairs distance computations is often a drawback. Examples of hierarchical clustering algorithms include MST [50], CURE [18], and CHAMELEON [23].

Grid-based clustering works via segmenting the entire volume into a discrete number of cells, and then scanning those cells to find regions of high density. The grid structure typically lets these methods scale well to larger datasets. Some examples include STING [49], Wavecluster [44], CLIQUE [1], Clustering Based Local Outlier Factor (CBLOF) [20], and Local Correlation Integral (LOCI) [36].

1.2 Density-based Approaches

Density-based clustering methods rely on finding regions of high point-density separated by regions of lower density. These algorithms generally do not work well when data are sparse or uniformly distributed. Some examples include DBSCAN [14] and DENCLUE [21]. In this paper, we compare against Local Outlier Factor (LOF) [7].

1.3 Graph-based Approaches

Graph-based methods build graph representations of the data by treating points, or collections of points, as nodes in a graph and drawing edges between them based on various rules. One challenge with graph-based methods is finding a suitable graph representation for a dataset; a user-specified distance threshold is often used to induce edges. In this paper, we compare against Connectivity-based Outlier Factor (COF) [45].

1.4 Distance-based Approaches

Distance-based methods use direct distance comparisons and largely rely on k-Nearest Neighbors as their substrate [47]. They tend to use the following intuitions. First, points with fewer than p other points within some distance d are outliers; second, the n points with the greatest distances to their $k^{th}-$nearest neighbor are outliers; and finally, the n points with the greatest average distance to their k nearest neighbors are outliers. Drawbacks to distance-based approaches include the computational complexity of the underlying KNN algorithm, and determining a good value for k. We compare against k-Nearest Neighbors (kNN) [39, 3] and Subspace Outlier Detection (SOD) [25].

1.5 Deep Learning Approaches

With the recent explosion in Deep Learning, several new methods for anomaly detection have been introduced [35]. While most such approaches are supervised or weakly-supervised, such as DAGMM [53], there have been some advances

in unsupervised methods. These typically use autoencoders [9], such as RandNet [8], or generative-adversarial-networks, such as MO-GAAL and SO-GAAL [31]. Drawbacks to deep learning approaches include interpretability of the model, the wide variety of possible architecture and thus hyperparameters, and intensive computational and GPU requirements during training. We compare against MO-GAAL, SO-GAAL, and two autoencoders from [9].

1.6   Other Approaches

There are several approaches to anomaly detection that do not fall neatly into any of the aforementioned categories. These methods often rely on support vector machines, random forests, or histograms to detect outliers. We compare against seven methods among these: Histogram-Based Outlier Detection (HBOS)[17], Isolation-Forest (IFOREST) [28, 29], One-class Support Vector Machine (OCSVM) [43], Linear Model Deviation-base outlier Detection (LMDD) [4], Lightweight Online Detector of Anomalies (LODA) [38], Minimum Covariance Determinant (MCD) [41, 19], and Subspace Outlier Detection (SOD) [25].

1.7   Our Approach

With the term *manifold learning* being largely synonymous with dimension reduction, we propose *manifold mapping* to refer to approaches that study the the geometric and topological properties of manifolds in their original embedding spaces. We introduce a novel technique, Clustered Learning of Approximate Manifolds (CLAM) for datasets in a metric space (more general than a Banach space); [5] essentially a set of datapoints and a distance metric defined on that set. CLAM presupposes the manifold hypothesis and uses a divisive hierarchical clustering to build a map of the manifold occupied by the dataset. CLAM then provides this map of the manifold to be used by a collection of anomaly detection algorithms, which we call CHAODA (Clustered Hierarchical Anomaly and Outlier Detection Algorithms).

CLAM extends CHESS [22] by adding memoized calculations of several geometric and topological properties of clusters that are useful to CHAODA, and does so in expected $\mathcal{O}(n \lg n)$ time. While, in principle, we could have used any hierarchical clustering algorithm, these memoized calculations are not provided for by any other algorithm. Other clustering algorithms also suffer from problems, such as: an ineffective treatment of high dimensionality, an inability to interpret results, and an inability to scale to exponentially-growing datasets [1]. CLAM, as we will demonstrate, largely resolves these issues. Thus, CLAM and CHAODA are well-suited to anomaly detection on large, high-dimensional "big data."

CHESS was used to build a hierarchical clustering to a user-specific depth for the sole purpose of accelerating search. CLAM, however, divisively clusters the data until each cluster contains only one datum. Using the cluster-tree, CLAM induces graphs by mapping specific clusters to vertices of a graph, and drawing an edge between any two vertices whose corresponding clusters have overlapping volumes (i.e., the distance between their centers is less than or equal to the sum of their radii). Clusters can be selected from a fixed depth, or from heterogeneous depths based on properties such as their local fractal dimension, cardinality, radius, etc. We can consider clusters at lower depths in the tree to be, in some sense, "lower resolution" than those at greater depths. Inducing a graph across a variety of depths effectively maps a manifold with a variety of "resolutions," the intuition being that some regions of the manifold may have a higher density of points than others and, thus, graphs induced from clusters deeper in the tree may be more informative for those regions.

Having mapped a manifold by clustering and inducing graphs, we can start to analyze several properties of the clusters in the graphs. For example: what are the relative cardinalities of the clusters in the graph, how connected are the clusters, how often is each cluster visited by a random walk? CHAODA uses answers to such questions, among others, to build an ensemble approach to anomaly detection.

2   Methods

CLAM and CHAODA comprise many components, all described in this section. We start with a brief overview of those components. CLAM begins with a dataset and a distance metric, to which it applies hierarchical clustering to build a tree. CLAM selects clusters from the tree using meta-machine-learning (meta-ml) models trained[2] according to several geometric and topological properties. These meta-ml models learn relationships between these properties and expected anomaly-detection performance. CLAM then induces graphs from the selected clusters. CHAODA applies

---

[2]Note that during training, CHAODA is somewhat supervised. CHAODA uses this training to learn a set of meta-ml models for selecting clusters and inducing graphs. During testing, and inference, on new datasets, CHAODA is unsupervised and uses the learned meta-ml models.

its constituent algorithms to these graphs, and combines the individual scores into an ensemble, ultimately producing anomaly scores for each datum. See Figure 1 for a high-level illustration.
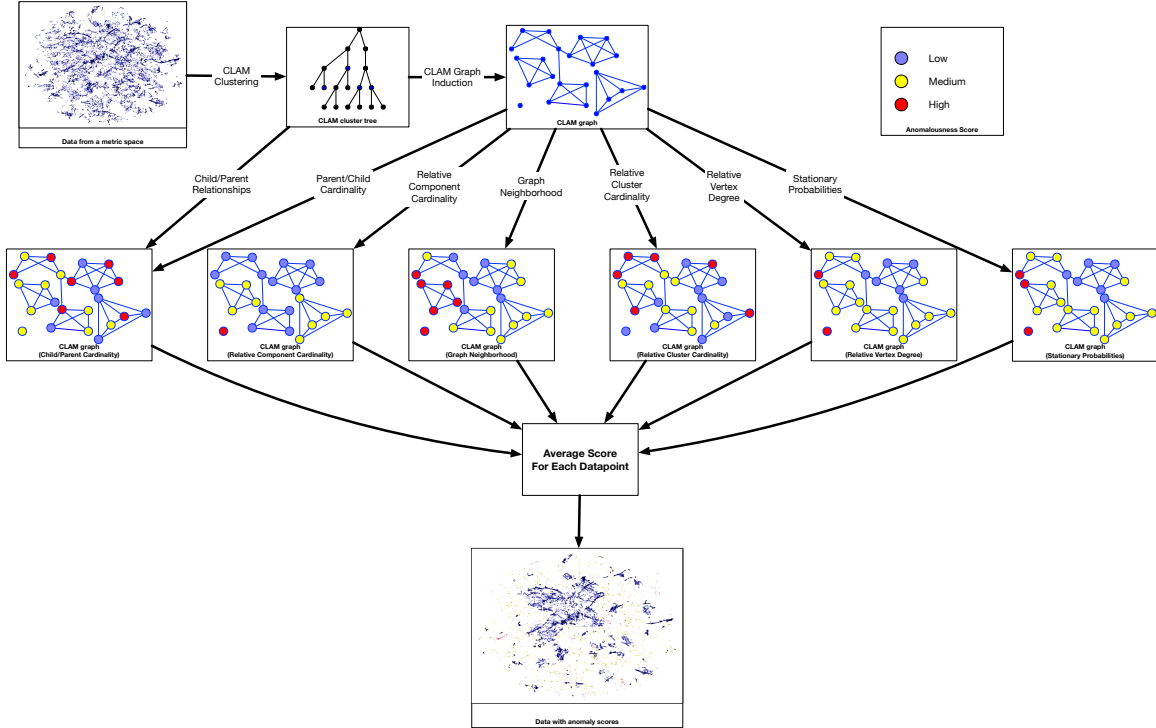

Figure 1: **Overview of the CHAODA workflow.** Beginning with a dataset and a distance metric, CLAM builds a cluster tree and induces several graphs from this tree; for the sake of simplicity, we illustrate only one such graph here. Each of CHAODA's constituent algorithms provides distinct anomaly scores on its graph. These scores are normalized and aggregated into a final score for each cluster, and by extension, each datum. In this figure, we have simplified the scores to a ternary color scheme; actual scores are real-valued between 0 and 1. Note that each algorithm provides its own scoring, but there may be similarities such as between vertex degree and stationary distribution.

2.1 Dataset and Distance Function

We start with a dataset $\mathbf{X} = \{x_1 \ldots x_n\}$ with n points and a distance function $f : (\mathbf{X}, \mathbf{X}) \mapsto \mathbb{R}^+$. The distance function takes two points from the dataset and deterministically produces a non-negative real number. We also require the distance function to be symmetric and for the distance between two identical points to be zero, i.e., $\forall x, y \in X$, $f(x, y) = f(y, x)$ and $f(x, y) = 0 \Leftrightarrow x = y$. CLAM and CHAODA are general over any distance function that obeys these constraints.

CLAM assumes the "manifold hypothesis" [16], i.e. datasets collected from constrained generating phenomena that are embedded in a high-dimensional space typically only occupy a low-dimensional manifold in that space. CLAM and CHAODA learn the geometric and topological properties of these manifolds in a way that generalizes across datasets and distance function regardless of dataset-specific properties such as total number of points, dimensionality, absolute distance values, etc. We demonstrate this genericity by our anomaly detection performance in Section 3.

Note that we often speak of each datum as embedded in a D-dimensional metric space and we use Euclidean notions, such as voids and volumes, to talk about the geometric and topological properties of the manifold. The purpose of such notions is to help build intuition and to aid understanding. Mathematically, CLAM does not rely on such notions; in-fact, the details of an embedding space can be abstracted away behind the distance function.

Also note that we can provide certain guarantees (see CHESS [22]) when the distance function is a metric, i.e. it obeys the triangle inequality. While CLAM and CHAODA work well with distance functions that are not metrics, we have not investigated how the lack of the triangle inequality changes, or breaks, those guarantees in the context of anomaly detection. For this paper, we show results using the $L1$-norm and $L2$-norm.

2.2 Clustering

We start by building a divisive hierarchical clustering of the dataset. We partition, as described in Algorithm 1, a cluster with k points using a pair of well-separated points from among a random sampling of $\sqrt{k}$ points. Starting from a root-cluster containing the entire dataset, we continue until each leaf contains only one datum. This achieves clustering in expected $\mathcal{O}(n \lg n)$ time. This procedure improves upon the clustering approach from CHESS [22] by a better selection of maximally-separated points, and by memoizing critical information about each cluster (discussed below).

