December 2017

Ensemble Methods for Anomaly Detection

Zhiruo Zhao
Syracuse University

ABSTRACT

Anomaly detection has many applications in numerous areas such as intrusion detection, fraud detection, and medical diagnosis. Most current techniques are specialized for detecting one type of anomaly, and work well on specific domains and when the data satisfies specific assumptions. We address this problem, proposing ensemble anomaly detection techniques that perform well in many applications, with four major contributions: using bootstrapping to better detect anomalies on multiple subsamples, sequential application of diverse detection algorithms, a novel adaptive sampling and learning algorithm in which the anomalies are iteratively examined, and improving the random forest algorithms for detecting anomalies in streaming data.

We design and evaluate multiple ensemble strategies using score normalization, rank aggregation and majority voting, to combine the results from six well-known base algorithms. We propose a bootstrapping algorithm in which anomalies are evaluated from multiple subsets of the data. Results show that our independent ensemble performs better than the base algorithms, and using bootstrapping achieves competitive quality and faster runtime compared with existing works.

We develop new sequential ensemble algorithms in which the second algorithm performs anomaly detection based on the first algorithm's outputs; best results are obtained by combining algorithms that are substantially different. We propose a novel adaptive sampling algorithm which uses the score output of the base algorithm to determine the hard-to-detect examples, and iteratively resamples more points from such examples in a complete unsupervised context. On streaming datasets, we analyze the impact of parameters used in random trees, and propose new algorithms that work well with high-dimensional data, improving performance without increasing the number of trees or their heights. We show that further improvements can be obtained with an Evolutionary Algorithm.

ENSEMBLE METHODS FOR ANOMALY DETECTION

By

Zhiruo Zhao
B.S. Northwestern Polytechnical University, 2011
M.S. Syracuse University, 2013

DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computer and Information Science and Engineering (CISE)

Syracuse University
December 2017

ACKNOWLEDGMENTS

First of all, I would like to express my sincere gratitude to my advisors, Prof. Kishan G. Mehrotra and Prof. Chilukuri K. Mohan, for the continuous support of my Ph.D study and related research. I thank them for encouraging my research and for guiding me to be a researcher throughout my Ph.D journey. I am also thankful for the excellent examples they have provided as successful researchers and scientists.

I would also like to thank for my dissertation defense committee members: Prof. Vir Phoha, Prof. Qinru Qiu, Prof. Sucheta Soundarajan, and Prof. Ramesh Raina, for their time, interests and insightful comments.

My special thanks to Prof. Soundarajan and Prof. Reza Zafarani for giving me valuable feedbacks at our weekly research meeting. And all the professors at Syracuse University who have helped me throughout my graduate study.

I thank for all my lab mates and friends at our department, they are among the most brilliant people I have ever met. I would never forget all the discussions we have, all the fun projects we have worked on, and the support you gave me during my study.

I am grateful to my parents, who have raised me with a love of science and engineering and been supportive for all my pursuits. And my fiance, Qiuwen Chen, who has always been loving and caring.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

Ensemble learning methods have many applications in machine learning and data mining area, especially in the context of classification. In classification problems, ensemble methods have been proven to be effective and robust over the base individual learners both empirically and theoretically. But few such works exist in the context of unsupervised anomaly detection. In this dissertation, we explore the application of ensemble learning methods for anomaly detection on both static and streaming datasets.

In this chapter, we first introduce the general concept of anomaly detection, then review some state-of-the-art detection algorithms which we apply (as components of ensembles) throughout this dissertation. Next, we describe the datasets and metrics we use to evaluate the algorithms described in this dissertation.

1.1 Anomaly Detection

Anomaly detection, also known as outlier detection, is one of the most widely studied among different research and application areas. In fact, the discussion of outlier detection in data sets can be traced back to the 18th century when Bernoulli questioned the practice of deleting the outliers [14]. In recent surveys, the problem of finding anomalies is

often described as *the problem of finding patterns in data that do not conform to expected behavior* [20]; or of *finding data objects with behaviors that are very different from expectation* [33]. Such patterns or data objects are the so-called anomalies or outliers.

Anomalies are usually associated with security threats, financial fraud, medical failure, system failures, etc. One of the most widely applicable areas for anomaly detection is detecting intrusions. For example, the financial losses by the respondent companies due to network attacks were over 130 million dollars in 2005 [52]; and according to a security report, more than 90% respondents experienced cyber-attacks in 2015 [56]. WannaCry ransomware attack, a worldwide cyberattack, launched on May 12 2017, was reported to have infected more than 2 million computers in over 150 countries [13, 21]. Therefore, detecting anomalies is a very critical problem.

Researchers have developed many anomaly detection methods over the years, based on statistics, machine learning, and information theory techniques. Each of the algorithms has an explicit or implicit assumption regarding what types of observations fall in the category of anomalies. Therefore, each algorithm has been developed to target a specific class of problem. However, in reality, the types of anomalies in a dataset can be of various kinds and hence cannot be detected by a single anomaly detection algorithm. To achieve better and more robust solutions, the application of ensemble learning is needed. Ensemble learning in the context of classification has been well explored both empirically and theoretically; Oza and Tumer [50] provide a survey of these techniques. However, as pointed out in [9, 74], the study of ensemble learning in the field of anomaly detection is still an open research problem. In the following, we first describe some of the existing state-of-the-art anomaly detection algorithms, and then introduce the current status of ensemble learning in anomaly detection.

1.2 Review of Existing Detection Algorithms

In this section we first describe recent density and rank based algorithms that we apply in ensemble methods.

The common theme among the three density based algorithms (LOF, COF, and INFLO) is that an anomaly score is assigned to an object based on density comparison of the object with its k nearest neighbors; an object is considered an anomaly if its anomaly score is greater than a pre-defined threshold. The other algorithms are based on the notion of *rank*, and use the concept that if k nearest neighbors of an object consider the object as one of their close neighbors, then it is less likely to be an anomaly.

1.2.1 Density Based Anomaly Detection Algorithms

The density based anomaly detection algorithms assume a symmetric distance function $dist$, with $dist(x, y) = dist(y, x)$, where $x, y \in D$ are two observations from the dataset. An important notion for various algorithms is $N_k(x)$, the set of k nearest neighbors (k-NN) of a point $x \in D$ [18]. To find the k nearest neighbors, first define the k-distance of any observation x as:

$$k\text{-}dist(x) = dist(x, y),$$

where $y \in D$ is the k^{th} nearest neighbor of x such that [18]:

- $|\{z \in D \setminus \{x\} \mid dist(z, x) < dist(y, x)\}| < k$, and
- $|\{z \in D \setminus \{x\} \mid dist(z, x) \leq dist(y, x)\}| > k - 1$.

The set of k nearest neighbors of x is defined as:

$$N_k(x) = \{y \in D \setminus \{x\} \mid dist(x, y) \leq k\text{-}dist(x)\}. \tag{1.1}$$

The *reverse nearest neighborhood* of an object x, $R_k(x)$, is defined to be the set of

points y such that x is among the k nearest neighbors of y, *i.e.*,

$$R_k(x) = \{y \in D \mid x \in N_k(y)\}. \tag{1.2}$$

Note that $N_k(x)$ has at least k objects but $R_k(x)$ may be empty, because x may not be in any of the k-NN sets of any of its k nearest neighbors.

LOF (Local Outlier Factor)

Local outlier factor (LOF), proposed in [18], captures the degree of outlierness of an object based on its local neighbors. It was proposed to solve a problem in global distance-based method [40] as illustrated in Figure 1.1, the distance-based method correctly identifies o_1 as an outlier but fails to identify o_2 because its distance is not large enough. LOF, on the other hand, assigns an outlier score for each object based on its local neighbors, and can successfully detect such outliers. The computed LOF score of an object x is essentially the average of the ratio of the local reachability distance of an object with the average distance to the object's k nearest neighbors $\mathrm{N}_k(x)$. To calculate the LOF score of each point in a given dataset D of a given k value, one should follow the steps given below:

- Calculate the k nearest neighborhood set $N_k(x)$ of each object $x \in D$.
- Calculate the local reachability density of each object $x \in D$:

$$lrd_k(x) = \left[\frac{\sum_{y \in N_k(x)} reach\text{-}dist_k(x, y)}{|N_k(x)|} \right]^{-1}$$

  where $reach\text{-}dist_k(x, y) = \max\{k\text{-}dist(y), dist(x, y)\}$ is the *reachability distance* of x from y.

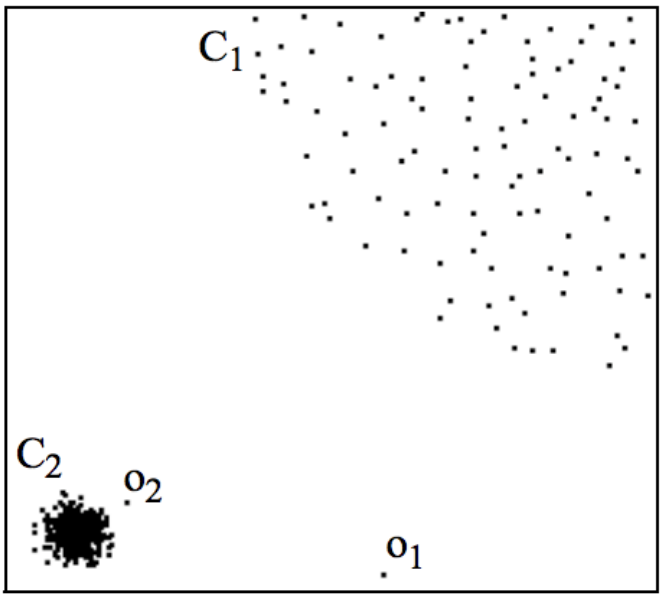


Fig. 1.1: Distance-based based outlier detection algorithm fails to capture o_2 because its distance is not large enoughx – from [18]

- Calculate the local outlier factor (LOF) score of each object $x \in D$:

$$LOF_k(x) = \frac{\sum_{y \in N_k(x)} \frac{lrd_k(y)}{lrd_k(x)}}{|N_k(x)|}$$

  $LOF_k(x)$ captures the degree of x's outlierness from its k-NN neighborhood. If x's local reachability density is very low while its k nearest neighbors $y \in N_k(x)$ densities are high, then, x gets assigned a high LOF score, indicating that x is a potential outlier.

As a result, LOF score > 1 indicates that the object is potentially an outlier, whereas if LOF is ≤ 1 then the object's local density is as large as the average density of its neighbors, *i.e.*, it is a non-outlier. LOF is considered to be the best method among NN approach, unsupervised SVM approach, and Mahalanobis based approach in the area of network intrusion detection [43]. Though it was proposed over ten years ago, it is still widely used in many recent applications. LOF has been used for fault process detection in a plant-wide process

monitoring system [45]; LOF was used to reduce the false alarm rate in their false data in the application of solar photovoltaic (PV) systems [71]; more recently, LOF was adopted in WiFall [69], an automatic fall detection system, to detect the anomalous wireless signal and to contribute to elders' health care and facilitate injury rescue.

COF (Connectivity-based Outlier Factor)

After LOF, many other local kNN-based algorithms were proposed. Connectivity-based Outlier Factor (COF), proposed in [65], attempts to solve one of the deficiencies that LOF fails to detect outliers when the dataset contains clusters of different shapes, as shown in Figure 1.2.

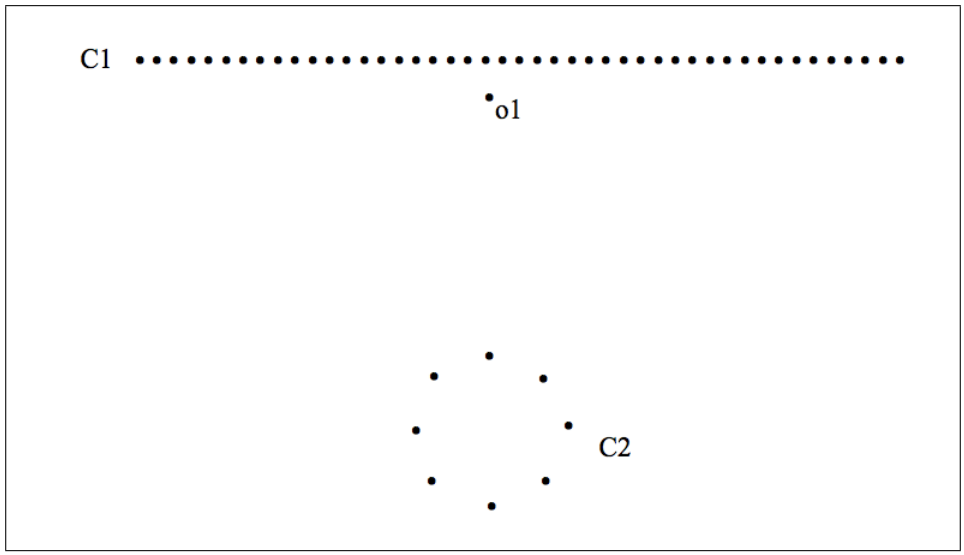


Fig. 1.2: LOF fails to detect outliers – from [65]

To solve this problem, [65] proposed an alternative method for local density calculation which considers the "connectivity" – how an object connects to its neighbors, and use relative isolation to indicate whether an object is deviating from others. This connectivity is defined by *set-based nearest path (SBN)* and *set-based trails (SBT)*. To calculate a COF score for each object $x \in D$ for a given k, one can follow the steps given below:

- Define the shortest distance between elements of two non-empty, disjoint sets A and

B, as:

$$set\text{-}dist(A, B) = \min\{dist(x, y) : x \in A, y \in B\}^1$$

- Calculate the k nearest *SBN-path* starting from x_1: $< x_1, x_2, ..., x_k >$ such that x_{i+1} is the nearest neighbor of set $\{x_1, ..., x_i\}$ in set $\{x_{i+1}, ..., x_k\}$, for $1 \leq i \leq k-1$. In general, the *SBN-path* is generated in an iterative expansion manner starting from $SBN(x) = \{x\}$; in each iteration, the nearest neighbor object is added to $SBN(x)$. In other words, the *SBN-path* of object x is an ordered list of x's nearest neighbors.

- The *set-based nearest trail (SBN-trail)* w.r.t a *SBN-path* $p =< x_1, x_2, ..., x_k >$ is defined as a sequence $< e_1, e_2, ..., e_{k-1} >$ such that $e_i = (y, x_{i+1})$ where $y \in \{x_1, ..., x_i\}$, $|e_i| = set\text{-}dist(\{x_1, ..., x_i\}, \{x_{i+1}, ..., x_k\})$. $|e_i|$ is called the *cost description* of edge e_i. An illustration of a *SBN-trail* is shown in Figure 1.3.

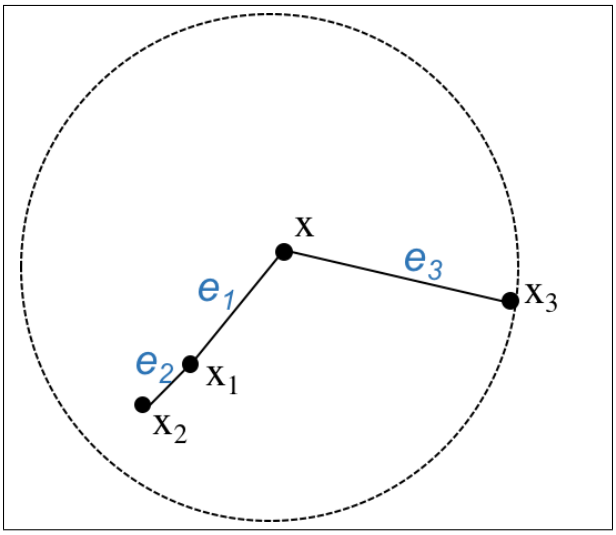


Fig. 1.3: For $k = 3$, the *SBN-path* of x is $\{x, x_1, x_2, x_3\}$, the *SBN-trail* is $\{e_1, e_2, e_3\}$

- Calculate the average chaining distance from x to $N_k(x)$:

$$AC\text{-}dist_{N_k(x)} = \frac{1}{k-1} \cdot \sum_{i=1}^{k-1} \frac{2(k-i)}{k} \cdot |e_i|$$

[1]Note that this is not formally a metric since it does not satisfy the triangle inequality.

the $AC\text{-}dist$ is the average of the weighted distances in the *cost description* of the *SBN-trail*. In the *SBN-trail*, if the edges closer to x have larger cost descriptions than the edges far away from x, then, the closer edges contribute more in the calculation of $AC\text{-}dist$.

- Calculate the COF score for each $x \in D$:

$$COF_k(x) = \frac{|N_k(x)| \cdot AC\text{-}dist_{N_k(x)}(x)}{\sum_{y \in N_k(x)} AC\text{-}dist_{N_k(y)}(y)}.$$

The COF score of an object x indicates how x deviates from the objects it connects to, compared with its k nearest neighbors. An object with higher score is considered to be a potential outlier.

INFLO (INFLuential measure of Outlierness by symmetric relationship)

Another variation of LOF was proposed by [38], which addressed the problem that LOF fails to detect outliers when more than one cluster is present in the dataset, and different clusters have different densities. As shown in Figure 1.4, when q is slightly closer to a dense cluster than p, then p will be assigned a higher LOF score. INFLO proposes to use not only the k-nearest-neighborhood but also the reverse-nearest-neighborhood (*RNN*) ($R_k(x)$), defined in Equation (1.2). INFLO score is calculated as follows:

- Calculate $N_k(x)$ and $R_k(x)$ for each $x \in D$.

- Calculate the local density of each object:

$$den(x) = [k\text{-}dist(x)]^{-1}$$

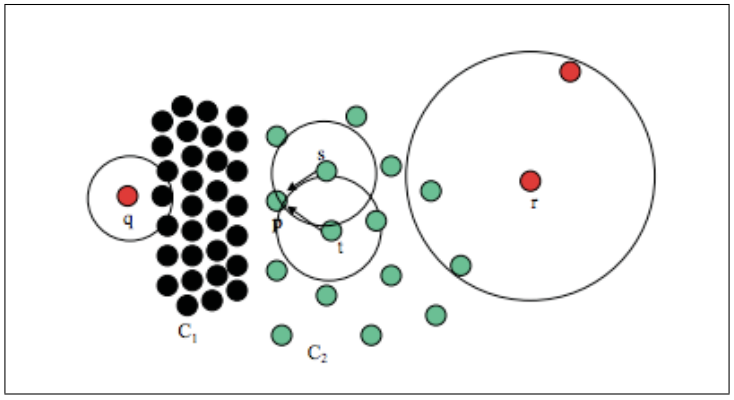


Fig. 1.4: LOF assigns low score to q and r when clusters of different densities are present – from [38]

- Get the k-influential-space for each object $x \in D$:

$$IS_k(x) = N_k(x) \cup R_k(x)$$

- Calculate the INFLO score as:

$$INFLO_k(x) = \frac{\sum_{y \in IS_k(x)} den(y)}{|IS_k(x)|} \cdot [den(x)]^{-1}$$

Therefore, for each object, INFLO compares its local density to that of both its kNN and RNN.

1.2.2 Rank Based Anomaly Detection Algorithms

RBDA (Rank Based Detection Algorithm)

Rank Based Detection Algorithm (RBDA) [37] exploits the mutual closeness of an object to its neighbors. In Figure 1.5, for example, Jack and Eric rank each other high in their list of friends while Bob is not considered as friend by his friends. Therefore, Bob is not as popular in this social network. These personal friends are essentially the local nearest neighbors in the dataset, and the friendship rank measures how far away an individual

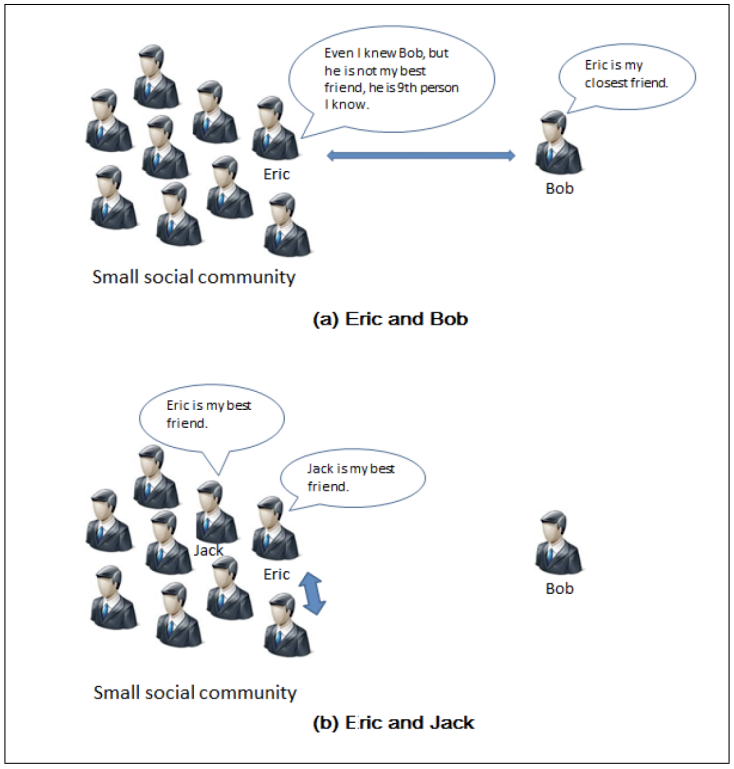


Fig. 1.5: Using ranks of friendship in social network to test the popularity – from [37]

deviates from its friends. This mutual closeness is measured by friendship in a social network, while it is measured by the distance function in a generic dataset. The following steps describe RBDA:

- Calculate $N_k(x)$ for each object $x \in D$.
- For each pair of objects $(x, y) \in D$, the rank of y respect to x is defined as:

$$r_x(y) = |\{z : d(x, z) < d(x, y)\}|.$$

  Rank measures how 'close' y is to x; if there are fewer observations between x and y, then, y is 'close' to x, *i.e.* $r_x(y)$ is very small.

- Calculate the RBDA score for each object x as the averaged rank with respect to its

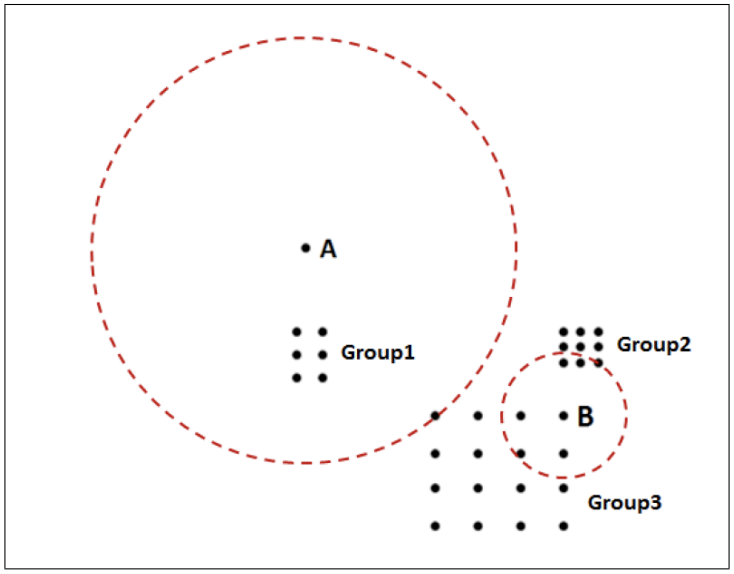


Fig. 1.6: Density-based algorithm assigns higher score to B than A – from [37]

kNN:

$$RBDA_k(x) = \frac{\sum_{y \in N_k(x)} r_y(x)}{|N_k(x)|}.$$

RBDA has an advantage in detecting outliers when clusters with different densities exist in the dataset, unlike the density based algorithm that misidentifies the point at a cluster boundary as an outlier as shown in Figure 1.6.

RADA (Rank with Averaged Distance Algorithm)

RADA [36] adjusts the rank of each object x by the averaged distance from its k nearest neighbors. The function for measuring outlierness of an object x is defined as:

$$RADA_k(x) = RBDA_k(x) \times \frac{\sum_{y \in N_k(x)} d(x, y)}{|N_k(x)|}.$$

1.2.3 Statistical Methods

Statistical anomaly detection methods detect an anomaly *which is suspected of being partially or wholly irrelevant because it is not generated by the stochastic model assumed* [11]. In general, a statistical anomaly detection model fits a statistical model for the normal observations, and marks any observation that does not fit in with the given model as an outlier. Detailed discussion of statistical anomaly detection techniques can be found in [20,52]. We introduce one of the popular techniques in detail in this section, namely, the Gaussian Mixture Model.

Gaussian Mixture Model (GMM)

GMM is one of the most frequently used parametric models for detecting anomalies when there is no prior knowledge of the density distribution [42]. GMM is a probabilistic model which assumes that all observations are generated from N gaussian distributions but with unknown parameters, known as the mixture components. Each multivariate component is represented by a parameter set

$$\theta_i = \{\mu_i, \mathbf{\Sigma}_i\}$$

where μ_i and $\mathbf{\Sigma}_i$ are the mean vector and the covariance vector for the i^{th} mixture model, respectively. The set of parameters $\{\theta_1, ..., \theta_N\}$ is estimated using Expectation-Maximization (EM) algorithm [23]. EM algorithm might converge to a local maximum; to overcome this problem, multiple trials of GMM are executed in our implementation.