Algorithm 1 Partition

Require: $cluster$

1: $k \leftarrow \lfloor \sqrt{|cluster.points|} \rfloor$
2: $seeds \leftarrow k$ random points from $cluster.points$
3: $c \leftarrow$ geometric median of $seeds$
4: $r \leftarrow \arg\max d(c, x) \ \forall \ x \in cluster.points$
5: $l \leftarrow \arg\max d(r, x) \ \forall \ x \in cluster.points$
6: $left \leftarrow \{x | x \in cluster.points \wedge d(l, x) \leq d(r, x)\}$
7: $right \leftarrow \{x | x \in cluster.points \wedge d(r, x) < d(l, x)\}$
8: **if** $|left| > 1$ **then**
9:     Partition($left$)
10: **end if**
11: **if** $|right| > 1$ **then**
12:     Partition($right$)
13: **end if**

These clusters have several interesting and important properties for us to consider. These include the *cardinality*, the number of points in a cluster; *center*, the approximate geometric median of points contained in a cluster; *radius*, the distance to the farthest point from the center; and *local fractal dimension*, as given by:

$$\log_2 \left(\frac{|B_X(c, r)|}{|B_X(c, \frac{r}{2})|} \right) \tag{1}$$

where $B_X(c, r)$ is the set of points contained in a ball of radius r on the dataset X centered on a point c [22]. Thus, local fractal dimension captures the "spread" of points on the manifold in comparison to the (typically much larger) embedding space. This is motivated by the idea that the induced graphs will learn to adapt to use different "resolutions" to characterize different regions of the manifold (see Figure 3).

We can also consider *child-parent ratios* of the cardinality, radius, and local fractal dimension of a cluster, as well as the *exponential moving averages* of those child-parent ratios along a branch of the tree. In particular, we use the child-parent ratios and their exponential moving averages to help CHAODA generalize from a small set of training datasets to a large, distinct set of testing datasets. During clustering, we memoize these ratios as we create each cluster. CHAODA can then make direct use of these ratios to aid in anomaly detection.

2.3 Graphs

Clusters that are close together sometimes have overlapping volumes; i.e., the distance between their centers is less than or equal to the sum of their radii. We define a graph $G = (V, E)$ with vertices in one-to-one correspondence to CLAM clusters and with an edge between two vertices if and only if their corresponding clusters overlap. While it is fairly standard in the literature to define graphs in this way, the challenge lies in selecting the right clusters to build useful graphs. Our selection process, presented in Section 2.6, is among the major novel contributions of CLAM and CHAODA.

In the context of graphs, we use the terms *cluster* and *vertex* interchangeably. By *graph cardinality* we mean *vertex cardinality*, i.e., the number of clusters in the graph, and by *graph population* we mean the sum of cardinalities of all clusters in the graph. Note that *cluster cardinality* refers to the number of points within a cluster. We use *layer-graph* to refer to a graph built from clusters at a fixed depth from the tree and *optimal-graph* to refer to a graph built from clusters selected by the processes described in Section 2.6.

Figure 2 illustrates how CLAM induces a graph from non-uniform depths in a cluster tree, while Figure 3 illustrates how, if clusters are chosen at the right "resolution," these graphs can capture the structure of the manifold. Interestingly,

the clusters are not necessarily hyperspheres, but polytopes akin to a high-dimensional Voronoi diagram [46]. The induced graph need not be fully connected and, in practice, often contains many small, disjoint connected components.
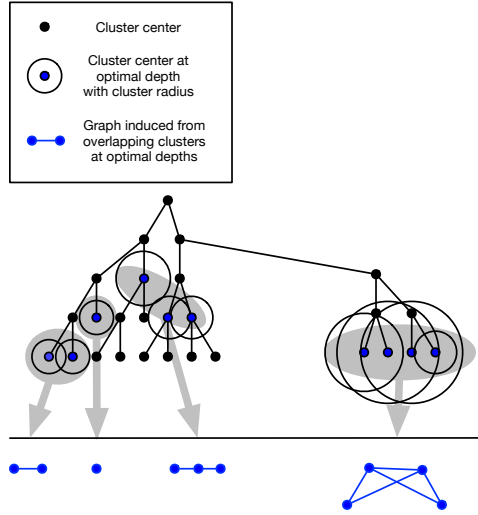

Figure 2: Using CLAM to induce a graph from a cluster tree. Dots in the tree represent cluster centers; blue dots represent centers of chosen clusters. Circles represent the volume of a cluster (the radius is the distance from the center to the furthest point contained within that cluster). Gray arrows point to the induced graph components, which are indicated in blue below the horizontal line.

For our purposes, a CLAM graph exhibits an important invariant. The clusters corresponding to vertices in the graph collectively contain every point in the dataset, and each point in the dataset is assigned to exactly one cluster in the graph. A corollary to this invariant is that a graph will never contain two clusters such that one cluster is an ancestor or descendant of another cluster. This also assures that a *graph's population* is equal to the cardinality of the dataset, i.e. $|\mathbf{X}|$ or n.

2.4   Individual Algorithms

Given an induced graph that characterizes a manifold, we must extract information about the anomalousness of clusters in that graph. Here we describe six simple algorithms for anomaly detection, each using a CLAM graph to calculate an anomalousness score for each cluster and datum. Given that the key to an effective ensemble is for each member to contribute a unique inductive bias [9], we also note the intuition behind each algorithm's contributions. These scores can be used, along with the ground-truth labels, to compute the area under the curve (AUC) of the receiver operating characteristic (ROC) [15] to measure the anomaly detection performance of the graph which produced those scores.

In the following, V and E are the sets of clusters and edges respectively in a graph, $|c|$ is the cardinality of a cluster c, and $|C|$ is the cardinality of a component C. Each algorithm assigns an anomalousness score to each cluster. Each point is then assigned the anomalousness score of the cluster it belongs to. These scores are internally consistent for each individual algorithm, i.e. low scores indicate inliers and high scores indicate outliers. However, different algorithms assign scores in wide, and often different, ranges of values. We use Gaussian normalization to constrain the raw scores to a $[0, 1]$ range. This lets us combine scores into an ensemble (see Section 2.7). See [26] for a thorough discussion of anomaly-score normalization in ensemble methods.

The overall computational complexity of these algorithms appears in Table 1.

2.4.1   Relative Cluster Cardinality

We measure the anomalousness of a point by the cardinality of the cluster that the point belongs to relative to the cardinalities of the other clusters in the graph. Points in the same cluster are considered equally anomalous. Points in clusters with lower cardinalities are considered more anomalous than points in clusters with higher cardinalities. Formally, $\forall c \in G$, $\mathrm{score}(c) = -|c|$.

The intuition is that points in clusters with higher cardinalities are close to each other, and thus are less likely to be anomalous. The time complexity is $\mathcal{O}(|V|)$ because this requires a single pass over the clusters in a graph.
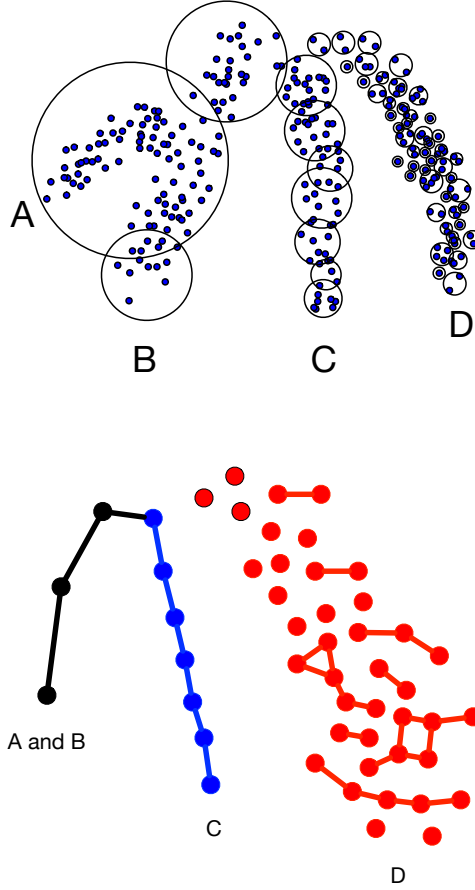
Figure 3: **Cluster resolution.** Consider this toy dataset whose manifold comprises the four named branches. In the top part of the figure, with each branch, we show clusters that we might get from different depths in the tree. Clusters on branches A and B come from low depths in the tree and have large voids, i.e. regions with no points in them. Clusters on branch D come from a high depth in the tree and are, in a sense, too small for the "thickness" of the branch they cover. Clusters on branch C are "just right" because their diameters are roughly equal to the thickness of the branch, and they contain no large voids. We can track how the local fractal dimension of these clusters changes as we traverse the tree and as we move along clusters that are adjacent on the manifold. In this way, changes in the local fractal dimension can serve as a proxy for deciding which clusters would help "well characterize" the underlying manifold. In the bottom part of the figure, we show the graphs CLAM would induce from these different clusters. Note that branches A and B are not distinguished; the separation between the branches is lost in the graph representation. A graph induced from branch D would consist of many disconnected subgraphs, and would not represent the structure of the entire branch. Finally, a graph induced from branch C represents the branch structure, including its connection to branches A and B.

Table 1: Time complexity of CHAODA algorithms.

Algorithm	Complexity
Relative Cluster Cardinality	$\mathcal{O}(|V|)$
Relative Component Cardinality	$\mathcal{O}(|E| + |V|)$
Graph Neighborhood Size	$\mathcal{O}(|E| \cdot |V|)$
Child-Parent Cardinality Ratio	$\mathcal{O}(|V|)$
Stationary Probabilities	$\mathcal{O}(|V|^{2.37})$
Relative Vertex Degree	$\mathcal{O}(|V|)$

2.4.2 Relative Component Cardinality

We use the usual definition of connected components: no two vertices from different components have an edge between them and every pair of vertices in the same component has a path connecting them. We consider points in clusters in

smaller components to be more anomalous than points in clusters in larger components. Points in clusters in the same component are considered equally anomalous. Formally, $\forall C \in G$, $\forall c \in C$,

$$score(c) = -|C|.$$

The intuition here, as distinct from the previous algorithm, is to capture larger-scale structural information based on disjoint connected components from the graph. The time complexity is $\mathcal{O}(|E| + |V|)$ because we first need to find the components of the graph using a single pass over the edges, and then score each cluster in the graph using a single pass over those clusters.

2.4.3 Graph Neighborhood Size

Given the graph, we consider the number of clusters reachable from a starting cluster within a given graph distance k, i.e. within k hops along edges. We call this number the *graph-neighborhood size* of the starting cluster. With k small compared to the diameter of a component, we consider the relative graph-neighborhood-sizes of all clusters. Clusters with small graph-neighborhoods are considered more anomalous than clusters with large graph-neighborhoods.

The intuition here is to capture information about the connectivity of the graph in the region around each cluster. The computation is defined in Algorithm 2. Its time complexity is $\mathcal{O}(|E| \cdot |V|)$ because we need to compute the eccentricity of each cluster.