1.2.4 Streaming Anomaly Detection Algorithms

A datastream is defined as an infinite sequence of data points [59]:

$$DS = \{(x_1, t_1), ..., (x_n, t_n), ...\},$$

where data x_i arrives at timestamp t_i. Recent research papers in [32, 59] discuss the challenges and open problems in detecting anomalies for streaming data. Anomaly detection techniques developed for datastreams should have the following features:

- **online**: an observation should be identified as an anomaly at the time it arrives;
- **temporal context**: an observation should be compared with its temporal context (e.g. within a time period);
- **incremental learning**: a detection model should be adjustable incrementally;
- **concept drift**: a model should be able to detect anomalies under the presence of distribution change; and
- **multi-dimensional**: a model should be able to apply fast detection on multi-dimensional datastreams.

Over the years, many online anomaly detection techniques have been proposed, including statistics-based methods [70] which incrementally learn a probabilistic model and determine whether an observation is an anomaly compared to the learnt model. The incremental LOF [55] was the online version of LOF, which computes the neighborhood in an incremental way, and reduces the time complexity for detection from $O(n^2)$ to $O(n \log n)$. Though many detection methods have been proposed, there are few ensemble methods on streaming data. The recent work proposed in [64] applies the concept of random forest in the context of streaming anomaly detection, and reports good performance in both accuracy and time complexity. We will discuss these streaming methods in detail in Chapter 5.

1.3 Ensemble Methods in Machine Learning

Ensemble methods are widely used in the field of data mining and machine learning, especially in the context of classification and clustering. Ensemble methods are considered to

be among the best data mining techniques in many of the data mining competitions. Vitaly Kuznetsov said in NIPS 2014 that *"This is how you win ML competitions: you take other peoples' work and ensemble them together."* [4]. As a matter of fact, using ensemble methods to solve data mining problems is becoming one of the most popular techniques. For example, in 2009 Netflix launched a 1 million dollar competition for user rating prediction and the team "The Ensemble" achieved the highest accuracy [5, 72]; in 2009-2011 KDD cups, all the first place and second place winners used ensemble methods [72]; the 2016 KDD cup winning team used the gradient boosted decision trees which is an ensemble method which combines the results of a collection of weak predictors, typically decision trees [60]. However, as Charu C. Aggarwal mentioned in [9], " *Compared to the clustering and classification problems, ensemble analysis has been studied in a limited way in the outlier detection literature.*"

A particular algorithm may be well-suited to the properties of one data set and be successful in detecting anomalous observations of the particular application domain, but may fail to work with other datasets whose characteristics do not agree with the first dataset. The impact of such mismatch between an algorithm and an application can be alleviated by using ensemble methods where multiple algorithms are pooled before a final decision is made. Mathematically, ensemble methods help by addressing the classical bias-variance dilemma, at least in the classification context.

Recently, a few researchers have studied ensemble methods such as anomaly detection and distributed intrusion detection in Mobile Ad-Hoc Networks (see Cabrera *et al.* [19]). Others have studied supervised anomaly detection using random forests and distance-based outlier partitioning (see Shoemaker and Hall [62]). In the semi-supervised case, one possible approach is to convert the problem to a supervised anomaly detection by exploiting strong relationships between features; Tenenboim-Chekina *et al.* [66] and Noto *et al.* [49] provide two different approaches to accomplish this goal. Pevny *et al.* [54] propose anomaly detector processing for a continuous stream of data, which requires high

acquisition rate, limited resources for storing the data, and low training and classification complexity. The key idea is to use bagging on multiple weak detectors, each implemented as a one-dimensional histogram. In these approaches the word 'ensemble' has a different connotation.

In this dissertation, we propose and evaluate different ensemble methods for anomaly detection on static and streaming datasets. We evaluate our methods with respect to many real-world benchmark datasets. We now describe the evaluation metric and datasets we use throughout the entire dissertation, then, discuss our contribution and summarize the content for each chapter.

1.4 Evaluation Metrics

In this dissertation, we mainly use Area Under Curve (AUC) as the evaluation metric. The details are described below:

A *Receiver Operating Characteristic (ROC)* was originally used for radar signal analysis during World War II [31], to measure the power of rader receiver operators [24]. It has a wide range of applications in signal detection [26], biomedical informatics [41], clinical medicine performance [76], etc. Detailed descriptions can be found in [27, 34]. More recently, ROC curve has become one of the most popular performance metrics in the area of machine learning [51], it plots the true positive rate vs. the false positive rate. The ROC curve provides an illustration of detection power in a 2-dimensional space, however, we often need a single scalar value to simplify the evaluation process. Therefore, the area under the ROC curve (AUC) is used to reduce the ROC curve to a single-valued number [34]. A perfect detection algorithm reaches an AUC score of 1.0 while a random guess for two class problem is 0.5. In machine learning, the positive class is often considered to be the class of interest, therefore, in our work, positive class is the anomaly class (outliers), denoted as $\mathcal{O}$, and the negative class is the normal class (inliers), denoted as $\mathcal{I}$. While the set of predicted

outliers is denoted as $\hat{\mathcal{O}}$, predicted inlier set is denoted as $\hat{\mathcal{I}}$. Therefore, the True Positive Rate (TPR) and False Positive Rate (FPR) are calculated using:

$$TPR = \frac{|\hat{\mathcal{O}} \cap \mathcal{O}|}{|\mathcal{O}|} \quad \text{and} \quad FPR = \frac{|\hat{\mathcal{O}} \cap \mathcal{I}|}{|\mathcal{I}|}.$$

1.5 Dataset Descriptions

In this section, we describe first the benchmark static datasets we use for ensemble algorithms evaluation in Chapters 2, 3, and 4. Then, we describe the streaming datasets we use for Chapter 5.

1.5.1 Benchmark Static Datasets

- **Abalone dataset (Abalone)** [46] : This dataset was used originally for predicting the age of abalones from physical measurements. We keep the 7 numerical features and choose the minority classes as outliers, and downsampled the two major classes (class 9 and class 10) as the inliers.

- **Smartphone-Based Recognition of Human Activities and Postural Transitions Data Set (ACT)** [57] : This dataset collects data from 30 volunteers wearing smartphones. There are three static postures (standing, sitting, lying) and three dynamic activities (walking, walking downstairs and walking upstairs), as well as postural transitions that occurred between the static postures (stand-to-sit, sit-to-stand, sit-to-lie, lie-to-sit, stand-to-lie, and lie-to-stand). The 3-axial linear acceleration and 3-axial angular velocity values from the sensors data are used for classification, with 561 features and 12 classes. To construct our outlier detection evaluation dataset, we consider the class with the least number of instances (sit-to-stand) as outlier points and the class with most number of instances (standing) as inlier points.

- **EEG dataset (EEG)** [46] : All data is from one continuous EEG measurement with the Emotiv EEG Neuroheadset. The duration of the measurement was 117 seconds. The eye state was detected via a camera during the EEG measurement and added later manually to the file after analyzing the video frames. It has two states: eye-close and eye-open. We downsample the eye-close class as the outlier class.

- **Glass dataset (Glass)** [39] : The original glass identification dataset from UCI machine learning repository is a classification dataset. The study of classification of types of glass was motivated by criminological investigation. At the scene of the crime, the glass left can be used as evidence, if correctly identified. This dataset contains features regarding several glass types (multi-class). Here, class 6 is a clear minority class, as such points of class 6 are marked as outliers, while all other points are inliers.

- **Ionosphere dataset (Iono)** [46] : The Johns Hopkins University Ionosphere dataset contains 351 data objects with 34 features; all features are normalized in the range of 0 and 1. There are two classes labeled as *good* and *bad* with 225 and 126 data objects respectively. There are no duplicate data objects in the dataset. To form the rare class, 116 data objects from the *bad* class are randomly removed. The final dataset has only 235 data objects with 225 *good* and 10 *bad* data objects.

- **KDD 99 dataset (KDD)** [35] : KDD 99 dataset is available from DARPA's intrusion dataset evaluation program. This dataset has been widely used in both intrusion detection and anomaly detection area, and data belong to four main attack categories. In our experiments, we select 1000 normal connections from test dataset and insert 14 attack-connections as anomalies. All 41 features are used in experiments. The dataset is available at [3].

- **Lymphography dataset (Lympho)** [44] : The original lymphography dataset from the UCI machine learning repository is a classification dataset. It is a multi-class

dataset having four classes, but two of them are quite small (2 and 4 data records). Therefore, those two small classes are merged and considered as outliers compared to the other two large classes (81 and 61 data records).

- **NBA Basketball dataset (NBA)** [1] : This dataset contains Basketball player statistics from 1951-2009 with 17 features. It contains all players statistics in regular season and information about all star players for each year. We construct our outlier detection evaluation dataset by selecting all star players for year 2009, use their regular season stats as the outlier data points, and select all the other players stats in year 2009 as normal points.

- **Packed Executables dataset (PEC)** [53] : Executable packing is the most common technique used by computer virus writers to obfuscate malicious code and evade detection by anti-virus software. This dataset was collected from the Malfease Project to classify the non-packed executables from packed executables so that only packed executables could be sent to universal unpacker. In our experiments, we select 1000 packed executables as normal observations, and insert 8 non-packed executables as anomalies. All 8 features are used in experiments. The dataset is available at [2].

- **Popularity (Pop) dataset** [28] : This dataset contains features of articles published by Mashable in a period of two years. The original goal in the paper was to predict whether a news is popular or not in the social network. In the original paper [28], they threshold the number of shares to be 1400 to indicate whether a post is popular. For our experiments, we randomly down sampled 10 data points from the Popular class and mark them as outliers. Inliers are from the class non-popular.

- **Statlog (Landsat Satellite) dataset (Sat)** [46] : This dataset consists of the multi-spectral values of pixels in 3x3 neighborhoods in a satellite image. The original usage was to predict the class associated with the central pixel in each neighborhood.

To construct the outlier detection dataset, we downsampled class 4 (damp grey soil class) as the outlier, and class 7 (very damp grey soil class) as the inliers.

- **Wine dataset** [61] : This dataset has 13 features and 3 classes. To construct an outlier detection dataset, class 2 and 3 are used as inliers and class 1 is downsampled to 10 instances to be used as outliers. These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines.

- **Wisconsin Breast Cancer Dataset (WIS)** [46] : This dataset contains 699 instances and has 9 features. There are many duplicate instances and instances with missing feature values. 236 instances are labeled as benign class and 236 instances as malignant. In our experiments the final dataset consisted 213 benign instances and 10 malignant instances as anomalies.

1.5.2 Streaming Datasets

In this section, we describe the datasets we used for evaluating our streaming ensemble methods in chapter 5.

1. **Http Dataset** [64] : The KDD99 dataset can be divided into different subsets by the most basic feature 'service'. Http is one of the subsets, in which the continuous features are transformed by taking a log function. This dataset is constructed using 567,497 'http' service data and contains 0.4% anomalies.

2. **CoverType Dataset** [46, 64] The original ForestCover/Covertype dataset is used for predicting the forest cover types from the cartographic variables. The original dataset has 54 features including the quantitative variables, binary wilderness areas and bi-

Table 1.1: Summary of the benchmark datasets used for static data analysis

Datasets	No. of features	Size of	No. of outliers	% of outliers	Short description
ABALONE	7	492	5	1.02	the minority classes vs. class 9 & 10
ACT	561	1446	23	1.59	class sit-to-stand vs. class standing
EEG	13	779	7	0.90	class 1 vs. class 0
GLASS	9	214	9	4.21	class 6 vs. the rest
IONO	34	235	10	4.26	bad vs. good
KDD	38	1012	12	1.19	intrusions vs. normal traffic
LYMPHO	18	148	6	4.05	two small classes vs. two large classes
NBA	17	578	24	4.15	all star players in year 2009 vs. the rest
PEC	9	1008	8	0.79	malicious software vs. normal software
popularity	10	1010	59	0.99	popular vs. non-popular news
SAT	4	474	36	0.08	class 4 vs. class 7
WINE	10	129	13	7.75	class 1 vs. class 2 and 3
WIS	9	223	10	2.23	malignant vs. benign

nary soil type variables. However, the outlier detection dataset is constructed by the 10 quantitative features. There are 286,048 data and 0.9% anomalies are inserted.

3. **Polish Companies Bankruptcy Dataset** [73] This dataset is collected from the Emerging Markets Information Service (EMIS), which collects the information about emerging markets. The dataset contains financial information about the Polish companies from 2000-2013. We use the 5th year dataset which contains companies that are bankrupted after one year. This dataset has 64 financial features about 5910 total companies, of which, 410 companies were bankrupted during the predicting period.

1.6 Our Contribution

Anomaly detection ensembles has been categorized into two classes: independent learning and sequential learning, depending on how the base learners are used [9]. In this dissertation, we explore and propose new algorithms for independent, sequential and adaptive learning. We study the application for ensemble methods in two contexts: static datasets and streaming datasets. For static datasets, we select the state-of-the-art unsupervised anomaly detection algorithms as our base algorithms and design different ensemble methods. For streaming anomaly detection, we study both unsupervised and semi-supervised anomaly detection techniques.

- In Chapter 2, we first discuss the usage of independent ensemble methods for unsupervised anomaly detection. We consider three ensemble approaches using: (1) normalized scores, (2) rank aggregations, and (3) majority voting. We select 6 different state-of-the-art unsupervised anomaly detection algorithms for ensembles, and our evaluation on different benchmark datasets show that using independent ensembles results in better performance than most of the base algorithms. In particular, using minimum ranking method achieves the best and most robust solutions. We then propose to use the bootstrapping idea for anomaly detection, *i.e.*, taking multi-

ple samples of the dataset, obtain anomalies for each sample, and combine the results. This algorithm is comparable with existing works but has a lower computation cost. Finally, we discuss how to combine model-level ensembles and data-level ensembles, by boosting together to reach more robust decisions.

- In Chapter 3, we describe new sequential ensemble algorithms where the outputs of the first algorithm are further refined by another algorithm to detect anomalies. It has been argued that an anomaly detection algorithm performs better on a subsample of the dataset. We incorporate the sampling concept in proposed sequential methods. To select two (or more) algorithms, we propose to use the algorithms which have higher diversity among themselves as the base algorithms. In this chapter, we consider several minor variations of anomaly detection based on the sieve method which takes the output from one base algorithm to filter out the non-anomalous observations, then compare the suspect anomalies with a second algorithm.

- Boosting is considered as an example of iterative sequential learning; in particular, AdaBoost [29] has gained popularity in the classification context. However, boosting has not been used for unsupervised anomaly detection. In Chapter 4, we propose a novel adaptive learning algorithm for unsupervised outlier detection which uses the score output of the base algorithm to determine the hard-to-detect examples, and iteratively resamples more points from such examples in a completely unsupervised context. Finally, we propose several methods to combine the results from each iteration.

- The random forests algorithm has shown better performance than single decision trees in the classification context [17]. Recently, a random forest approach has been proposed for anomaly detection [64]. However, this approach suffers from several deficiencies: (1) parameters are chosen based on empirical learning, and (2) this method does not perform well on high-dimensional data. In Chapter 5, we first an-

alyze the impact of parameters used in random trees, from both empirical and theoretical points of view. Our main contribution is building better random forests for anomaly detection, achieved by : (1) determining the appropriate number of trees of heights based on our mathematical analysis, (2) feature clustering to build random forests without increasing the number of trees or tree heights, in order for the approach to be applicable to datasets with large number of features, and (3) applying an Evolutionary Algorithm to further improve performance of the randomly built trees.

- In Chapter 6, we review and summarize the accomplishments of this dissertation. Then, we discuss the possible future works and improvements over the existing study.

CHAPTER 2
INDEPENDENT ENSEMBLE METHODS FOR ANOMALY DETECTION

Independent ensemble methods combine the decisions from multiple base learners to reach a more robust and accurate final decision. In this chapter, we discuss the application of independent ensemble methods in the context of anomaly detection from both theoretical and empirical perspectives. We review and summarize some of the recent works using ensemble methods for anomaly detection. Then, we propose a bootstrapping algorithm based on the conclusion that using subsamples from the dataset will reach higher detection rate [75]. Results show that our bootstrapping algorithm provides competitive detection accuracy with much less computational cost. To combine the results from different base components, multiple combining strategies can be applied, including score based, ranking based and other approaches. While most existing works only apply a simple score averaging strategy, we explore the usage of rank aggregation and propose to apply a majority voting rule for the final result combination.

2.1 Independent Ensembles

In ensembles, the final decision is jointly decided by multiple base components. Independent ensembles harness the independence between the base learners to achieve a lower error rate. The base components of independent ensembles can be constructed using *instantiations of base algorithms*, using *subsets of datasets*, and using *different projections of dataset on feature space*. After obtaining all the outputs from each component, the next step is *final output combination* where the outputs are combined to achieve a better solution.

An illustration of the independent ensemble approach is shown in Fig 2.1. There are T base components, each of which could be a different algorithm, or the same algorithm but with different parameter settings or initiation [10]. The entire dataset D is taken, while a separate data transform function g_i is applied for the i^{th} base component. For example, if g is a random subsampling approach as in [75], then $g_i(D)$ is a random sample from the dataset D and serves as the input for the i^{th} base component. Each base component outputs a result vector $\mathbf{Res_i}$ indicates the results made at component i. A combination function f is applied to $\mathbf{Res_i}, i = 1...T$ to make the final decision $\mathbf{Res}_{final}$; the combination function could be averaging, minimum, etc.

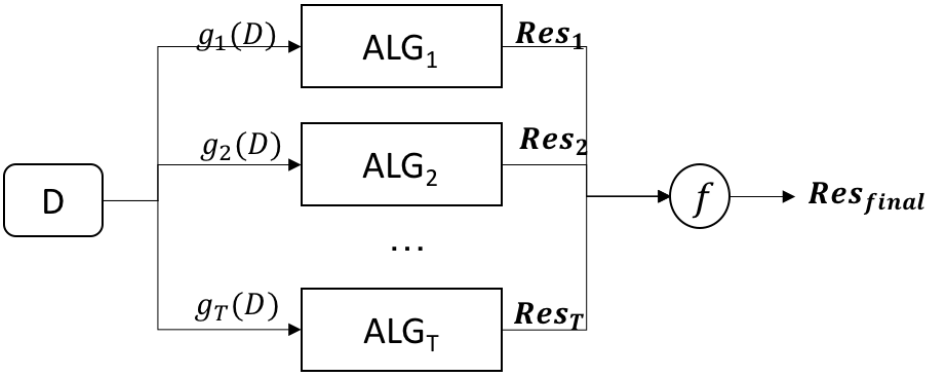


Fig. 2.1: Independent Ensemble for Anomaly Detection

2.1.1 Justification for Independent Ensemble Methods

The concept of using ensemble methods to achieve a better decision can be traced back to the famous Condorcet's Jury Theorem [58] which was first proposed by the Marquis de Condorcet in his 1785 essay on the *Application of Analysis to the Probability of Majority Decisions* [22]. The theorem says that when a jury of T voters need to reach a decision by majority voting, if the probability of each voter being correct is p and the final decision being correct is $p^\star$, then it follows that:

- If $p > 0.5$, then $p^\star > p$.
- If the number of votes approaches infinity, then $p^\star$ approaches to 1.0 if $p > 0.5$

The theory has two assumptions, that each vote should be independent and there should exist only two outcomes, for example, to convict or not. This can be easily adopted to independent ensembles for anomaly detection since:

- all votes (detection algorithms) are independent with respect to each other; and
- the decision outcome for anomaly detection falls into binary classes: whether an observation is an anomaly or is not.

As a result, from Condorcet's Jury Theorem, it follows that applying independent ensemble methods for anomaly detection could lead to a more robust and stronger decision than the single detection result. However, Condorcet did not give a mathematical proof. A theoretical justification is provided by *Hoeffding's Inequality* that as the number of base learners becomes very large, the generalization error goes to zero [72].

Consider a binary classification problem where the label of object x from the i^{th} classifier is $H_i(x) \in \{+1, -1\}$. Suppose we have T classifiers which are independent with each other. Define p as the probability that the decision made by one of the members in the jury is correct. As a result, each classifier has a classification error $1 - p$. After combining these

T classifiers, the final decision for each object is represented as:

$$H(x) = \text{sign}\Big(\sum_{i=1}^{T} h_i(x)\Big)$$

The final decision makes an error if more than half of the T classifiers make errors. By *Hoeffding's Inequality*, the generalization error of an ensemble is [72]:

$$\text{Error} = \sum_{i=0}^{\lfloor T/2 \rfloor} \binom{T}{i} (p)^i (1-p)^{T-i} \leq \exp\Big(-\frac{1}{2} T\big(2(1-p)-1\big)^2 \Big)$$

This guarantees that as the number of base learners increases, the generalization error is exponentially decreasing.

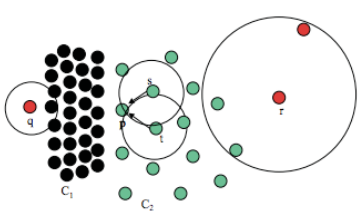
2.1.2 Benefits of Ensembles - A Toy Example

In unsupervised anomaly detection, each algorithm makes an assumption about what is an anomaly. For example, in density-based detection algorithms, the observations in a low-density area are identified as anomalies. Fig 2.2 shows examples of the illustrations of data on which different algorithms are successful.

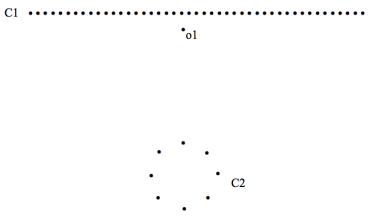
To show how to 'cancel' the bias of each algorithm in detecting anomalies, we construct a very simple toy example with three outliers, shown in Figure 2.3(a). Figure 2.3(b) shows the ROC curve for different algorithms. Each algorithm successfully captures 2 out of 3 outliers but fails to detect the other one. However, all outliers are captured by applying an ensemble using the minimum method which is defined in Equation (2.4) in Section 2.3.

2.2 Data Transformation

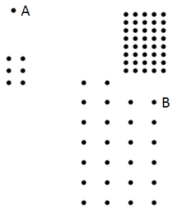
One of the requirements of the independent ensemble approaches is that the decisions made from each base component should be mutually independent. However, when we apply anomaly detection on the same dataset, the base components are inherently dependent due



(a) INFLO correctly detects q



(b) COF correctly detects $o1$



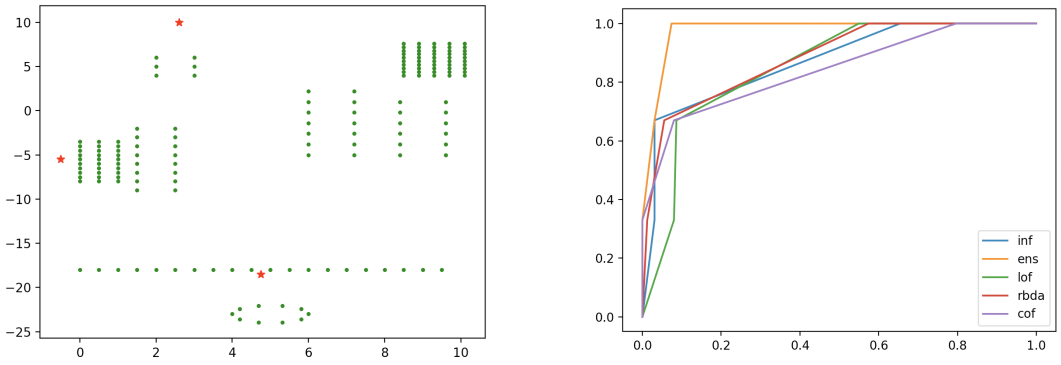
(c) RBDA correctly detects A

Fig. 2.2: Detection power of different algorithms [37, 38, 65]

to the reason that they share the same dataset. To reduce this dependence, different data transformation techniques can be applied. In this section, we review two of the existing data transformation functions including feature bagging [44] and random subsampling [75], then, we propose our new bootstrapping algorithm.

2.2.1 Random Feature Bagging

Feature bagging [44] is the first work to formally describe the application of an ensemble approach for unsupervised anomaly detection, motivated by two observations relevant to outlier detection: 1) outliers might only be detectable from a subspace or projection of feature space; 2) in high dimensional space, distances become sparse and outliers are difficult to distinguish from the normal observations. This is shown in Figure 2.4, where the two outliers A and B can only be detected on different projections of feature spaces. This effect may occur when different types of outliers exist in the dataset. Another problem is the famous curse of dimensionality: in Euclidean space, as the dimensionality increases, the distances between data points are less distinguishable from each other [15]. When the dimensionality d of the feature space becomes very large, the ratio of the difference in min-



(a) Toy example with 3 outliers

(b) Evaluation: ROC Curves of independent methods (INFLO, LOF, RBDA and COF) vs. ensemble method (ens)

Fig. 2.3: Toy example with three outliers

imum and maximum Euclidean distance from data points to the centroid, and the minimum distance itself, tends to zero, *i.e.*:

$$\lim_{d\to\infty} E\left(\frac{dist_{max} - dist_{min}}{dist_{min}}\right) \to 0.$$

A recent article [63] shows that classification performance reduces significantly when increasing the dimensionality but not the number of training samples. To solve the afore-

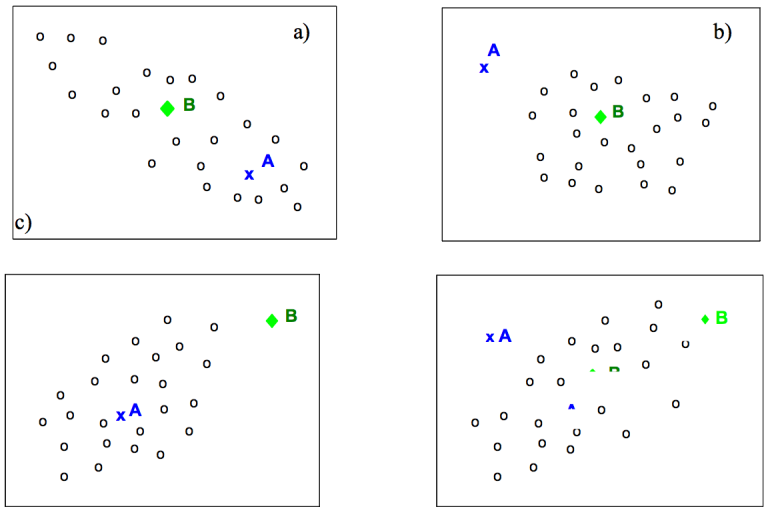


Fig. 2.4: Outliers may only be detectable on different projections of feature space [44]

mentioned problems, this work applies subsampling on the feature space as shown in Al-

gorithm 1. By contrast to the standard bagging approach, instead of randomly sampling from the data distribution, this approach draws random samples from the feature space and keeps all the data points. In the t^{th} iteration, all data samples are preserved, but N_t features are drawn uniformly from the entire feature space FS.