Algorithm 2 Graph Neighborhood

Require: G, a graph
Require: $\alpha \in \mathbb{R}$ in the range $(0, 1]$ (0.25 by default).
1: **for** cluster $c \in G$ **do**
2:      $e_c \leftarrow$ the eccentricity of c
3:      $s \leftarrow e_c \cdot \alpha$
4:      perform a breadth-first traversal from c with s steps
5:      $v \leftarrow$ the number of unique clusters visited
6:      score$(c) \leftarrow -v$
7: **end for**

2.4.4 Child-Parent Cardinality Ratio

As described in Section 2.2, the partition algorithm used in clustering splits a cluster into two children. If a child cluster contains only a small fraction of its parent's points, then we consider that child cluster to be more anomalous. These child-parent cardinality ratios are accumulated along each branch in the tree, terminating when the child cluster is among those selected in the graph. Clusters with a low value of these accumulated ratios are considered more anomalous than clusters with a higher value. Formally, $\forall c \in G$, score$(c) = \frac{|p|}{|c|} + score(p)$ where p is the parent cluster of c.

This algorithm was inspired by iForest [28], and captures information from the tree and the graph. Unlike the other individual algorithms, this accumulates parent scores into the children. The time complexity of this algorithm is $\mathcal{O}(|V|)$, because these ratios are memoized during the clustering process and we need only look them up once for each cluster in the graph.

2.4.5 Stationary Probabilities

For each edge in the graph, we assign a weight inversely proportional to the distance between the centers of the two clusters that connect to form that edge. The outgoing probabilities from a cluster are stochastic over the edge weights for that cluster. We compute the transition probability matrix of each component that contains at least two clusters. The process of successively squaring this matrix will converge [27]. We follow this process for each component in the graph and find the convergent matrix. Consider the sum of the values along a row in this matrix. This is the expected proportion of visits to that cluster during an infinitely long random walk over the component. We consider this sum to be inversely related to the anomalousness of the corresponding cluster.

The intuition here is that clusters that are more difficult to reach during an infinite random walk are more likely to contain anomalous points. The algorithm is defined in Algorithm 3. Its worst-case time complexity is $\mathcal{O}(|V|^{2.37})$ given by the matrix multiplication algorithm from [2]. In practice, however, this algorithm has much better performance than indicated by the theoretical complexity, because the induced graphs are often composed of several small components rather than one, or a few large, component(s).

Algorithm 3 Stationary Probabilities

Require: G, a graph
 1: **for** component $C \in G$ **do**
 2:    $M \leftarrow$ the transition matrix for C
 3:    **repeat**
 4:       $M \leftarrow M^2$
 5:    **until** M converges
 6:    **for** cluster $c \in C$ **do**
 7:       $s \leftarrow$ the row from M corresponding to c
 8:       $\text{score}(c) \leftarrow -\Sigma(s)$
 9:    **end for**
10: **end for**

2.4.6 Relative Vertex Degree

For each cluster in the induced graph, consider its degree, i.e. the number of edges connecting to that cluster. We consider a cluster with high degree to be less anomalous than a cluster with low degree. This is essentially a version of the previous algorithm that ignores edge weights, and will have different biases with regard to the sampling density of the dataset. Formally, $\forall c \in G,\ \text{score}(c) = -\deg(c)$. Its time complexity is $\mathcal{O}(|V|)$.

2.5 Training Meta-Machine-Learning Models

Section 2.2 makes note of some important geometric and topological properties of CLAM clusters, i.e. cardinality, radius and local fractal dimension. We find the child-parent ratios of these properties and the exponential moving averages of these ratios along each branch in the tree. Each child-parent ratio is obtained by dividing the value for the child by the value for the parent, e.g. $R = \frac{|child|}{|parent|}$. Each new exponential moving average (EMA) is the weighted sum of the previous EMA and the current ratio. Specifically, $ema_{i+1} = \alpha * R_{i+1} + (1 - \alpha) * ema_i$ for some $\alpha \in [0, 1]$. We chose an α of $\frac{2}{11}$.

Using these ratios instead of the raw values themselves makes CHAODA agnostic to dataset-specific properties; it need only consider how those properties change as we traverse the tree or a graph. For a given graph, we can take the average values of the six ratios from its constituent clusters to form a feature-vector. We can use the methods described in Section 2.4 to compute the area under the ROC curve from using each individual algorithm to predict anomalousness scores from that graph. Each pairing of the feature-vector and an ROC score forms a training sample for our meta-ml models. We use linear regression and decision-tree regressors to fill the role of those meta-ml models. We use these data to train the meta-ml models to predict the ROC score for a graph from its feature-vector.

We randomly selected six datasets whose cardinalities are between 10^3 and 10^5 for training, and we used the $L1$-norm and $L2$-norm for each dataset. For each pairing of dataset and distance function, CLAM builds a new cluster-tree. Meta-ml training then proceeds over several epochs, the first of which we seed with some layer graphs from each tree. During each epoch, we extract the feature vector from each graph, and we find the ROC AUC of applying each individual algorithm to each graph. Each pairing of feature-vector and ROC score forms a training sample. For each pairing of dataset and distance function, we initialize a linear regressor and a decision-tree regressor to form our suite of meta-ml models. We train each meta-ml model with every training sample collected thus far, for ten epochs. We use the trained meta-ml models to select clusters (see Section 2.6) for new graphs that are used for the next epoch. We note that this was not k-fold cross validation, but a one-time selection of six datasets for training based on size as a selection criterion.

During the earlier epochs, we expect to have selected graphs that exhibit poor anomaly detection performance. For later epochs, we expect this performance to improve. With each epoch, we add to the set of training samples collected thus far and we train a new suite of meta-ml models for selecting better clusters. This is so that the meta-ml models can learn to distinguish between ratios that select for low ROC AUC from those ratios that select for high ROC AUC. Each meta-ml model sees training data from each pairing of dataset and distance function. This lets CHAODA generalize across different datasets and distance functions.

2.6 Cluster Selection for Graphs

The heart of the problem with CHAODA is in selecting the "right" clusters that would build a graph that provides a useful representation of the underlying manifold. One could try every possible combination of clusters to build graphs,

but this quickly leads to combinatorial explosion. Instead, CHAODA focuses on intelligently selecting clusters for a graph which is expected to perform well for anomaly detection. Area under the curve (AUC) of the receiver operating characteristic (ROC) is often used to benchmark anomaly detectors [15]. CHAODA selects clusters to optimize for this measure.
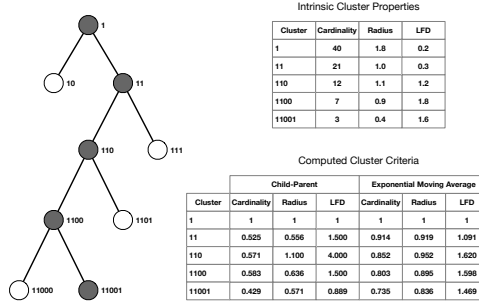

Intrinsic Cluster Properties

Cluster	Cardinality	Radius	LFD
1	40	1.8	0.2
11	21	1.0	0.3
110	12	1.1	1.2
1100	7	0.9	1.8
11001	3	0.4	1.6

Computed Cluster Criteria

Cluster	Child-Parent			Exponential Moving Average		
	Cardinality	Radius	LFD	Cardinality	Radius	LFD
1	1	1	1	1	1	1
11	0.525	0.556	1.500	0.914	0.919	1.091
110	0.571	1.100	4.000	0.852	0.952	1.620
1100	0.583	0.636	1.500	0.803	0.895	1.598
11001	0.429	0.571	0.889	0.735	0.836	1.469

Figure 4: **Cluster properties.** In the illustrated tree, we highlight only one branch for simplicity. We name the root '1' and we name the descendants as we might for a huffman tree. The upper table is an example of the values that intrinsic cluster properties might take on. The lower table shows the derived ratios we use for learning how to select clusters.

Specifically, we train a number of meta-ml models (see Section 2.5 for details) and, from each model, we extract a function of the form $g : c \rightarrow \mathbb{R}$. This function assigns high values to clusters which would increase ROC AUC and low values to clusters which would decrease ROC AUC. As described in Algorithm 4, the selection process begins by sorting, in non-increasing order, all clusters in the tree by the value assigned by g. This sorting represents a ranking of the clusters for expected anomaly detection performance. We iteratively select the best cluster from the rankings, and with each selection, we remove the ancestors and descendants of the selected cluster from the list of rankings. Once the list of rankings is exhausted, we have selected the clusters with which to build an *optimal graph*.

Algorithm 4 Cluster Selection

Require: T, a cluster-tree
Require: $g : c \rightarrow \mathbb{R}$ a ranking function
 1: $G \leftarrow$ an empty graph
 2: $h \leftarrow$ a list of all clusters $c \in T$ sorted by $g(c)$
 3: **repeat**
 4:     $c \leftarrow$ pop_first(h)
 5:     Add c to G
 6:     Remove all ancestors and descendants of c from h
 7: **until** h is empty

2.7 The Ensemble

During the testing and inference phases, we begin with a new dataset (not included in the training set of datasets) and one or more distance functions. CLAM first builds cluster-trees using each distance function with the given dataset. CHAODA uses the trained meta-ml models to select a different graph from each tree for each individual algorithm. CHAODA applies each individual algorithm to its corresponding graph and produces anomalousness scores for each datum. With two distance functions, six individual algorithms, and two meta-ml models, we can get up to 24 different members with which to form an ensemble. CHAODA normalizes the scores from all members and aggregates them, by their mean, into a final set of predictions for the anomalousness of each datum.

2.8 Datasets and Comparisons

We sourced 24 datasets containing only numerical features, i.e. not categorical features, from Outlier Detection Datasets (ODDS) [40]. All of these datasets were adapted from the UCI Machine Learning Repository (UCIMLR) [12], and were standardized by ODDS for anomaly detection benchmarks. Note that CHAODA is able to handle either entirely-numerical or entirely-categorical datasets, but not mixed datasets. We discuss some future work relating to this in Section 5.

We randomly selected six datasets to train CHAODA: ann-thyroid, mnist, pendigits, satellite, shuttle, and thyroid. The other eighteen datasets were used for testing and benchmarks: arrhythmia, breastw, cardio, cover, glass, http, ionosphere, lymphography, mammography, musk, optdigits, pima, satimage-2, smtp, vertebral, vowels, wbc, and wine. We benchmarked CHAODA 30 times, using different random seeds, on the test set of datasets (see the Supplement at https://github.com/URI-ABD/chaoda for more details). During testing, we noticed that even though we often see $|V| \ll n$, the graph neighborhood size and stationary probabilities methods from 2.4 took prohibitively long to run, so we only use them when $|V| < max(128, \lfloor \sqrt{n} \rfloor)$. We present these results in Table 2 under the CHAODA-fast and CHAODA rows. CHAODA-fast exhibits comparable performance to CHAODA, and we offer it as an option in our implementation. All benchmarks were conducted on a 28-core Intel Xeon E5-2690 v4 2.60GHz, 512GB RAM and CentOS 7 Linux with kernel 3.10.0-1127.13.1.el7.x86_64 #1 SMP and Python 3.6.8.

We use the ground-truth labels only during the training phase with a small set of datasets. Having once been trained, CHAODA becomes an unsupervised algorithm for any new dataset. As such, we compared CHAODA only against other unsupervised algorithms. We selected 18 unsupervised algorithms from the pyOD suite [52] and Scikit-Learn [37], as well as RS-Hash [42]. A supervised version of CHAODA is possible future work, which would open up comparisons against supervised or weakly-supervised methods such as REPEN [34] and DAGMM [53].