Algorithm 1 Feature Bagging Approach

Data: Dataset D with $d - dimensional$ feature space FS

Input: A set of anomaly detectors $A = A_1, ..., A_T$

Result: A vector of anomaly scores H

Normalize dataset D

for $t = 1, ..., T$ **do**

N_t = $\mathcal{U}\{\lceil d/2 \rceil, d-1\}$;

FS_t = randomly pick N_t features from FS without replacement;

H_t = apply detection algorithm A_t on FS_t, get a score vector for each object;

end

Final output: H = COMBINE($H_1, ..., H_T$)

The detection algorithm used in the original paper [44] is the same over T iterations, which is LOF. For the final *COMBINE* function, it discussed two combination methods: cumulative sum and breadth first approach, which we discuss in detail in Section 2.3. The paper showed that the approach outperforms LOF in many datasets especially when there exist noisy features; also cumulative sum outperforms breadth first combination.

2.2.2 Random Data Bagging

Bagging (**B**ootstrap **agg**regat**ing**) is one of the most used independent ensemble methods, and has been widely used in the classification context. Bagging was first proposed by Leo Breiman [16] where he proposed to improve the classification accuracy by building classifiers on random subsamples of training set. In Section 2.2.1, we have discussed Bagging

on the feature space. In this section, we discuss the Bagging approach by first summarizing Zimek's [75] paper and then propose a new Bootstrapping algorithm.

Benefits from Detecting Outliers by Random Subsampling

The work in [75] proposed to use random subsampling methods for unsupervised anomaly detection to reduce the false detection rate. This method draws a random sample s from the dataset D, and for all observations $x \in D$, an anomaly detection algorithm is performed on $\{x\} \cup D$. This step is repeated multiple times and the final score is an average over all iterations. Zimek *et al.* [75] argues that by doing so, the benefits come from two aspects: 1. the density estimate for each sample will be more robust; 2 subsampling will increase the 'gap' between then. We now discuss the first benefit. Consider a scenario in which the true (but unknown) density distribution of inliers from dataset D are generated from f, so that the estimated density of object x is

$$\hat{f_D}(x) = f(x) + \epsilon_D(x),$$

where ϵ_D is the sample error of estimation. When multiple samples are taken, the expected density of x is

$$E\{\hat{f_D}(x)\} = E\{f(x)\} + E\{\epsilon_D(x)\} = f(x) + E\{\epsilon_D(x)\}$$

Then, if the errors $\epsilon_D(x)$ are independent between multiple subsamples, the estimation of x will maintain the same order of true density $f(x)$ since in this case $E\{\epsilon_D(x)\}$ is zero. However, since there is no guarantee that the errors are the same over multiple samples, Zimek's [75] paper argues that using their random subsampling method will not cause ranking inversion between inliers and outliers, but increasing the 'gap' between them, this is a more important finding as summarized in below.

In a d-dimensional sphere of radius r containing n uniformly distributed points, the

expected Euclidean distance from a point to its k nearest neighbors, defined as $E\{kdist\}$ is:

$$E\{kdist\} = r\left(\frac{k}{n}\right)^{1/d}$$

Now, suppose we have two spheres in the data space with n_1 and n_2 objects. Suppose $n_1 << n_2$, therefore, the sphere with n_2 observations stands for the inliers, while the other one contains the outliers since outliers lie in sparse area. The expected kNN distances for the two spheres with n_1 and n_2 objects are:

$$E\{kdist_1\} = r\left(\frac{k}{n_1}\right)^{1/d};\ E\{kdist_2\} = r\left(\frac{k}{n_2}\right)^{1/d};$$

Taking a fraction $0 < m < 1$ of samples from the dataset D:

$$E\{kdist_1'\} = r\left(\frac{k}{n_1 \times m}\right)^{1/d};\ E\{kdist_2'\} = r\left(\frac{k}{n_2 \times m}\right)^{1/d};$$

The difference between the sampled distance and the original is:

$$\Delta_1 = E\{kdist_1'\} - E\{kdist_1\} = r\left(\frac{k}{n_1}\right)^{1/d}\left(\frac{1-m^{1/d}}{m^{1/d}}\right) \tag{2.1}$$

$$\Delta_2 = E\{kdist_2'\} - E\{kdist_2\} = r\left(\frac{k}{n_2}\right)^{1/d}\left(\frac{1-m^{1/d}}{m^{1/d}}\right) \tag{2.2}$$

The expected kNN distance in the sampled space increases as a function of sampling rate m:

$$\frac{\Delta_1}{E\{kdist_1\}} = \frac{\Delta_2}{E\{kdist_2\}} = \frac{1-m^{1/d}}{m^{1/d}} \tag{2.3}$$

Equations (2.1) and (2.2) justify that the expected k-NN distance after sampling will be larger for a sparse sphere than a dense one, which means that when $n_1 < n_2$, we have $\Delta_1 > \Delta_2$. This is equivalent to say that the sampling distances for outliers will grow more from normal objects since outliers lie in the sparse area while normal objects are in

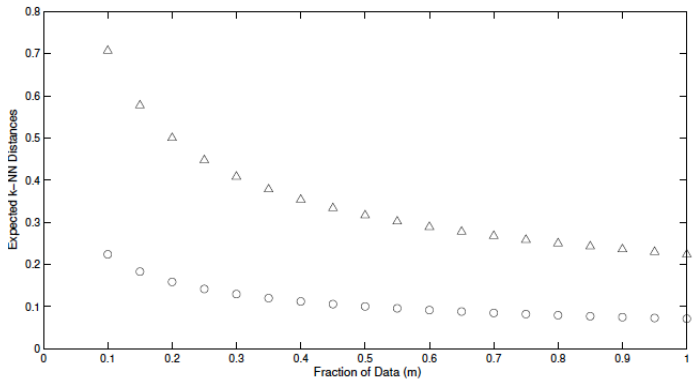


Fig. 2.5: Expected 5-NN distance for two spheres with radius r = 1, in a 2D Euclidean space, containing 1000m (circles) and 100m (triangles) objects uniformly distributed ($0 < m < 1$ is the sampling rate) [75]

the denser areas. This effect can be seen in Figure 2.5, where two spheres with the same radius are represented, but the one shown with circles is denser than the one shown with triangles, and the expected distances after sampling grow faster for the sparse sphere as the sampling rate decreases. These findings justify the attempt to improve outlier detection with a sampling technique, Algorithm 2 describes this procedure.

Algorithm 2 Random Subsampling

Data: Dataset D

Input: Sampling rate $0 < m < 1$; An anomaly detector A

Result: A vector of anomaly scores H

for $t = 1, ..., T$ **do**
  D_t = randomly pick $m \times |D|$ data points from D;
  for $x \in D$ **do**
    $H_t(x)$ = anomaly score of x obtained by applying detection algorithm A on $x \cup D_t$;
  end
end

for $x \in D$ **do**
  $H(x) = \frac{1}{T} \sum_{i=1}^{T} H_t(x)$
end

This approach uses the same base algorithm (LOF and its variants) for each iteration, and empirical results show that when the sampling fraction is 0.1, the anomaly detection performance is the best.

2.2.3 A Bootstrapping Approach – Proposed Algorithm

In statistics, bootstrapping techniques are commonly used for estimating the properties (mean, variance, confidence intervals, etc.) of the population using multiple samples [25]. Bootstrapping is a resampling method where multiple samples are drawn each time. The commonly used bootstrap method is the *.632 method*, which has been used to construct the training set and testing set for classification problems [33]. On resampling n points with replacement n times, 63.2% observations are presented in the training set while the other 36.8% observations are left for testing set. The number .632 comes from the approximate that $(1 - \frac{1}{n})^n$ approaches $e^{-1} = 0.368$ for a large n. In our method, we explore different resampling rates.

As shown in [75], applying distance-based outlier detection techniques on a subsampled data space will result in better performance than on the original dataset. They evaluate every point against the random subsample to avoid the chance that some objects will not be sampled. However, we argue that given enough number of random samples that the probability of missing an observation is very small; therefore, using a simple Bootstrapping approach will result in similar performance but be more efficient.

In our method, we select a subsample without repeating an observation. Hence, if we wish to select a proportion of m, $0 < m < 1$, then the probability of *not* selection in a sample is $(1 - m)$. The probability that each point is *not* drawn from T random samples is:

$$(1 - m)^T.$$

The probability that all points occur at least once in the T samples is, assuming that $T \times$

$m \times N >> N$:

$$(1 - (1 - m)^T)^N.$$

We want this probability to be greater than $1 - \delta$, where $0 < \delta < 1$ is a small number. To obtain the number of samples T, we derive the following:

$$\begin{aligned} \{1 - (1 - m)^T\}^N &\geq (1 - \delta) \\ 1 - (1 - m)^T &\geq (1 - \delta)^{1/N} \\ T &\geq \log_{1-m}\{1 - (1 - \delta)^{1/N}\} \end{aligned}$$

For example, if $\delta = 0.001$, $m = 0.1$, and $N = 1000$, then we need to sample 132 times to achieve desired contraints.

The algorithm is very simple and shown in Algorithm 3. As shown later in the evaluation section, this simple Bootstrapping approach exhibits the aforementioned subsampling effect for improving anomaly detection, and is faster than the approach in [75] since one loop is eliminated from the algorithm.

Algorithm 3 Bootstrapping Approach

Data: Dataset D

Input: Sampling rate $m, 0 < m < 1$; A set of anomaly detectors $\{A_1, ..., A_T\}$

Result: A vector of anomaly scores H

for $t = 1, ..., T$ **do**

D_t = by randomly select $m \times |D|$ data points from D;

H_t = anomaly scores of the datapoints in D_t are obtained by applying detection algorithm A_t;

end

for $x \in D$ **do**

$H(x) = \frac{1}{T} \sum_{i=1}^{T} H_t(x)$

end

2.3 Final Results Combination

Most anomaly detection algorithms output a score for each object indicating the degree of that object being an anomaly. Although different algorithms output scores with different scales, we assume here w.l.o.g. that in all cases the larger the score, the higher the probability that an object is an anomaly. How to combine the final scores to achieve a better result is very important in ensemble methods. Next, we review some of the most popular combination approaches and propose some new ideas.

2.3.1 Review of Earlier Methods

Feature bagging [44] is considered as the first ensemble approach for unsupervised anomaly detection [9]. In their approach, each base component is constructed by applying the same base algorithm on a random projection of the entire dataset on feature space. The paper evaluated two different combination approaches: cumulative sum and breadth first approach. The cumulative sum approach is the same as averaging approach and is the most

popular combination method in most existing works [18, 44, 75]. The work in [18] evaluates different combination methods such as using the maximum score output, using the LOF algorithm but with different values of k, the number of nearest neighbors.

2.3.2 Different Types of Combination Methods

Score Based Combination Methods

The score averaging method combines anomaly scores for each observation, first normalized to be between 0 and 1. Let $\alpha_i(x)$ be the normalized anomaly score of $\in D$, according to algorithm i. Then the normalized score, averaged over all T base components, is obtained as follows:

$$\alpha(x) = \frac{1}{T}\sum_{i=1}^{T}\alpha_i(x).$$

The maximum score combination method selects the maximum score output from the T base components for each object. It was shown in [18] to perform better than the minimum score and averaging score approach.

$$\alpha(x) = \max_{i=1}^{T}\alpha_i(x)$$

Rank Based Combination Methods

Since each algorithm outputs anomaly scores in different scales, the required score normalization in a score-based combination method may be biased. Using rank based methods, we can overcome this problem.