For a "Big-Data" challenge, we ran CHAODA on the APOGEE2 data from the SDSS [6]. This dataset has a cardinality of $528, 319$ and a dimensionality of $8, 575$. See Section 3.1 for results. All of these datasets were prepared by UCI and ODDS. In our experiments, we simply read them as 2-dimensional arrays where the columns are the features and the rows are the instances. We pre-processed the APOGEE2 data into a similar array, but of course it has no ground-truth labeling.

3   Results

The performance on the 18 test datasets is in Table 2. Performance on the 6 training datasets is shown in the Supplement at `https://homepage.cs.uri.edu/~ndaniels/pdfs/chaoda-supplement.pdf`. Each column shows the ROC scores of CHAODA and every competitor. The highest score and every score within 0.02 is presented in bold. We found that setting a different random seed resulted in a variance of at most 0.02 ROC AUC for CHAODA.

If a method exceeded 10 hours on a dataset, we mark the corresponding cell with "*TO*". If a method crashed, we mark the cell with "*EX*". Notably, CHAODA performed best (or tied for best) on 16 of the 18 test datasets. Runtime performance is presented in the Supplement. Note that we implemented CLAM and CHAODA entirely in Python, while the methods we compared against are often implemented in C/C++. Therefore, the comparison of runtime is not truly fair to CHAODA. An implementation in a high-performance language, such as Rust, would be worthwhile.

We considered several recently published algorithms against which to compare. Those with available implementations are included in Table 2. When unable to find a working implementation, we include here the performance claimed by the respective authors. RS-Hash [42] reported AUCs of 0.92 on Cardio, 1.00 on Lympho, 0.99 on Musk, and 0.76 on OptDigits. This beats CHAODA on Cardio, ties on Lympho and Musk, and is outperformed by CHAODA on OptDigits. We considered Clustering with Outlier Removal [30] but we could not find a working implementation, and the authors did not report AUC scores, instead only reporting F-measure. We considered comparisons against REPEN [34] and RDP [48], but REPEN's publicly available source code lacks information about dependencies and their versions, and training RDP took prohibitively long.

3.1   SDSS-APOGEE2

We demonstrate the ability of CLAM and CHAODA to scale to the next generation of "Big-Data" problems. As a proof of concept, we consider the APOGEE2 data. This dataset contains spectra of a large number of stars collected, so far, during the SDSS project [6]. We extracted $528, 319$ spectra in $8, 575$ dimensions and used CHAODA, under the $L1$-norm and the $L2$-norm, to produce anomaly scores. Since there is no ground-truth available, we simply report the scores and the time taken in the Supplement.

4   UMAP Visualization

A visualization in Figure 5 using UMAP illustrates three different examples; the anomalies in the Cardio dataset, where CHAODA outperforms other methods, appear to be at the edges of a complex manifold (though, clearly, the UMAP projection has distorted the manifold). In the Musk dataset, where many methods including CHAODA achieve perfect performance, there are several distinct components to the manifold, likely corresponding to different digits. In the Pima

Table 2: Performance (ROC AUC) of CHAODA vs. other methods on the 18 test datasets.

MODEL	ARRH	BREASTW	CARDIO	COVER	GLASS	HTTP	IONO.	LYMPHO	MAMMO
CHAODA-FAST	**0.79**	0.78	**0.86**	0.58	**0.80**	**0.99**	0.72	**0.96**	**0.85**
CHAODA	0.76	**0.94**	0.82	**0.82**	0.71	**1.00**	**0.88**	0.99	**0.86**
ABOD	0.62	0.50	0.49	0.51	0.53	0.50	0.85	0.80	0.50
AUTOENCODER	0.65	0.91	0.74	0.52	0.54	0.51	0.65	0.83	0.51
CBLOF	0.70	0.83	0.57	*EX*	0.54	*EX*	0.86	0.83	0.50
COF	0.65	0.26	0.50	0.50	0.59	0.51	0.81	0.83	0.51
HBOS	0.65	0.93	0.58	0.49	0.48	0.51	0.36	0.91	0.50
IFOREST	0.72	0.91	0.69	0.50	0.54	0.53	0.77	0.83	0.59
KNN	0.68	0.84	0.51	0.51	0.54	0.51	**0.90**	0.83	0.51
LMDD	0.68	0.64	0.60	0.49	0.54	0.51	0.67	0.65	0.56
LOCI	0.62	*TO*	*TO*	*TO*	0.58	*TO*	0.58	0.90	*TO*
LODA	0.65	0.93	0.60	0.52	0.48	0.51	0.63	0.48	0.52
LOF	0.67	0.30	0.49	0.50	0.54	0.51	0.79	0.83	0.53
MCD	0.65	0.94	0.55	0.50	0.48	0.50	**0.90**	0.83	0.51
MOGAAL	0.42	0.40	0.45	*TO*	0.59	*TO*	0.36	0.48	*TO*
OCSVM	0.70	0.77	0.70	0.56	0.54	0.50	0.68	0.83	0.60
SOD	0.59	0.77	0.48	*TO*	0.54	*TO*	0.84	0.65	0.51
SOGAAL	0.48	0.30	0.45	0.61	0.59	0.51	0.36	0.48	0.50
SOS	0.51	0.50	0.50	*TO*	0.48	*TO*	0.72	0.48	*TO*
VAE	0.65	**0.95**	0.74	0.52	0.48	0.51	0.65	0.83	0.56
MODEL	MUSK	OPTDIGITS	PIMA	SATIMG-2	SMTP	VERT	VOWELS	WBC	WINE
CHAODA-FAST	**1.00**	0.57	0.57	**0.98**	**0.96**	0.29	**0.83**	0.93	**1.00**
CHAODA	**1.00**	**0.96**	0.60	**1.00**	0.95	0.29	**0.90**	**0.97**	**0.99**
ABOD	0.47	0.54	0.60	0.53	0.50	0.49	0.75	0.50	0.43
AUTOENCODER	0.63	0.48	0.57	0.71	0.50	0.49	0.51	0.77	0.51
CBLOF	**1.00**	0.52	**0.64**	0.90	0.50	0.49	0.52	0.82	0.46
COF	0.53	0.52	0.54	0.56	0.50	0.51	0.71	0.47	0.46
HBOS	**1.00**	0.60	0.55	0.49	0.68	0.47	0.56	0.77	0.57
IFOREST	0.97	0.50	**0.65**	0.94	0.50	0.45	0.63	0.72	0.51
KNN	0.51	0.51	0.60	0.61	0.53	0.47	0.72	0.51	0.47
LMDD	0.48	0.49	0.37	0.49	0.65	0.43	0.49	0.80	0.62
LOCI	*TO*	*TO*	*TO*	*TO*	*TO*	0.49	*TO*	0.72	0.46
LODA	0.54	0.51	0.62	0.69	0.57	0.43	0.51	0.82	0.57
LOF	0.50	0.53	0.55	0.55	0.50	0.49	0.69	0.50	0.46
MCD	0.97	0.48	**0.66**	0.61	0.50	0.45	0.63	0.60	0.46
MOGAAL	0.48	0.48	0.61	0.49	*TO*	0.51	0.48	0.60	0.46
OCSVM	0.48	0.49	0.56	0.84	0.50	0.49	0.50	0.82	0.46
SOD	0.51	0.51	0.56	0.58	*TO*	0.45	0.66	0.60	0.46
SOGAAL	0.48	0.52	0.48	0.49	0.62	**0.54**	0.48	0.47	0.46
SOS	0.52	0.52	0.51	0.52	*TO*	0.49	0.59	0.52	0.46
VAE	0.63	0.48	0.61	0.71	0.50	0.45	0.51	0.77	0.67

dataset, all methods perform fairly poorly, the anomalies appear to be distributed across the manifold, including in the interior.

5 Discussion

We have presented CHAODA, an ensemble of six algorithms that use the map of the underlying manifold produced by CLAM. The six individual algorithms are simple to implement on top of CLAM and, when combined into an ensemble, often outperform state-of-the-art methods. In future work, new ideas for additional algorithms that contribute novel inductive biases can be easily incorporated into CHAODA.

While the meta-ml models in CHAODA optimize for the best ROC-AUC scores for anomaly detection, this approach can be used to optimize for any measure on any type of task. Future work should explore tasks other than anomaly detection and come up with mathematical measures for performance on those tasks. Variants of CHAODA could then be trained for such tasks.
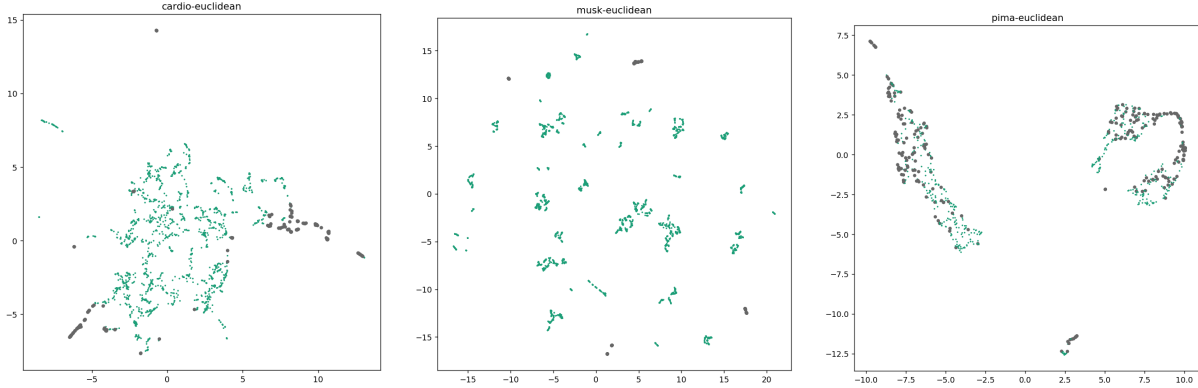
Figure 5: UMAP projections of Cardio (left), Musk (middle), and Pima (right) under Euclidean distance. For Cardio, there is a single main component to the manifold, and anomalies tend to be at the edges of that manifold. For Musk, there are several distinct pieces to the manifold, perhaps corresponding to different digits. CHAODA outperforms other approaches on Cardio, while many approaches achieve perfect performance on Musk. On Pima, all approaches fare poorly, and the UMAP projection illustrates that the anomalies cover much of the manifold, including in the interior.

Cross-validation on a single dataset is commonplace in machine learning. While k-fold cross-validation splits a single dataset into k folds and then evaluates performance on each fold separately, these folds are still drawn from the same underlying distribution; indeed, this is the principle that justifies cross-validation. In contrast, CHAODA's meta-ml approach learns geometric and topological properties for inducing graphs from a *set* of training datasets, and *transfers* that knowledge to an *entirely distinct* set of datasets, which may differ in cardinality, dimensionality, domain, and the generating phenomenon or process that produced the data.

CLAM uses the geometric and topological properties such as fractal dimension of the data to build a map of the low-dimensional manifold that the data occupy. CLAM extends CHESS [22] by: a better selection of "poles" (i.e. a pair of well-separated points) for partitioning clusters, memoizing important cluster properties, and introducing a novel graph-induction approach using a notion of optimal depths, learned via a form of "meta-machine-learning" and transfer learning. Whereas CHESS was developed specifically for accelerating search, CHAODA uses this manifold-mapping framework to discover properties of the manifold that are useful for anomaly detection. Intuitively, we expect CHAODA to perform particularly well when the data lie on an "interesting" manifold, and to perform merely average when the data derive from an easily-described distribution (or "boring" manifold). Just as CHESS demonstrated an acceleration of search when the data exhibited *low fractal dimension* and *low metric entropy*, we see that CHAODA vastly improves ROC scores when the data exhibit these properties. When the data do not exhibit these properties, CHAODA is still often competitive with other approaches. CLAM is free of hyper-parameters other than the fairly standard choice of α in Section 2.5; the weights learned from the meta-ml step could vary, but we learned them once on a distinct training set of datasets.

We briefly discuss the Vertebral Column (Vert.) dataset, with regards to the robustness of our approach. On this dataset, most algorithms we evaluated performed similarly to random guessing, while CHAODA performed much worse. We suspect this is due to how this specific dataset was collected. Each instance represents six biomechanical attributes derived from scans of a patient's pelvis and lumbar spine. This dataset contains 210 instances of the Abnormal class treated as inliers and 30 instances of the Normal class treated as outliers. Each attribute must have a narrow range to be in the Normal class, but can have a wider range in the Abnormal class. This causes the Normal instances to group together, while Abnormal instances remain distant from each other. As CHAODA relies on clusters as the substrate, it assigns low scores to instances in the Normal class, i.e. the outliers, and high scores to those in the Abnormal class, i.e. the inliers. Put plainly, CHAODA sees the manifold as the Normal class, which the ground-truth labels as outliers.

The choice of distance function could significantly impact anomaly-detection performance. In this case, domain knowledge is likely the best way to determine the distance function of choice. Future work should explore a more diverse collection of domain-appropriate distance functions, such as Wasserstein distance on images, Levenshtein distance on strings, and Jaccard distance on the maximal common subgraph of molecular structures. Currently, CLAM only functions on a metric space defined by a distance metric (it is not, however, limited to complete Banach spaces). This poses a limitation on datasets that have heterogenous features, such as a mix of continuous and categorical variables. Future work should explore linear combinations of normalized distance functions to overcome this limitation. Additionally, we do not yet know how CHAODA would generalize across distance functions; i.e. predicting

13

anomalousness with distance functions different from those used to train the meta-ml models. Future work should investigate this question of generalization. It would also be worth exploring the question of whether CHAODA extends to nondeterministic distance functions, as well as performance on other non-metric distance functions, such as those disobeying the triangle inequality (e.g. cosine distance or Damerau-Levenshtein edit distance [11]).

In this paper, we have used CHAODA (and the methods under comparison) to score entire datasets with known anomaly labels for purposes of evaluating CHAODA's accuracy. In real-world usage, one might wish to assign anomaly scores to an incoming data stream. This is a simple extension: given some corpus of data (some of which may or may not be anomalous), build a CLAM tree and the induced graphs, and assign anomaly scores from the CHAODA algorithms as we have demonstrated. Then, as each new datum arrives, simply fit it into the CLAM tree ($\mathcal{O}(\lg|V|)$ time using tree-search from CHESS) into a cluster that is found in a graph and assign it the anomaly score for that cluster. If an incoming datum is too far from any cluster (further than any existing datum at that depth from its cluster center) then it can initialize a new cluster, which would be assigned a high anomaly score. Thus, in general, CHAODA requires $\mathcal{O}(\lg|V|)$ time to assign an anomaly score to a new datum.

CHAODA is demonstrably highly effective on large high-dimensional datasets, and so may be applied to neural networks. Using CLAM to map a dataset where each datum represents the activation-pattern of a neural network from an input to the neural network, we would expect to detect malicious inputs to neural networks based on the intuition that malicious inputs produce atypical activation patterns.

In conclusion, we have demonstrated that by mapping the manifolds occupied by data, CLAM reveals geometric and topological structure that allows CHAODA to outperform other state-of-the-art approaches to anomaly detection, representing an actualization of the manifold hypothesis.

Supplementary results and figures are available at `https://homepage.cs.uri.edu/~ndaniels/pdfs/chaoda-supplement.pdf`. The source code for CLAM and CHAODA are available under an MIT license at `https://github.com/URI-ABD/clam`.

References

[1] AGRAWAL, R., GEHRKE, J., GUNOPULOS, D., AND RAGHAVAN, P. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data* (1998), pp. 94–105.

[2] ALMAN, J., AND WILLIAMS, V. V. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2021), SIAM, pp. 522–539.

[3] ANGIULLI, F., AND PIZZUTI, C. Fast outlier detection in high dimensional spaces. In *Principles of Data Mining and Knowledge Discovery* (Berlin, Heidelberg, 2002), T. Elomaa, H. Mannila, and H. Toivonen, Eds., Springer Berlin Heidelberg, pp. 15–27.

[4] ARNING, A., AGRAWAL, R., AND RAGHAVAN, P. A linear method for deviation detection in large databases. In *KDD* (1996), vol. 1141, pp. 972–981.

[5] BANACH, S. Sur les fonctionnelles linéaires ii. *Studia Mathematica 1* (1929), 223–239.

[6] BLANTON, M. R., BERSHADY, M. A., ABOLFATHI, B., ALBARETI, F. D., ALLENDE PRIETO, C., ALMEIDA, A., ALONSO-GARCÍA, J., ANDERS, F., ANDERSON, S. F., ANDREWS, B., AND ET AL. Sloan Digital Sky Survey IV: Mapping the Milky Way, Nearby Galaxies, and the Distant Universe. *The Astronomical Journal 154* (July 2017), 28.

[7] BREUNIG, M. M., KRIEGEL, H.-P., NG, R. T., AND SANDER, J. Lof: Identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2000), SIGMOD '00, Association for Computing Machinery, p. 93–104.

[8] CHANDOLA, V., BANERJEE, A., AND KUMAR, V. Anomaly detection: A survey. *ACM computing surveys (CSUR) 41*, 3 (2009), 1–58.

[9] CHEN, J., SATHE, S., AGGARWAL, C., AND TURAGA, D. Outlier detection with autoencoder ensembles. In *Proceedings of the 2017 SIAM international conference on data mining* (2017), SIAM, pp. 90–98.

[10] COPPERSMITH, D., AND WINOGRAD, S. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation 9*, 3 (1990), 251 – 280. Computational algebraic complexity editorial.

[11] DAMERAU, F. J. A technique for computer detection and correction of spelling errors. *Communications of the ACM 7*, 3 (1964), 171–176.

[12] DUA, D., AND GRAFF, C. UCI machine learning repository, 2017.

[13] ELSAYED, G., SHANKAR, S., CHEUNG, B., PAPERNOT, N., KURAKIN, A., GOODFELLOW, I., AND SOHL-DICKSTEIN, J. Adversarial examples that fool both computer vision and time-limited humans. In *Advances in Neural Information Processing Systems* (2018), pp. 3910–3920.

[14] ESTER, M., KRIEGEL, H.-P., SANDER, J., XU, X., ET AL. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd* (1996), vol. 96, pp. 226–231.

[15] FAWCETT, T. An introduction to roc analysis. *Pattern recognition letters 27*, 8 (2006), 861–874.

[16] FEFFERMAN, C., MITTER, S., AND NARAYANAN, H. Testing the manifold hypothesis. *Journal of the American Mathematical Society 29*, 4 (2016), 983–1049.

[17] GOLDSTEIN, M., AND DENGEL, A. Histogram-based outlier score (hbos): A fast unsupervised anomaly detection algorithm. *KI-2012: Poster and Demo Track* (2012), 59–63.

[18] GUHA, S., RASTOGI, R., AND SHIM, K. Cure: an efficient clustering algorithm for large databases. *ACM Sigmod record 27*, 2 (1998), 73–84.

[19] HARDIN, J., AND ROCKE, D. Outlier detection in the multiple cluster setting using the minimum covariance determinant estimator. *Computational Statistics and Data Analysis 44*, 4 (Jan. 2004), 625–638.

[20] HE, Z., XU, X., AND DENG, S. Discovering cluster-based local outliers. *Pattern Recognition Letters 24*, 9 (2003), 1641 – 1650.

[21] HINNEBURG, A., KEIM, D. A., ET AL. An efficient approach to clustering in large multimedia databases with noise. In *KDD* (1998), vol. 98, pp. 58–65.

[22] ISHAQ, N., STUDENT, G., AND DANIELS, N. M. Clustered hierarchical entropy-scaling search of astronomical and biological data. In *2019 IEEE International Conference on Big Data (Big Data)* (2019), IEEE, pp. 780–789.

[23] KARYPIS, G., HAN, E.-H., AND CHAMELEON, V. K. A hierarchical clustering algorithm using dynamic modeling. *IEEE Computer 32*, 8 (1999), 68–75.

[24] KAUFMAN, L., AND ROUSSEEUW, P. J. *Finding groups in data: an introduction to cluster analysis*, vol. 344. John Wiley & Sons, 2009.

[25] KRIEGEL, H.-P., KRÖGER, P., SCHUBERT, E., AND ZIMEK, A. Outlier detection in axis-parallel subspaces of high dimensional data. In *Advances in Knowledge Discovery and Data Mining* (Berlin, Heidelberg, 2009), T. Theeramunkong, B. Kijsirikul, N. Cercone, and T.-B. Ho, Eds., Springer Berlin Heidelberg, pp. 831–838.

[26] KRIEGEL, H.-P., KROGER, P., SCHUBERT, E., AND ZIMEK, A. Interpreting and unifying outlier scores. In *Proceedings of the 2011 SIAM International Conference on Data Mining* (2011), SIAM, pp. 13–24.

[27] LEVIN, D. A., AND PERES, Y. *Markov chains and mixing times*, vol. 107. American Mathematical Soc., 2017.

[28] LIU, F. T., TING, K. M., AND ZHOU, Z.-H. Isolation forest. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining* (USA, 2008), ICDM '08, IEEE Computer Society, p. 413–422.

[29] LIU, F. T., TING, K. M., AND ZHOU, Z.-H. Isolation-based anomaly detection. *ACM Trans. Knowl. Discov. Data 6*, 1 (Mar. 2012).

[30] LIU, H., LI, J., WU, Y., AND FU, Y. Clustering with outlier removal. *IEEE Transactions on Knowledge and Data Engineering* (2019).

[31] LIU, Y., LI, Z., ZHOU, C., JIANG, Y., SUN, J., WANG, M., AND HE, X. Generative adversarial active learning for unsupervised outlier detection. *IEEE Transactions on Knowledge and Data Engineering 32*, 8 (2019), 1517–1528.

[32] MACQUEEN, J., ET AL. Some methods for classification and analysis of multivariate observations. *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability 1*, 14 (1967), 281–297.

[33] NG, R., AND HAN, J. Efficient and effective clustering methods for spatial data. In *Proc_of the Int. Conf. on Very Large Data Bases, Santiago, Chile* (1994).

[34] PANG, G., CAO, L., CHEN, L., AND LIU, H. Learning representations of ultrahigh-dimensional data for random distance-based outlier detection. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2018), pp. 2041–2050.

[35] PANG, G., SHEN, C., CAO, L., AND HENGEL, A. V. D. Deep learning for anomaly detection: A review. *ACM Computing Surveys (CSUR) 54*, 2 (2021), 1–38.

[36] PAPADIMITRIOU, S., KITAGAWA, H., GIBBONS, P. B., AND FALOUTSOS, C. Loci: fast outlier detection using the local correlation integral. *Proceedings 19th International Conference on Data Engineering (Cat. No.03CH37405)* (2003), 315–326.

[37] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., MICHEL, V., THIRION, B., GRISEL, O., BLONDEL, M., PRETTENHOFER, P., WEISS, R., DUBOURG, V., ET AL. Scikit-learn: Machine learning in python. *the Journal of machine Learning research 12* (2011), 2825–2830.