The minimum ranking approach considers the minimum rank of each object as the final output, instead of using the score output. Let the anomalous rank of x, assigned by algorithm i, be given by:

$$r_i(x) = |D| - |\{y|\alpha_i(y) < \alpha_i(x)\}|.$$

A smaller rank implies that the observation is more anomalous. The minimum rank method assigns

$$\text{rank}(x) = \min_{1 \le i \le T} r_i(x). \tag{2.4}$$

Thus, if object x is found to be the most anomalous by at least one algorithm, then the Min-rank method also declares it to be the most anomalous object. If all six algorithms give substantially different results, six different points may have the same rank.

The averaging ranking approach is similar to score averaging approach but considers the mean value of rankings over different base learners. The results obtained from this approach is not an integer anymore, we denote it as $rank'$. For any object x, the smaller the $rank'(x)$, the more it is considered anomalous.

$$\text{rank'}(x) = \frac{1}{T} \sum_{1 \le i \le T} r_i(x).$$

Majority Voting Rule Methods

As discussed earlier, the majority voting rule has a solid foundation, supporting the argument that the final combination reaches more robust decisions. Majority voting rules have been widely applied in the area of ensemble-based classification. However, the concept of majority voting has rarely been applied in the context of unsupervised outlier detection.

We propose to use a majority voting rule in which the basic idea is to throw away the outputs that are inconsistent with the rest of the base components. The challenge for designing a majority voting rule for unsupervised anomaly detection is that most anomaly detectors output a score instead of a clear decision. Since the number of anomalies (rare events) should be very small, we consider the ranked top $\tau\%$ as the 'true' anomalies for

each base learner, therefore the decision for each object x from the i^{th} component is:

$$H_i(x) = \begin{cases} 1, & \text{if } r_i(x) \leq \tau\% \times |D| \\ 0, & \text{otherwise.} \end{cases} \tag{2.5}$$

The final vote for each object x is:

$$V(x) = \sum_{i=1}^{T} H_i(x). \tag{2.6}$$

The final decision for whether an object x is anomalous is:

$$H(x) = \begin{cases} 1, & \text{if } V(x) > T/2 \\ 0, & \text{otherwise.} \end{cases} \tag{2.7}$$

When Equation (2.6) is used, the ranks of objects should be in descending order, *i.e.*, truly anomalous objects should have more 1's than 0's. A binary decision is output using Equation (2.7). In our evaluation, we denote the method using Equation (2.6) as *majority*, while the results obtained by Equation (2.7) as *majority bin* since it output a binary decision.

The weighted majority voting approach: In the previous combination methods, each learner is assigned an equal weight. By doing so, each algorithm gets to vote for the final decision. However, given the assumption that most base learners are accurate at detecting anomalies, we assign more weight to the algorithms that agree more with the majority, hence there is a potential that the final decision reaches 'closer' to the truth. The weight assignment is based on the idea that if a base learner detects the same set of anomalies as the majority, it gets a higher weight than the one that often disagrees with the majority. To do so, we design a weight assignment w_i for each base algorithm i as:

$$w_i = \frac{\sum_{x \in D} I(H_i(x), H(x))}{\sum_{x \in D} H(x)} \tag{2.8}$$

where $H_i(x)$ is the decision of the i^{th} algorithm about whether x is an anomaly, as defined in Equation (2.5). $H(x)$ is the decision reached by majority, as defined in Equation (2.7). I is a function defined as:

$$I(x,y) = \begin{cases} 1, & \text{if } x = y; \\ 0, & \text{otherwise.} \end{cases}$$

After such assignment, the algorithms that agree more often with the majority will be assigned a higher weight than the others.

Other Combination Methods

The breadth first approach [44] sorts all the results obtained from different base algorithms in descending order, then selects the next object that has the highest anomaly score from all the base algorithms, in a breadth first searching order. An illustration is shown in Fig 2.6 with three base algorithms. ALG1 ranks x as most anomalous, z as second, and y as the third, *i.e.* $r_1(x) < r_1(z) < r_1(y)$. For ALG2, it has $r_2(x) < r_2(y) < r_2(z)$, while in ALG3, $r_3(z) < r_3(x) < r_3(y)$. To obtain the final result, the method 'scans' each algorithm horizontally, in the figure. It finds the first anomaly obtained by ALG1, which is x, then, it goes horizontally to ALG2, which is x again, when it reaches ALG3, the second anomaly z is output; then, the method continues in the next row, z is already obtained as anomaly, so it is ignored, the third anomaly is y obtained by second row of ALG2. The final output is x, z, y.

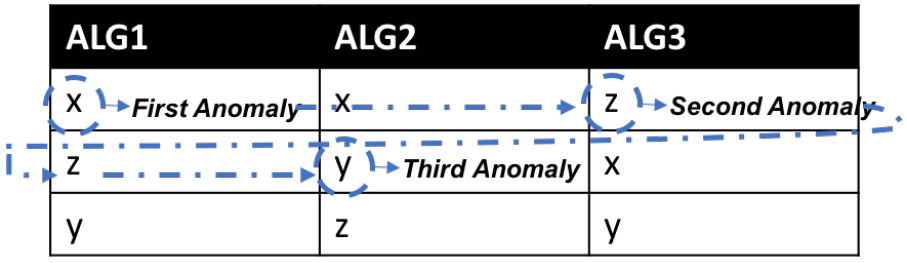


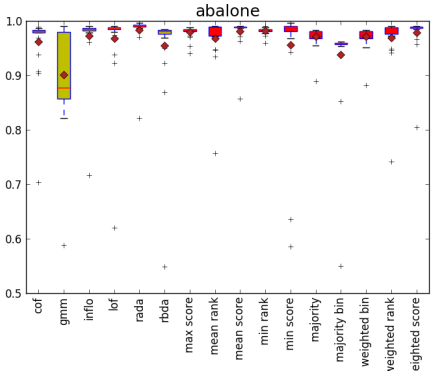
Fig. 2.6: An illustration for breadth first approach

2.4 Evaluation of Independent Ensemble Methods

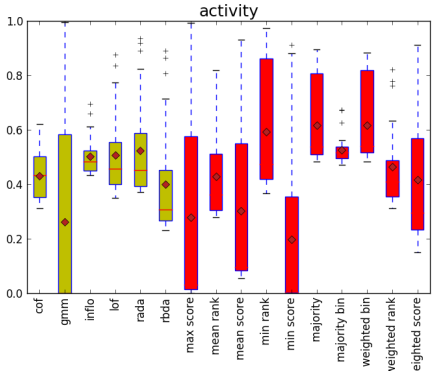
The datasets we use in this section are summarized in Table 1.1. For each dataset, we vary k values from 1 to 25 for the kNN based algorithms. For GMM, we execute 25 random trials.

2.4.1 Performance over Different Combination Methods

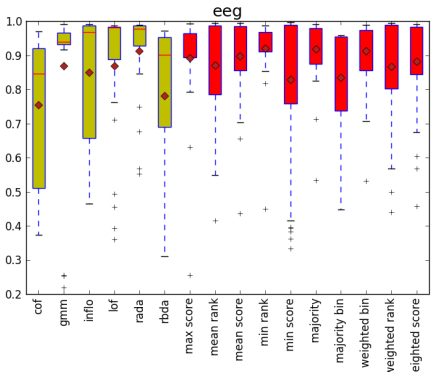
We plot the AUC scores (defined in Section 4.5) for variant individual and ensemble methods, using boxplots for each dataset in Figure 2.7. We observe that among the individual base algorithms, RADA performs the best. The reason is that RADA itself is an ensemble which considers both distances and ranks for detecting anomalies. However, on some datasets, *e.g.*, on NBA and SAT datasets, GMM performs better than RADA, which shows that if we choose RADA in all cases, then it does not perform well on these datasets. Using ensemble methods, on the other hand, might not beat the best individual base algorithm in all the cases, but it generates more robust solutions than the individual base algorithms. To summarize these results, we show the AUC scores for each dataset in Table 2.1, we observe that the combination method *min rank* generates the best AUC at 0.832 ± 0.179 while the best base algorithm RADA generates 0.828 ± 0.217. We also observe that though weighted methods are not the best, but they are perform better than the pure averaging methods: *weighted score* has AUC at 0.791 ± 0.214 while the pure *mean score* has 0.781 ± 0.233; *weighted rank* reaches 0.778 ± 0.202 while *mean rank* has 0.774 ± 0.204. Also, we observe that when the base methods are all not performing well, for example, on Popularity dataset, while RADA gets an AUC of 0.628 and it is not the best among all the base algorithms, however, using our ensemble method *min rank* achieves an AUC at 0.843 which indicates that using an ensemble can achieve more robust results than a single algorithm.



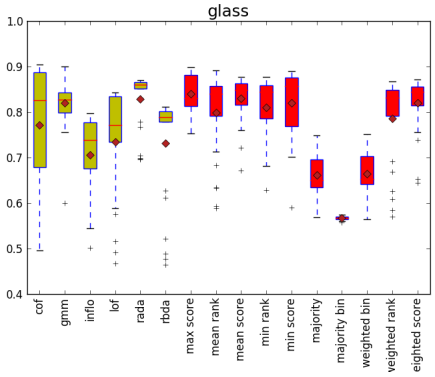
(a) Abalone



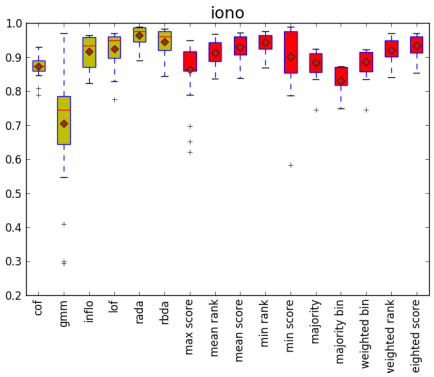
(b) Activity



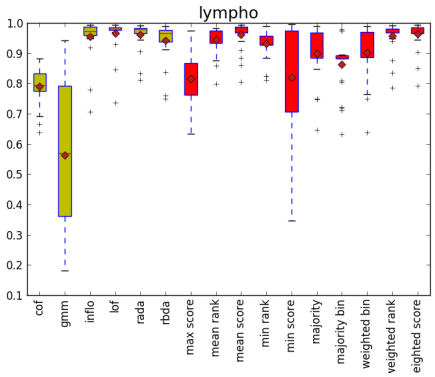
(c) EEG



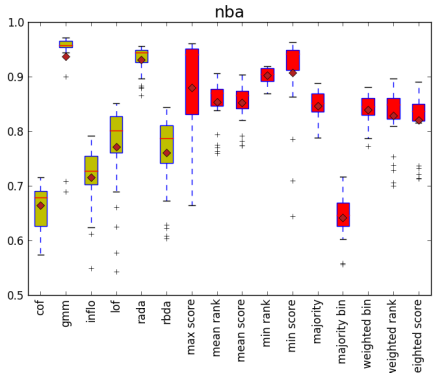
(d) Glass



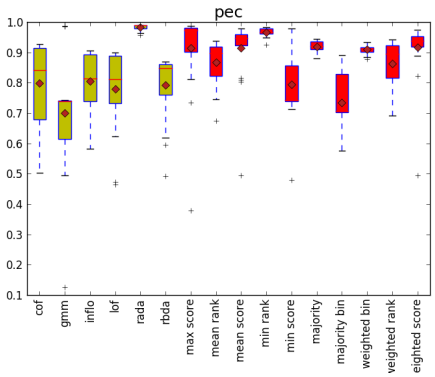
(e) Ionosphere



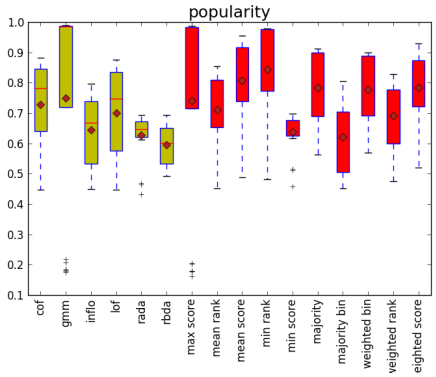
(f) Lympho



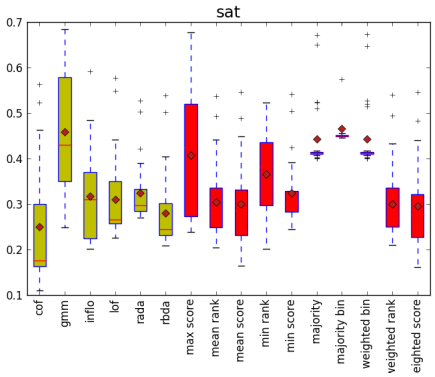
(g) NBA



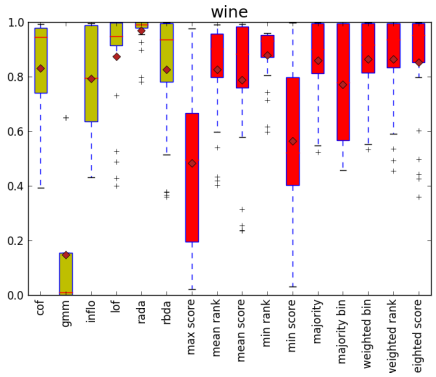
(h) Pec



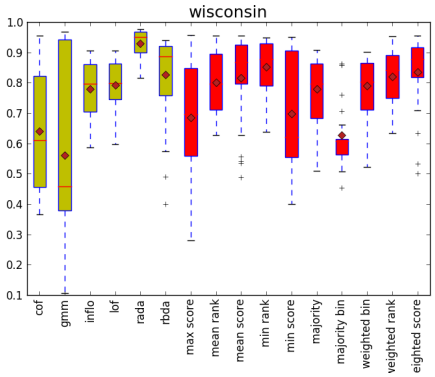
(i) Popularity



(j) Sat



(k) Wine



(l) Wisconsin

Fig. 2.7: AUC Performance comparison over all methods for different datasets

2.4.2 Performance for Bootstrapping Methods

Performance vs. Previous Work

We compare our proposed bootstrapping method with Zimek's work [75], where they use 25 iterations and sample rate $m = 0.1$ to obtain the best performance. Therefore, for the first set of experiments, we use these settings for Zimek's method and use $\delta = 0.0001$ for our bootstrapping method. Results shown in Table 2.2 shows that for all the datasets, the averaged AUC is about the same for both methods, but our method is much faster than Zimek's since we do not examine each object with respect to the sample in each iteration.