[38] PEVNÝ, T. Loda: Lightweight on-line detector of anomalies. *Machine Learning 102*, 2 (Feb 2016), 275–304.

[39] RAMASWAMY, S., RASTOGI, R., AND SHIM, K. Efficient algorithms for mining outliers from large data sets. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data* (2000), pp. 427–438.

[40] RAYANA, S. ODDS library, 2016.

[41] ROUSSEEUW, P., AND DRIESSEN, K. A fast algorithm for the minimum covariance determinant estimator. *Technometrics 41* (08 1999), 212–223.

[42] SATHE, S., AND AGGARWAL, C. C. Subspace outlier detection in linear time with randomized hashing. In *2016 IEEE 16th International Conference on Data Mining (ICDM)* (2016), IEEE, pp. 459–468.

[43] SCHÖLKOPF, B., PLATT, J. C., SHAWE-TAYLOR, J., SMOLA, A. J., AND WILLIAMSON, R. C. Estimating the support of a high-dimensional distribution. *Neural Computation 13*, 7 (2001), 1443–1471.

[44] SHEIKHOLESLAMI, G., CHATTERJEE, S., AND ZHANG, A. Wavecluster: a wavelet-based clustering approach for spatial data in very large databases. *The VLDB Journal 8*, 3-4 (2000), 289–304.

[45] TANG, J., CHEN, Z., FU, A. W.-C., AND CHEUNG, D. W.-L. Enhancing effectiveness of outlier detections for low density patterns. In *Proceedings of the 6th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining* (Berlin, Heidelberg, 2002), PAKDD '02, Springer-Verlag, p. 535–548.

[46] VORONOI, G. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. premier mémoire. sur quelques propriétés des formes quadratiques positives parfaites. *Journal für die reine und angewandte Mathematik 1908*, 133 (1908), 97–102.

[47] WANG, H., BAH, M. J., AND HAMMAD, M. Progress in outlier detection techniques: A survey. *IEEE Access 7* (2019), 107964–108000.

[48] WANG, H., PANG, G., SHEN, C., AND MA, C. Unsupervised representation learning by predicting random distances. *arXiv preprint arXiv:1912.12186* (2019).

[49] WANG, W., YANG, J., MUNTZ, R., ET AL. Sting: A statistical information grid approach to spatial data mining. In *VLDB* (1997), vol. 97, pp. 186–195.

[50] ZAHN, C. T. Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Transactions on computers 100*, 1 (1971), 68–86.

[51] ZHANG, J. Advancements of outlier detection: A survey. *ICST Transactions on Scalable Information Systems 13*, 1 (2013), 1–26.

[52] ZHAO, Y., NASRULLAH, Z., AND LI, Z. Pyod: A python toolbox for scalable outlier detection. *Journal of Machine Learning Research 20*, 96 (2019), 1–7.

[53] ZONG, B., SONG, Q., MIN, M. R., CHENG, W., LUMEZANU, C., CHO, D., AND CHEN, H. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *International Conference on Learning Representations* (2018).

6 Datasets

Here we describe the datasets we use for benchmarks. See Table S3 for a summary of this information.

The **annthyroid** dataset is derived from the "Thyroid Disease" dataset from the UCIMLR. The original data has 7200 instances with 15 categorical attributes and 6 real-valued attributes. The class labels are "normal", "hypothyroid", and "subnormal". For anomaly detection, the "hypothyroid" and "subnormal" classes are combined into 534 outlier instances, and only the 6 real-valued attributes are used.

The **arrhythmia** dataset is derived from the "Arrhythmia" dataset from the UCIMLR. The original dataset contains 452 instances with 279 attributes. There are five categorical attributes which are discarded, leaving this as a 274-dimensional dataset. The instances are divided into 16 classes. The eight smallest classes collectively contain 66 instances and are combined into the outlier class.

The **breastw** dataset is also derived from the "Breast Cancer Wisconsin (Original)" dataset. This is a 9-dimensional dataset containing 683 instances of which 239 represent malignant tumors and are treated as the outlier class.

The **cardio** dataset is derived from the "Cardiotocography" dataset. The dataset is composed of measurements of fetal heart rate and uterine contraction features on cardiotocograms. The are each labeled "normal", "suspect", and "pathologic" by expert obstetricians. For anomaly detection, the "normal" class forms the inliers, the "suspect" class is discarded, and the "pathologic" class is downsampled to 176 instances forming the outliers. This leaves us with 1831 instances with 21 attributes in the dataset.

The **cover** dataset is derived from the "Covertype" dataset. The original dataset contains 581,012 instances with 54 attributes. The dataset is used to predict the type of forest cover solely from cartographic variables. The instances are labeled into seven different classes. For outlier detection, we use only the 10 quantitative attributes as the features. We treat class 2 (lodgepole pine) as the inliers, and class 4 (cottonwood/willow) as the outliers. The remaining classes are discarded. This leaves us with a 10-dimensional dataset with 286,048 instances of which 2,747 are outliers.

The **glass** dataset is derived from the "Glass Identification" dataset. The study of classification of types of glass was motivated by criminological investigations where glass fragments left at crime scenes were used as evidence. This dataset contains 214 instances with 9 attributes. While there are several different types of glass in this dataset, class 6 is a clear minority with only 9 instances and, as such, points in class 6 are treated as the outliers while all other classes are treated as inliers.

The **http** dataset is derived from the original "KDD Cup 1999" dataset. It contains 41 attributes (34 continuous and 7 categorical) which are reduced to 4 attributes (service, duration, src_bytes, dst_bytes). Only the "service" attribute is categorical, dividing the data into {http, smtp, ftp, ftp_data, others} subsets. Here, only the "http" data is used. The values of the continuous attributes are centered around 0, so they have been log-transformed far away from 0. The original data contains 3,925,651 attacks in 4,898,431 records. This smaller dataset is created with only 2,211 attacks in 567,479 records.

The **ionosphere** dataset is derived from the "Ionosphere" dataset. It consists of 351 instances with 34 attributes. One of the attributes is always 0 and, so, is discarded, leaving us with a 33-dimensional dataset. The data come from radar measurements of the ionosphere from a system located in Goose Bay, Labrador. The data are classified into "good" if the radar returns evidence of some type of structure in the ionosphere, and "bad" otherwise. The "good" class serves as the inliers and the "bad" class serves as the outliers.

The **lympho** dataset is derived from the "Lymphography" dataset. The data contain 148 instances with 18 attributes. The instances are labeled "normal find", "metastases", "malign lymph", and "fibrosis". The two minority classes only contain a total of six instances, and are combined to form the outliers. The remaining 142 instances form the inliers.

The **mammography** dataset is derived from the original "Mammography" dataset provided by Aleksandar Lazarevic. Its goal is to use x-ray images of human breasts to find calcified tissue as an early sign of breast cancer. As such, the "calcification" class is considered as the outlier class while the "non-calcification" class is the inliers. We have 11,183 instances with 6 attributes, of which 260 are "calcifications."

The **mnist** dataset is derived from the classic "MNIST" dataset of handwritten digits. Digit-zero is considered the inlier class while 700 images of digit-six are the outliers. Furthermore, 100 pixels are randomly selected as features from the original 784 pixels.

The **musk** dataset is derived from its namesake in the UCIMLR. It is created from molecules that have been classified by experts as "musk" or "non-musk". The data are downsampled to 3,062 instances with 166 attributes. The "musk" class forms the outliers while the "non-musk" class forms the inliers.

The **optdigits** dataset is derived from the "Optical Recognition of Handwritten Digits" dataset. Digits 1–9 form the inliers while 150 samples of digit-zero form the outliers. This gives us a dataset of 5,216 instances with 64 attributes.

The **pendigits** dataset is derived from the "Pen-Based Recognition of Handwritten Digits" dataset from the UCI Machine Learning Repository. The original collection of handwritten samples is reduced to 6,870 points, of which 156 are outliers.

The **pima** dataset is derived from the "Pima Indians Diabetes" dataset. The original dataset presents a binary classification problem to detect diabetes. This subset was restricted to female patients at least 21 years old of Pima Indian heritage.

The **satellite** dataset is derived from the "Statlog (Landsat Satellite)" dataset. The smallest three classes (2, 4, and 5) are combined to form the outlier class while the other classes are combined to form the inlier class. The train and test subsets are combined to produce a of 6,435 instances with 36 attributes.

The **satimage-2** dataset is also derived from the "Satlog (Landsat Satellite)" dataset. Class 2 is downsampled to 71 instances that are treated as outliers, while all other classes are combined to form an inlier class. This gives us 5,803 instances with 36 attributes.

The **shuttle** dataset is derived from the "Statlog (Shuttle)" dataset. There are seven classes in the original dataset. Here, class 4 is discarded, class 1 is treated as the inliers and the remaining classes, which are comparatively small, are combined into an outlier class. This gives us 49,097 instances with 9 attributes, of which 3,511 are outliers.

The **smtp** is also derived from the "KDD Cup 1999" dataset. It is pre-processed in the same way as the **http** dataset, except that the "smtp" service subset is used. This version of the dataset only contains 95,156 instances with 3 attributes, of which 30 instances are outliers.

The **thyroid** dataset is also derived from the "Thyroid Disease" dataset. The attribute selection is the same as for the **annthyroid** dataset but only the 3,772 training instances are used in this version. The "hyperfunction" class, containing 93 instances, is treated as the outlier class, while the other two classes are combined to form an inlier class.

The **vertebral** dataset is derived from the "Vertebral Column" dataset. 6 attributes are derived to represent the shape and orientation of the pelvis and lumbar spine. These attributes are: pelvic incidence, pelvic tilt, lumbar lordosis angle, sacral slope, pelvic radius and grade of spondylolisthesis. Each instance comes from a different patient. The "Abnormal (AB)" class of 210 instances are used as inliers while the "Normal (NO)" class is downsampled to 30 instances to be used as outliers.

The **vowels** dataset is derived from the "Japanese Vowels" dataset. The UCIMLR presents this data as a multivariate time series of nine speakers uttering two Japanese vowels. For outlier detection, each frame of each time-series is treated as a separate point. There are 12 features associated with each time series, and these translate as the attributes for each point. Data from speaker 1, downsampled to 50 points, form the outlier class. Speakers 6, 7, and 8 form the inlier class. The rest of the points are discarded. This leaves is with 1,456 points in 12 dimensions, of which 50 are outliers.

The **wbc** dataset is derived from the "Wisconsin-Breast Cancer (Diagnostics)" dataset. The dataset records measurements for breast cancer cases. The benign class is treated as the inlier class, while the malignant class is downsampled to 21 points and serves as the outlier class. This leaves us with 278 points in 30 dimensions.

The **wine** dataset is a collection of results of a chemical analysis of several wines from a region in Italy. The data contain 129 samples having 13 attributes, and divided into 3 classes. Classes 2 and 3 form the inliers while class 1, downsampled to 10 instances, is the outlier class.

7 Complexity of CHAODA

Here we provide short proofs for the time complexity and space complexity of the CHAODA algorithms. For each algorithm, we have a dataset X with $n = |X|$ points and a graph $G(V, E)$ of clusters/vertices V and edges E between overlapping clusters.

7.1 CLAM Clustering

We use CLAM to build the cluster-tree and the induced graphs. The time complexity of clustering is the same as for clustering in CHESS [22]; i.e., expected $\mathcal{O}(n \log n)$ and worst-case $\mathcal{O}(n^2)$ where n is the size of the dataset.

The cost for inducing graphs depends on whether it is a layer-graph or an optimal graph. For both types of graphs, we first have to select the right clusters, and then find neighbors based on cluster overlap.

Table S3: Datasets used for Benchmarks

Dataset	Cardinality	# Dim.	# Outliers	% Outliers
annthyroid	7,200	6	534	7.42
arrhythmia	452	274	66	15
breastw	683	9	239	35
cardio	1,831	21	176	9.6
cover	286,048	10	2,747	0.9
glass	214	9	9	4.2
http	567,479	4	2,211	0.4
ionosphere	351	33	126	36
lympho	148	18	6	4.1
mammography	11,183	6	260	2.32
mnist	7603	100	700	9.2
musk	3,062	166	97	3.2
optdigits	5,216	64	150	3
pendigits	6,870	16	156	2.27
pima	768	8	268	35
satellite	6,435	36	2036	32
satimage-2	5,803	36	71	1.2
shuttle	59,097	9	3,511	7
smtp	95,156	3	30	0.03
thyroid	3,772	6	93	2.5
vertebral	240	6	30	12.5
vowels	1,456	12	50	3.4
wbc	278	30	21	5.6
wine	129	13	10	7.7

We implemented CLAM in Python and the language does not have tail-call optimization for recursive functions. Therefore we implement partition to, instead of recursing until reaching leaves, iteratively increase the depth of the tree. During the course of this partition, we store a map from tree-depth to a set of clusters at that depth. Therefore, selecting all cluster at a given depth costs $\mathcal{O}(1)$ time and $\mathcal{O}(|V|)$ space where V is the set of selected clusters. Selecting clusters for optimal-graphs is more expensive. First, we use a trained meta-ml model to predict the AUC contribution from each cluster in a tree; this costs $\mathcal{O}(n)$ time and $\mathcal{O}(n)$ space. Next, we sort the clusters by this predicted value; this costs $\mathcal{O}(n\log n)$ time and $\mathcal{O}(n)$ space. Finally, we perform a linear pass over the clusters to select the best for the graph, while discarding the ancestors and descendants of any cluster that has already been selected; this costs $\mathcal{O}(n)$ time and $\mathcal{O}(|V|)$ space. Therefore, the total cost of selecting clusters for optimal graphs is $\mathcal{O}(n\log n)$ time and $\mathcal{O}(n)$ space.

Once the clusters have been selected for a graph, we have to find every pair of clusters with overlapping volumes. Naïvely, this can be done with an all-pairs distance computation for a cost of $\mathcal{O}(|V|^2)$ for time and space. However, our implementation is superior to the naïve method although the proof is beyond the scope of this supplement.

7.2 Relative Cluster Cardinality

This algorithm performs a single linear pass over the vertices in the graph. The cardinalities of clusters are cached during the tree-building phase of clam. Each lookup from this cache costs $\mathcal{O}(1)$. For a graph $G(V, E)$ the time-complexity is trivially $\mathcal{O}(|V|)$. Since each cluster stores its cardinality, the space complexity is also $\mathcal{O}(|V|)$.

7.3 Relative Component Cardinality

This method first finds the components of the graph. This costs $\mathcal{O}(|E|)$ time because we have to check each edge once. The cardinality of each component is cached when traversing the clusters to find components, thus the space complexity

is $\mathcal{O}(|C|)$ where C is the set of distinct connected components in the graph. With this done, the algorithm performs a single linear pass over each component. This brings the total worst-case cost to $\mathcal{O}(|E| + |V|)$.

7.4 Graph Neighborhood

This algorithm performs a linear pass over the clusters in the graph and first computes the eccentricity of each cluster. Finding the eccentricity of a vertex in a graph is worst-case $\mathcal{O}(|E|)$ time when the graph consists of a single component. This brings the total cost up to $\mathcal{O}(|E| \cdot |V|)$, with space complexity $\mathcal{O}(|V| + |E|)$. Next, the algorithm performs a traversal from each cluster. This adds a constant factor of $\mathcal{O}(|E|)$ to the time complexity and $\mathcal{O}(|V|)$ to the space complexity, which can be ignored. The total time-complexity of this algorithm is thus $\mathcal{O}(|E| \cdot |V|)$ and its space complexity is $\mathcal{O}(|V|)$ (because only the size of each graph-neighborhood needs to be stored).

7.5 Child-Parent Cardinality Ratio

While building the tree with CLAM, we cache the child-parent cardinality ratios of every cluster, because it proved useful for purposes other than anomaly detection. This method performs a single linear pass over the clusters in the graph and looks-up the cached child-parent ratios as needed. The time-complexity is thus $\mathcal{O}(|V|)$. Since the ratios are cached with their respective clusters, the space complexity is $\mathcal{O}(|V|)$.

7.6 Stationary Probabilities

This method starts by computing a transition matrix for each component in the graph. We set the transition probability from a cluster to a neighbor to be inversely proportional to the distance between their centers, normalized by all possible neighbors of the cluster. We successively square this matrix until it converges. The transition matrices from our graphs obey the criteria required for convergence as proven in [27]. Matrix multiplication for square matrices costs $\mathcal{O}(|V|^{2.373})$ with the Coppersmith-Winograd algorithm [10]. Thus the worst-case time complexity is the same as that for the matrix-multiplication algorithm employed. For space, we need only store a single $|V| \times |V|$ matrix, giving us a space complexity of $\mathcal{O}(|V|^2)$.

In practice, $|V| \ll n$ and graphs only rarely consist of only one component. Thus, the average run-time performance is much better than that suggested by the quadratic space time-complexity.

7.7 Vertex Degree

Since we already have a graph with vertices and edges, calculating the degree of each vertex only costs $\mathcal{O}(1)$ time. Thus, the complexity of this algorithm is $\mathcal{O}(|V|)$.

7.8 Normalization

Normalizing the outlier scores requires finding the mean and standard deviation of the raw scores, followed by a linear pass over the set of scores. Thus the time-complexity of this step is $\mathcal{O}(n)$. Since we need to store a score for each point, the space complexity is $\mathcal{O}(n)$.

The algorithm is presented in Algorithm 5.

Algorithm 5 Gaussian Normalization

Require: X, a dataset
Require: S, a set of outlier scores for each point in X
1: $erf : x \mapsto \frac{2}{\sqrt{\pi}} \int_0^x e^{-u^2} du$
2: $\mu \leftarrow mean(S)$
3: $\sigma \leftarrow std(S)$
4: **for** point $p \in X$ **do**
5:    $S[p] \leftarrow \frac{1}{2}\left(1 + erf\left(\frac{S[p] - \mu}{\sigma \cdot \sqrt{2}}\right)\right)$
6: **end for**

7.9 Ensemble

Given the normalized scores from the individual methods, we combine the scores by voting among them in an ensemble. There is a small, constant, number of scores for each point; each score is from a different graph built using the meta-ml models. We simply take the mean of all scores for each point. Thus the time-complexity of voting among these scores is $\mathcal{O}(n)$ for the entire dataset. Since we need to store a score for each point, the space complexity is $\mathcal{O}(n)$.

8 UMAP Visualization

A visualization in Figure S6 using UMAP illustrates a handful of different examples; the anomalies in the Cardio and OptDigits datasets, where CHAODA outperforms other methods, appear to be at the edges of a complex manifold (though, clearly, the UMAP projection has distorted the manifold). In the Mnist dataset, where several methods perform fairly well, the distribution is less interesting. Most anomalies are off to one side but there are several interspersed among the inliers.

In Figure S7, we show UMAP visualizations of the Pima dataset. The inliers and outliers seem inseparable, and so all the methods perform poorly.

9 Performance

9.1 Run-time performance on Test set of Datasets

Tables S4 and S5 report the running time, in seconds, of CHAODA and each competitor. The fastest methods on each dataset are presented in bold face.

9.2 AUC and Runtime performance on Train set of Datasets

Tables S6 and S7 report the the AUC performance and running time, respectively, of CHAODA and each competitor on the train set of datasets.

9.3 Performance of Individual CHAODA Algorithms

The ensemble of CHAODA algorithms is discussed extensively in the main paper, but we did not have room to discuss or present the performance of the individual algorithms. Due to the large numbers of graphs generated for the ensemble and with each method being applied to each graph, we cannot provide these intermediate results as a table in this document. We instead provide a .csv file which will be available for download.

9.4 Performance on APOGEE-2

For a read "Big-Data" challenge, we applied CHAODA to the APOGEE-2 dataset from the SDSS. This dataset contains spectra of stars, i.e. intensity/flux measured at several wavelengths, in $8,757$ dimensions. We extracted $528,323$ spectra from data-release 15. CHAODA was able to produce anomaly scores for these data in approximately 2 hours and 36 minutes. These scores, and the associated metadata, are provided in our github repository in the 'sdss' directory.
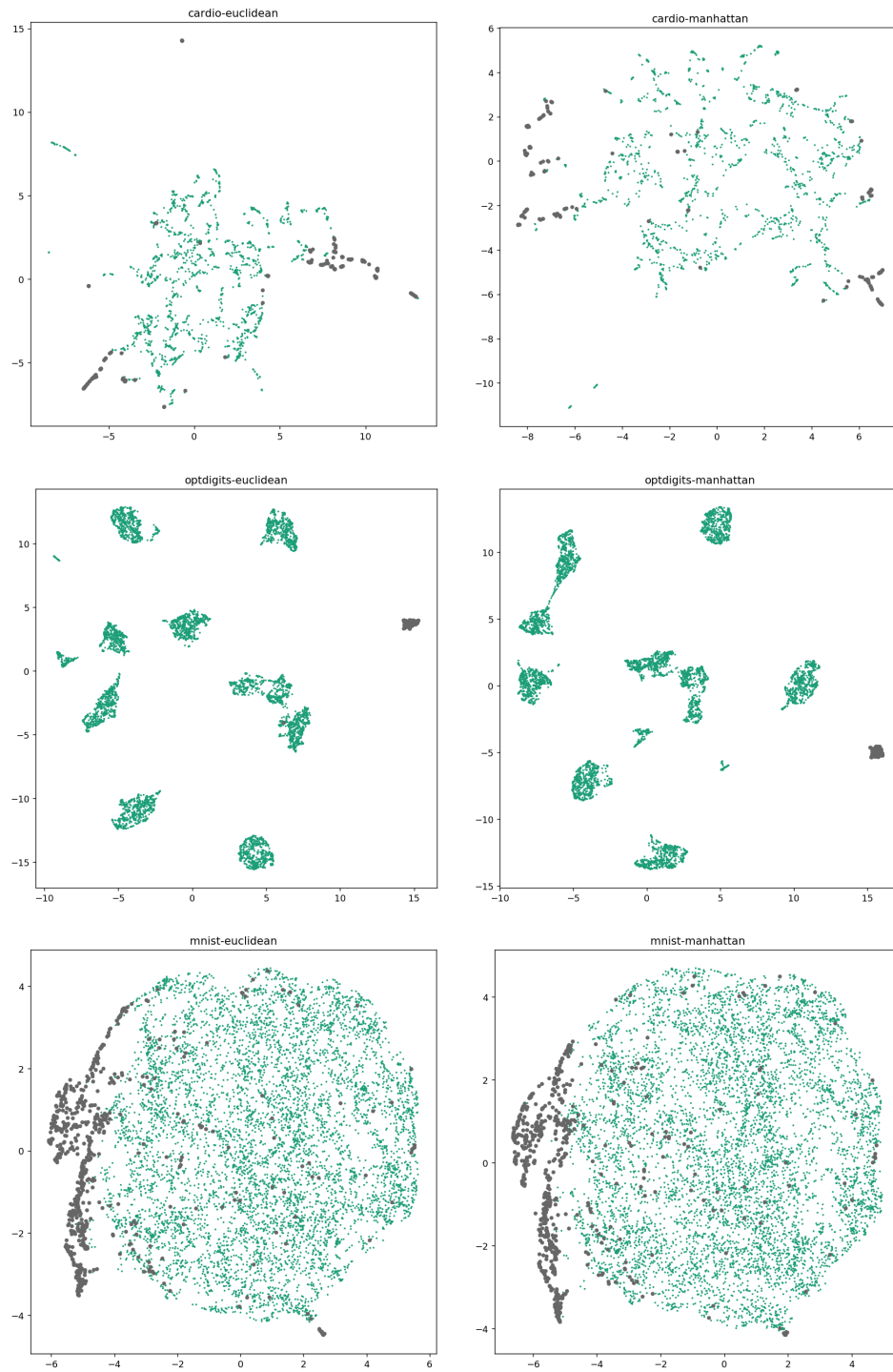
Figure S6: UMAP projections of Cardio (first row), Optdigits (second row) and Mnist (third row). The distance metrics used are Euclidean (left column) and Manhattan (right column). Anomalies are in gray. Note that for MNIST, the UMAP projection does not find much structure, though most of the anomalies congregate to one side. For Cardio, there is a single main component to the manifold, and anomalies tend to be at the edges of that manifold. For OptDigits, there are several distinct pieces to the manifold, perhaps corresponding to different digits. Most algorithms perform comparably on MNIST, while CHAODA outperforms others on Cardio and OptDigits.