Performance using different Combination Methods

We evaluate the performance of different combination methods compared to the bootstrapping algorithm. For these experiments, $k = 5$ is used and we vary the sampling rate m to be 0.1, 0.3, 0.5, 0.7, and 0.9, respectively. For each dataset, we calculated the averaged AUC score and show the results in Table 2.3. Results show that *max score* combination gives us the best performance, while *min rank* gives us the second best performance. In comparison, our previous results in Table 2.1 showed that using *min rank* results in the best ensemble method. Recall that the reason we use ranking based method is to avoid the score normalization problem; in bootstrapping algorithm, the base method is of the same scale (we use LOF for all the experiments), therefore, both *max score* and *min rank* give good results.

Performance vs. Sampling Rate

To examine the relationship between performance and different sampling rate, we design the following experiments. We fix the combination method to be *max score*, since it generates the best performance in our previous section. Then, for each sampling rate $m = 0.1, 0.3, 0.5, 0.7,$ and 0.9, we repeat our algorithms 20 times and compare the aver-

aged AUC score in Table 2.4. As in Zimek's work [75], we also find that using a small sampling rate tends to generate better performance.

2.5 Conclusion

In this chapter, we have evaluated multiple independent ensemble methods for unsupervised anomaly detection. We have used three different types of anomaly detection algorithms including density-based, rank-based, and statistical-based algorithms as the base learners. Instead of the commonly used score averaging method for final results combination, we proposed and evaluated multiple combination approaches based on scores, rank aggregation, and majority rule voting. We also proposed weight assignment for different methods based how much they agree with the majority voting rule. Our empirical study shows that using *minimum ranking* combination generates the best ensemble performance. Recently, sampling based methods were proposed for anomaly detection, which was justified in the previous work both theoretically and empirically to be very effective for anomaly detection. We proposed a bootstrapping method which adopted the idea of random subsampling and achieves comparable results while reducing the runtime significantly. We then proposed to combine the results from our previous combination techniques and bootstrapping algorithms; our evaluation shows that using *maximum score* or *minimum rank* generates the best performance among all the combination methods.

Table 2.1: Summary of results for all methods over all datasets

Dataset	Abalone	Activity	Eeg	Glass	Iono	Lympho	Nba	Pec	Popularity	Sat	Wine	Wisconsin	AVG	STD
weighted majority	0.972	0.616	0.912	0.664	0.886	0.902	0.839	0.911	0.778	0.443	0.864	0.790	0.798	0.153
majority bin	0.938	0.525	0.833	0.567	0.831	0.863	0.642	0.734	0.621	0.465	0.771	0.627	0.701	0.148
majority	0.972	0.617	0.917	0.661	0.884	0.900	0.846	0.919	0.783	0.443	0.859	0.779	0.798	0.154
weighted score	0.979	0.417	0.883	0.819	0.933	0.963	0.820	0.916	0.784	0.295	0.853	0.835	0.791	0.214
mean score	0.981	0.301	0.898	0.830	0.929	0.961	0.852	0.914	0.807	0.300	0.788	0.816	0.781	0.233
max score	0.980	0.278	0.892	0.839	0.863	0.817	0.880	0.913	0.742	0.408	0.484	0.685	0.732	0.224
min score	0.956	0.198	0.828	0.820	0.901	0.819	0.907	0.795	0.637	0.324	0.565	0.698	0.704	0.237
min rank	0.982	0.592	0.921	0.809	0.943	0.931	0.902	0.966	0.843	0.366	0.880	0.852	**0.832**	**0.179**
weighted rank	0.969	0.464	0.866	0.786	0.921	0.958	0.829	0.863	0.692	0.300	0.864	0.821	0.778	0.202
mean rank	0.968	0.428	0.870	0.799	0.913	0.945	0.853	0.867	0.710	0.305	0.826	0.800	0.774	0.204
RBDA	0.955	0.401	0.782	0.731	0.945	0.942	0.761	0.793	0.596	0.280	0.825	0.826	0.736	0.212
RADA	0.983	0.524	0.913	0.829	0.963	0.962	0.931	0.983	0.628	0.324	0.968	0.930	**0.828**	**0.217**
LOF	0.967	0.506	0.868	0.734	0.924	0.965	0.772	0.780	0.700	0.311	0.873	0.792	0.766	0.193
INFLO	0.973	0.503	0.849	0.706	0.916	0.954	0.715	0.805	0.644	0.317	0.794	0.778	0.746	0.190
GMM	0.901	0.263	0.868	0.819	0.705	0.564	0.937	0.699	0.749	0.459	0.147	0.562	0.639	0.251
COF	0.962	0.431	0.754	0.772	0.873	0.790	0.664	0.799	0.727	0.250	0.831	0.640	0.708	0.196

Table 2.2: Comparison between work in [75] and Bootstrapping: $m = 0.1$, $k = 5$, $\delta = 0.001$

	Base Algorithm	Zimek			bootstrap		
dataset	AUC	avg AUC	std AUC	time (sec)	avg AUC	std AUC	time (sec)
activity	0.528	0.989	0.003	52901	0.990	0.001	184
abalone	0.980	0.983	0.002	1151	0.981	0.003	15
eeg	0.710	0.990	0.002	3768	0.990	0.002	31
glass	0.457	0.781	0.012	181	0.785	0.019	5
iono	0.895	0.965	0.010	250	0.960	0.010	7
lympho	0.978	0.975	0.009	93	0.965	0.010	4
nba	0.690	0.851	0.033	1722	0.840	0.019	20
pec	0.664	0.887	0.024	7406	0.893	0.009	49
popularity	0.820	0.707	0.042	7566	0.689	0.029	51
sat	0.398	0.434	0.053	1079	0.428	0.039	14
wine	0.734	0.999	0.001	73	0.997	0.002	4
wisconsin	0.744	0.974	0.004	197	0.973	0.004	6

Table 2.3: Bootstrapping performance over different combination methods: averaged over different values of $m = 0.1, 0.3, 0.5, 0.7, 0.9$

	abalone	activity	eeg	glass	iono	lympho	nba	pec	popularity	sat	wine	wisconsin	avg	std
max_score	0.984	0.529	0.830	0.648	0.891	0.977	0.738	0.676	0.861	0.369	0.867	0.737	**0.759**	**0.175**
min_score	0.982	0.454	0.720	0.506	0.893	0.979	0.716	0.721	0.818	0.374	0.772	0.741	0.723	0.186
mean_score	0.982	0.500	0.786	0.579	0.894	0.979	0.724	0.664	0.848	0.378	0.850	0.743	0.744	0.179
weighted_score	0.981	0.523	0.766	0.581	0.891	0.971	0.699	0.658	0.841	0.417	0.822	0.731	0.740	0.168
mean_rank	0.982	0.502	0.762	0.543	0.894	0.980	0.720	0.689	0.839	0.384	0.818	0.742	0.738	0.178
min_rank	0.981	0.536	0.814	0.620	0.881	0.973	0.726	0.650	0.851	0.410	0.858	0.713	0.751	0.167
weighted_rank	0.982	0.502	0.761	0.542	0.894	0.980	0.720	0.688	0.839	0.388	0.815	0.741	0.738	0.178
majority	0.960	0.571	0.596	0.581	0.793	0.878	0.589	0.650	0.695	0.457	0.683	0.498	0.663	0.144
majority_bin	0.883	0.562	0.477	0.593	0.770	0.794	0.556	0.666	0.574	0.474	0.501	0.478	0.611	0.132
weighted_bin	0.961	0.571	0.597	0.581	0.793	0.878	0.589	0.650	0.695	0.457	0.682	0.498	0.663	0.144

Table 2.4: AUC score over all datasets for different sampling rate m

Sampling rate m	avg AUC	std AUC
0.9	0.759	0.175
0.7	0.796	0.190
0.5	0.811	0.199
0.3	0.862	0.179
0.1	0.863	0.146

CHAPTER 3
SEQUENTIAL ENSEMBLE METHODS FOR ANOMALY DETECTION

In Chapter 2, we discussed the usage of independent ensemble methods for unsupervised anomaly detection. One of the assumptions one needs for using independent ensemble methods is that the base components in ensembles are independent of each other, which, however, might be difficult to attain in reality. Introducing randomness into ensembles, as in the subsampling approach, decreases the dependence between learners; however, due to the reason that they are perturbations of the same general idea, there exists inherent dependence between the base learners. Hence, other approaches, which take such dependence into consideration, needed to be explored. Sequential learning, on the other hand, exploits the dependence between base components. In this chapter, we focus on exploiting the diversity between the learners and how to improve one algorithm's performance by applying another algorithm to its outputs. The general idea of sequential learning is like a pipeline: the following algorithms' inputs are generated from the former algorithms' outputs. It can be traced back to 1977 Tukey's Twicing's ensemble of two linear regression models [68], where he used the first regression model to fit the original data and the second for the residuals. The concept of sequential learning in classification was first brought up by Freund and

Schapire in their famous AdaBoost [29] paper. We discuss boosting learning in Chapter 4.

Sequential ensembles are not yet well-explored in the literature of outlier detection [9, 10]. One of the previous works [12] is considered a sequential ensemble method applied in the area of intrusion detection. In their work, they first apply association rule mining on the original internet traffic data to find the frequent itemsets and assume them to be normal (attack-free) data; then, they use these connections as the clustering seed points in the next clustering step to find more robust clusters of normal points for outlier detection. Another method was proposed in [48] which recursively explores the statistically relevant feature subspaces, and combines the results at the end. These methods are instances of sequential data-centered ensemble methods in the outlier detection context, but there is no previous work done in the area of sequential model-centered context as reported in [9]. In this chapter, we propose several algorithms that adopt the idea of sequential ensemble methods involving both data-centered and model-centered techniques. The base models (algorithms) we use are the recently proposed two families of outlier detection algorithms: density-based and rank-based outlier detection, as reviewed in Section1.2; in order to select diverse algorithms used in sequential ensemble, we examine the similarities between each pair among them, and propose that using diverse learners generate better ensemble results. To introduce ensemble diversity at the data-level, we adopt the subsampling approach proposed in Zimek *et al.* [75] such that drawing multiple subsamples from the dataset and combining the output as the final outlier detection is more efficient than detection on the whole dataset.

We categorize our proposed sequential ensemble methods into two classes: single-layer sequential and multi-layer sequential, in which each layer generates a decision for anomaly detection. For single-layer methods, we start off with the original dataset, then apply multiple algorithms where the subsequent algorithm's inputs are generated from the most recent algorithm's outputs. For multi-layer methods, an intermediate result is generated from a previous ensemble model, then another algorithm uses these results as inputs for the final

stage of anomaly detection.

3.1 Single-layer Sequential ensemble algorithms

A generalized framework for sequential applications using multiple base algorithms is illustrated in Figure 3.1. T base algorithms can be applied, and at each step, algorithm ALG_i is applied and an intermediate result Res_i is generated. A transformation function g can be applied to Res_i to generate the input D_i for the next algorithm ALG_{i+1}. Though multiple base algorithms can be used in sequential ensembles, we mainly explore a two-step sequential learning where the second algorithm tends to 'correct' the first algorithm's errors.

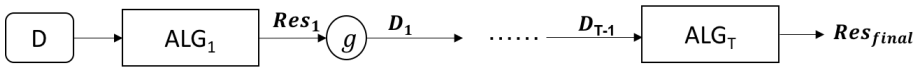


Fig. 3.1: Framework for sequential ensemble method

In this section, we describe our proposed two-phase sequential ensemble algorithm for unsupervised anomaly detection.