Figure S7: UMAP projections of the Pima dataset. All algorithms performed poorly on Pima. This may be because the anomalies and the outliers seem inseparable in the projection.

Table S4: Time taken, in seconds, on the first half of the Test Datasets
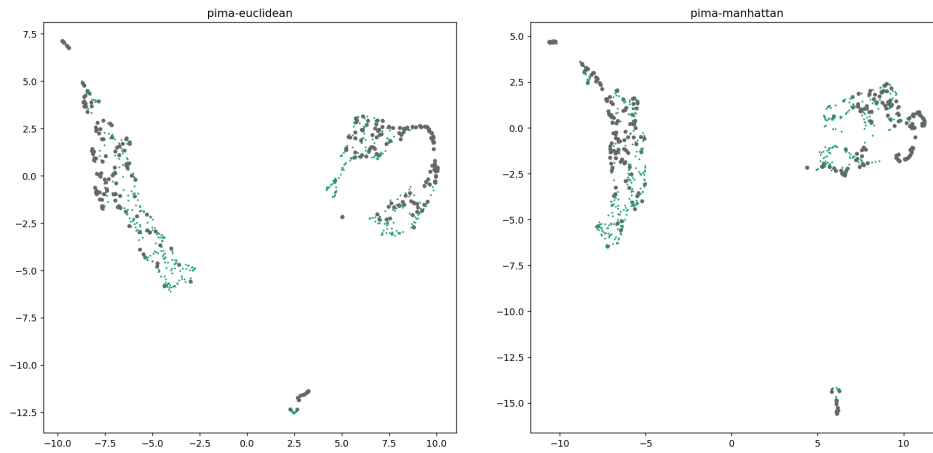
Model	Arrhy	BreastW	Cardio	Cover	Glass	Http	Iono	Lympho	Mammo
CHAODA-fast	8.06	5.01	42.37	829.62	1.73	5e3	3.71	1.78	29.92
CHAODA	27.43	14.00	344.27	6e3	7.44	2e4	22.13	1.74	244.81
ABOD	0.34	0.20	0.72	24.02	**0.07**	19.08	0.13	**0.05**	3.82
AutoEncoder	8.00	6.18	9.05	183.70	3.99	154.20	4.99	3.79	35.26
CBLOF	0.16	0.13	0.17	*EX*	**0.06**	*EX*	**0.07**	**0.05**	0.20
COF	1.24	1.84	12.63	2e4	0.24	2e5	0.60	0.14	513.46
HBOS	**0.08**	**0.00**	**0.01**	1.13	**0.00**	**0.01**	**0.01**	**0.01**	**0.01**
IFOREST	0.43	0.34	0.43	4.61	0.30	3.95	0.33	0.30	0.95
KNN	0.19	**0.07**	0.30	11.46	**0.02**	7.47	**0.05**	**0.02**	1.58
LMDD	14.85	2.62	22.12	1e4	0.57	4e3	1.73	0.42	243.01
LOCI	307.04	*TO*	*TO*	*TO*	25.68	*TO*	120.92	9.78	*TO*
LODA	**0.04**	**0.03**	**0.05**	0.81	**0.03**	0.66	**0.03**	**0.03**	0.17
LOF	0.16	**0.01**	0.18	10.24	**0.00**	1.93	**0.01**	**0.00**	0.59
MCD	5.08	0.64	0.89	28.65	**0.05**	8.21	0.15	**0.05**	2.11
MOGAAL	46.46	42.09	116.73	*TO*	40.67	*TO*	40.86	37.84	*TO*
OCSVM	**0.10**	**0.02**	0.23	257.95	**0.00**	257.28	**0.01**	**0.00**	6.37
SOD	1.19	1.89	13.39	*TO*	0.31	*TO*	0.76	0.20	521.25
SOGAAL	6.70	5.03	14.43	591.14	3.95	597.17	4.71	3.96	92.27
SOS	0.69	47.40	4.11	*TO*	0.18	*TO*	0.33	0.11	*TO*
VAE	10.00	7.75	10.89	223.09	5.27	175.59	6.66	5.21	40.87

Table S5: Time taken, in seconds, on the second half of the Test Datasets

Model	Musk	OptDigits	Pima	SatImg-2	Smtp	Vert	Vowels	WBC	Wine
CHAODA-fast	40.22	75.81	8.16	58.63	156.97	2.59	29.96	4.71	0.89
CHAODA	601.12	8e3	24.50	4e3	413.86	2.32	214.64	7.21	1.08
ABOD	4.39	6.81	0.26	3.20	18.00	**0.08**	0.51	0.14	**0.04**
AutoEncoder	23.40	24.29	4.65	23.73	195.61	2.85	6.98	4.70	3.31
CBLOF	0.27	0.44	**0.08**	0.31	0.65	**0.06**	**0.10**	**0.08**	**0.05**
COF	45.60	119.57	2.40	141.84	2e4	0.28	8.06	0.68	0.11
HBOS	**0.07**	**0.04**	**0.00**	**0.02**	**0.01**	**0.00**	**0.01**	**0.01**	**0.00**
IFOREST	0.93	0.92	0.35	0.79	3.90	0.30	0.40	0.33	0.29
KNN	3.50	5.53	**0.09**	1.84	6.83	**0.03**	0.18	**0.05**	**0.01**
LMDD	375.11	421.22	2.86	302.94	4e3	0.59	9.47	1.91	0.33
LOCI	*TO*	*TO*	*TO*	*TO*	*TO*	35.97	*TO*	154.49	6.84
LODA	**0.07**	**0.10**	**0.04**	**0.10**	0.73	**0.03**	**0.05**	**0.04**	**0.03**
LOF	3.59	5.49	**0.02**	1.50	1.36	**0.00**	**0.06**	**0.01**	**0.00**
MCD	84.74	7.24	0.69	6.83	13.78	**0.05**	0.81	0.11	**0.05**
MOGAAL	267.93	415.02	40.89	463.18	*TO*	39.86	80.35	40.65	38.10
OCSVM	2.51	3.71	**0.03**	3.14	252.52	**0.00**	0.11	**0.01**	**0.00**
SOD	56.94	131.49	2.48	210.39	*TO*	0.54	13.92	1.03	0.19
SOGAAL	29.55	44.72	5.20	48.79	592.27	4.69	10.38	4.76	4.01
SOS	9.37	26.09	1.01	34.96	*TO*	0.21	2.81	0.36	**0.10**
VAE	30.38	30.58	5.67	30.41	176.84	4.03	10.46	5.33	4.62

Table S6: Performance on Train Datasets

Model	Annthy	Mnist	PenDigits	Satellite	Shuttle	Thyroid
CHAODA-fast	0.64	0.67	0.83	0.63	**0.97**	**0.89**
CHAODA	**0.85**	**0.71**	**0.87**	**0.77**	0.86	**0.91**
ABOD	0.50	0.60	0.53	0.51	0.54	0.50
AutoEncoder	0.69	0.67	0.58	0.63	0.94	0.88
CBLOF	0.59	0.62	0.59	0.68	**0.99**	0.87
COF	0.59	0.56	0.53	0.56	0.52	0.49
HBOS	**0.84**	0.53	0.52	0.62	0.74	0.86
IFOREST	0.70	0.61	0.63	0.70	0.91	**0.91**
KNN	0.65	0.65	0.51	0.56	0.53	0.56
LMDD	0.52	0.59	0.56	0.42	0.92	0.70
LOCI	*TO*	*TO*	*TO*	*TO*	*TO*	*TO*
LODA	0.63	0.66	0.57	0.65	0.96	**0.90**
LOF	0.60	0.57	0.52	0.57	0.53	0.49
MCD	0.72	0.57	0.53	0.58	0.96	0.85
MOGAAL	0.46	*TO*	0.67	0.59	*TO*	0.49
OCSVM	0.62	0.63	0.59	0.62	**0.97**	0.78
SOD	0.64	0.55	0.52	0.52	*TO*	0.53
SOGAAL	0.46	0.57	0.57	0.60	0.96	0.49
SOS	0.50	0.52	0.52	0.47	*TO*	0.50
VAE	0.69	0.67	0.58	0.63	0.94	0.88

Table S7: Time taken, in seconds, on Train Datasets

Model	Annthy	Mnist	PenDigits	Satellite	Shuttle	Thyroid
CHAODA-fast	35.92	93.81	58.03	54.60	17.07	35.51
CHAODA	112.86	732.15	481.12	314.60	1e4	108.99
ABOD	3.42	16.39	2.83	3.53	21.58	1.25
AutoEncoder	22.80	40.54	23.87	26.74	158.72	12.84
CBLOF	1.01	1.14	0.23	0.30	0.71	0.21
COF	215.63	277.91	199.03	176.35	1e4	54.82
HBOS	1.58	**0.06**	**0.01**	**0.02**	**0.03**	**0.00**
IFOREST	0.73	1.41	0.80	0.84	3.24	0.54
KNN	0.84	14.30	1.25	2.05	9.13	0.42
LMDD	107.05	1e3	197.48	371.63	6e3	32.69
LOCI	*TO*	*TO*	*TO*	*TO*	*TO*	*TO*
LODA	0.13	0.13	0.12	0.11	0.59	**0.08**
LOF	0.26	14.79	0.90	1.71	6.30	**0.09**
MCD	1.50	20.56	3.01	10.39	10.78	1.03
MOGAAL	594.69	*TO*	561.40	513.84	*TO*	280.80
OCSVM	2.70	11.30	3.04	3.83	162.18	0.68
SOD	262.80	281.86	195.81	253.71	*TO*	91.48
SOGAAL	61.14	68.03	55.40	50.07	460.66	31.43
SOS	101.82	54.22	46.02	42.62	*TO*	42.10
VAE	27.95	50.16	29.63	33.51	182.23	16.18