3.1.1 Sequential Application of Two algorithms - A Sieve Method

The first approach of sequential applications is that we use the output of one algorithm on the original dataset D, extract a subset of D, and use another algorithm on the extracted dataset. Our argument is that similar algorithms are likely to make similar errors; by sequentially applying two substantially different algorithms, the second algorithm may 'correct' the previous one's errors and thus provide better results. As in the previous chapters, for each object in the dataset the ranks are calculated using the anomaly scores. In the sequential method, we run the first algorithm on the whole dataset, and obtain the ranks of all observations. Next we consider the dataset D_β obtained by retaining most anomalous

fraction (β) of D *i.e.*, those that suspected to contain most or all anomalies. The second detection algorithm calculates the anomaly scores of all objects in D with reference to D_β. In the following discussion, this algorithm is referred to as Sequential-1, and is described in Algorithm 4.

Algorithm 4 Sequential-1 Algorithm

Data: Dataset D

Result: a vector of scores $\mathbf{H}$ associated with each of the objects in D

Score vector $\mathbf{H_A}$ is obtained by applying algorithm A on D;

Rank vector $\mathbf{R_A}$ = $\{r_A(x) \mid \forall x \in D\}$; *objects are sorted in decreasing order of* H_A.

D_β = $\{x \mid r_A(x) < \beta \cdot |D|\}$; *i.e. retrieve the top ranked data.*

for *all* $x \in D$ **do**

| $H(x)$ = calculate the anomaly score of x by applying algorithm B on dataset $\{x\} \cup D_\beta$;

end

return H

3.1.2 Sub-sampling and Sequential Method

As described in Chapter 2, Zimek *et al.* [75] have argued that anomaly detection performance on a subsample of the dataset could be better than detection performance on the whole dataset. They select a random sample from the dataset D and evaluate the anomaly score of each object in D with reference to the data in the sample; they repeat the above experiment multiple times and report that the average score gives much better performance. In our second sequential approach, we take a random subsample of percentage γ from D_β and evaluate the anomaly score of each $x \in D$ using another algorithm with respect to the subsample. As in Zimek *et al.* [75], we repeat this T times and evaluate the average score, used in the final anomaly ranking. This algorithm is referred as Sequential-2, is described in Algorithm 5.

Algorithm 5 Sequential-2 Algorithm

Data: Dataset D

Result: a vector of scores $\mathbf{H}$ associated with each of the objects in D

Score vector $\mathbf{H_A}$ is obtained by applying algorithm A on D;

Rank vector $\mathbf{R_A}$ = $\{r_A(x) \,|\, \forall x \in D\}$; *objects are sorted in decreasing order of* H_A.

D_β = $\{x \,|\, r_A(x) < \beta \cdot |D|\}$; *i.e. retrieve the top ranked data.*

for *all* $x \in D$ **do**

  for $i = \{1, ..., T\}$ **do**

    D_γ = randomly pick $\gamma \cdot |D_\beta|$ objects from D_β;

    $H_i(x)$ = apply algorithm B on dataset $\{x\} \cup D_\gamma$; *i.e. obtain the score of* x *from the* i^{th} *iteration;*

  end

end

for $\forall x \in D$ **do**

  $H(x) = \frac{1}{T}\Sigma_{i=1}^{T} H_i(x)$;

end

return H

3.2 Multi-layer Sequential Ensembles

In the previous sections, we start off our detection process with input of the entire dataset D, then, refine the results in later steps sequentially. There is another possibility for sequential application, *i.e.*, the so-called multi-layer sequential methods. An illustration of the framework is shown in Figure 3.2, where we have a two-layer structure. At the first layer, it is equivalent to our previous independent ensemble methods such that T independent base algorithms are applied on thie same dataset D, a consensus function f can be applied to generate the first-layer decision. The first-layer application generates diverse ensembles on model level since it combines the results from multiple base algorithms. Then,

a data transformation function g is applied based on the results generated from f, in our experiments, we use the subsampling function as g to generate ensemble diversity on the data layer. A new data sample D' is generated from $g \cdot f$, for the second layer algorithm ALG_2 to apply on. Final result Res_{final} is used for final decision.

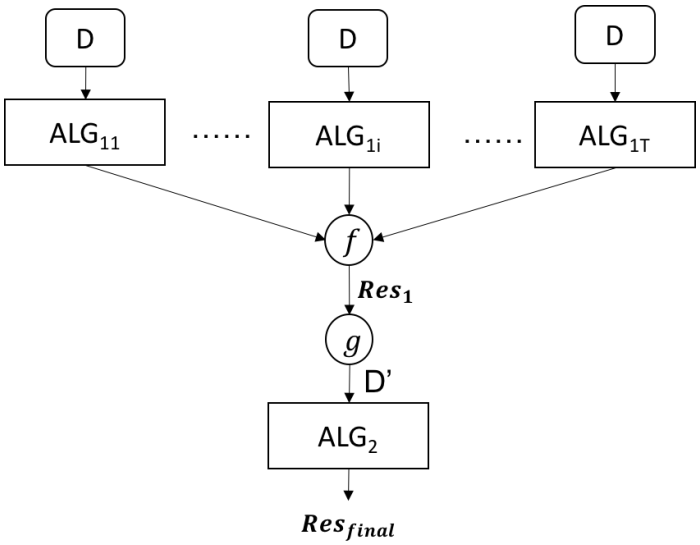


Fig. 3.2: Multi-layer sequential model

3.3 Evaluation of Sequential Ensemble Algorithms

In both of our algorithms, we select the top ranked $\beta \cdot |D|$ observations as those suspected to contain anomalies. As a result, β is an essential parameter in our algorithms, before we discuss our evaluation results, we first consider how to select a reasonable value for β.

3.3.1 Detection accuracy with the top ranked β observations

Unlike classification or clustering, there is no clear decision boundary for most anomaly detection algorithms, instead they often output a score that indicates the degree of outlierness. The first question we are interested in is how to determine whether a detector is accurate in successfully detecting most/all anomalies.

The i^{th} anomaly detection algorithm ALG_i outputs a real valued score for each object x, denoted as $\alpha_i(x)$, and the rank of that object is defined as:

$$r_i(x) = |D| - |\{y|\alpha_i(y) < \alpha_i(x)\}|.$$

In theory, the true anomaly should have the least rank, for example, if there are 5% anomalies, they should be ranked as the topmost 5% by a perfect detection algorithm. Although unsupervised detection algorithms often suffer from high false positives/negatives (due to the lack of training labels, or because other assumptions fail to hold), we should expect that a reasonable detector must identify the true anomalies within the top ranks. For instance, if there are 5% anomalies in the dataset, a detector might not be perfect, but it should be able to rank the easy-to-detect anomalies in the top 2%, and the rest of the 3% in the topped 10%, in order for it to be considered to be an accurate detector. Denoting the relevant fraction as β, we expect the following for an accurate algorithm ALG_i:

$$Pr(r_i(x) > |D| \cdot \beta | x \in \mathbb{O}) < f(\beta),$$

where $\mathbb{O}$ is the set of anomalies, $|D|$ is the number of observations in dataset D, f is a function of β. In theory, the number of anomalies should be decreasing as β increasing since most anomalies should be ranked in the topped percentage. In the following, we analyse the relationship between β and detection rate on both synthetic and real-world datasets when ground truth is available.

3.3.2 Relationship between Detection Rate and β on Synthetic Datasets

To study the relationship between detection accuracy and β, we first construct a synthetic dataset. Where the normal observations are drawn from a 2D Gaussian distribution with

$\mu = 0$, and $\sigma = 1$, then we randomly generate 1% of the dataset to be anomalies that lie on the circle whose radius R of the circle is proportional to σ. Each anomaly point i(x_a, y_a) is generated by:

$$\begin{cases} x_a = R \times \cos(2\pi\phi) + \mu \\ y_a = R \times \sin(2\pi\phi) + \mu \end{cases} \tag{3.1}$$

where $\phi \sim U(0, 1)$, $\mu = 0$.

The idea of generating such datasets is to evaluate the relationship between β and number of anomalies for easy-to-detect or hard-to-detect anomalies. If the anomalies lie farther away from the gaussian cluster, they are easier to detect. For example, the synthetic dataset shown in Figure 3.3 has 1% outliers on a circle with radius of $R = 4\sigma$ and those outliers are considered to be easy-to-detect. We created two synthetic datasets with $R = 3\sigma$ and $R = 4\sigma$, the intuition is that the latter should be easier to detect than the former, which should be reflected in the relative ranks in the two cases. From the Figure 3.4(b), we observe that all the anomalies are detected within top 5% when $R = 4\sigma$ for all algorithms, which indicates that the base algorithms are very accurate in detecting the easy-to-detect anomalies. However, as R decreases to 3σ, the detection difficulty increases the anomalies are detected within 30% for INFLO and COF, captured within 20% for LOF and RBDA, but RADA does much better since all the anomalies are captured within the top 5%.

3.3.3 Relationship between detection rate and β on real-world datasets

We now evaluate how the different base detectors work on real-world datasets. We observe from Figure 3.5 that most of the anomalies are detected within the top 20%. However, in a dataset D, number of anomalies are often few. Thus if we apply the commonly used top N method, and use a threshold to 20% for identifying anomalies, we still suffer a high false positive.

Above discussions show that the real outliers and some false positives are mixed within

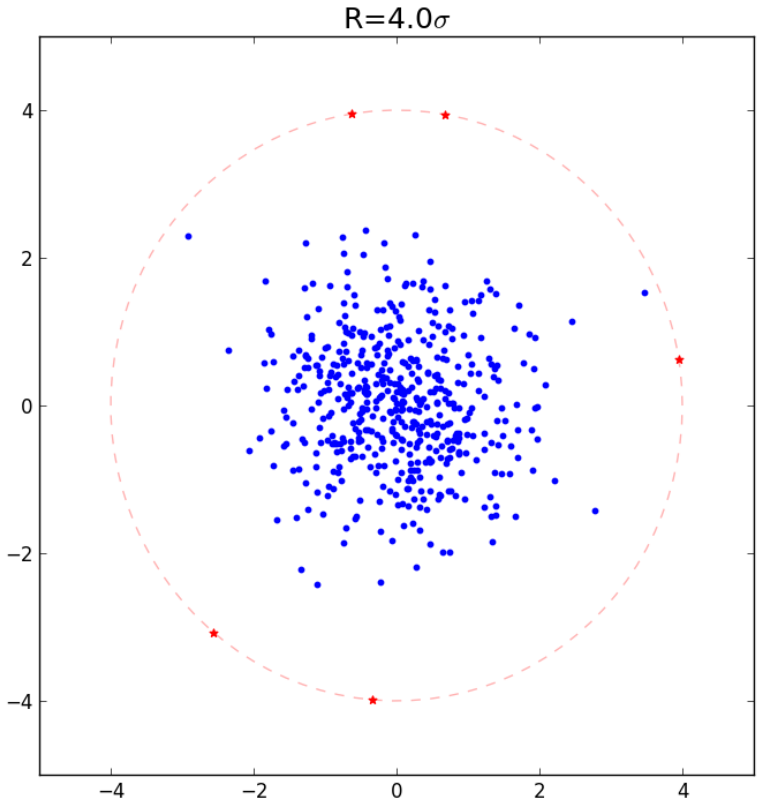
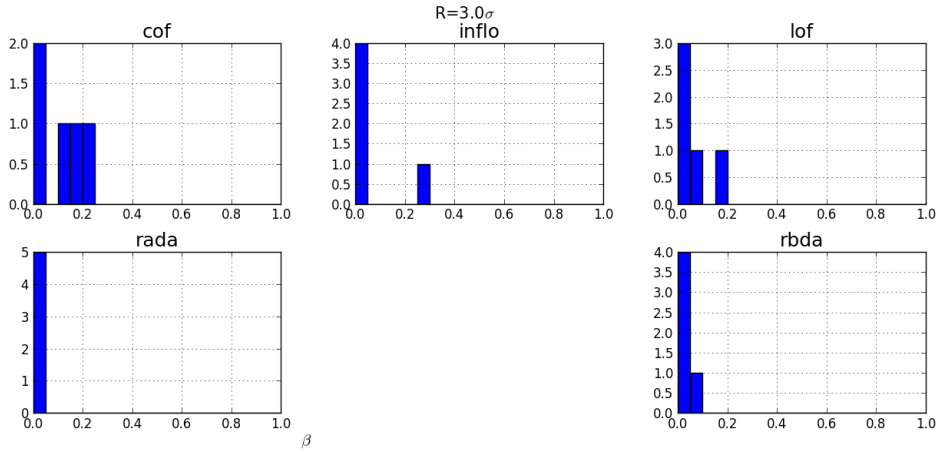


Fig. 3.3: Synthetic dataset: outliers are generated on circle with $R = 4\sigma$

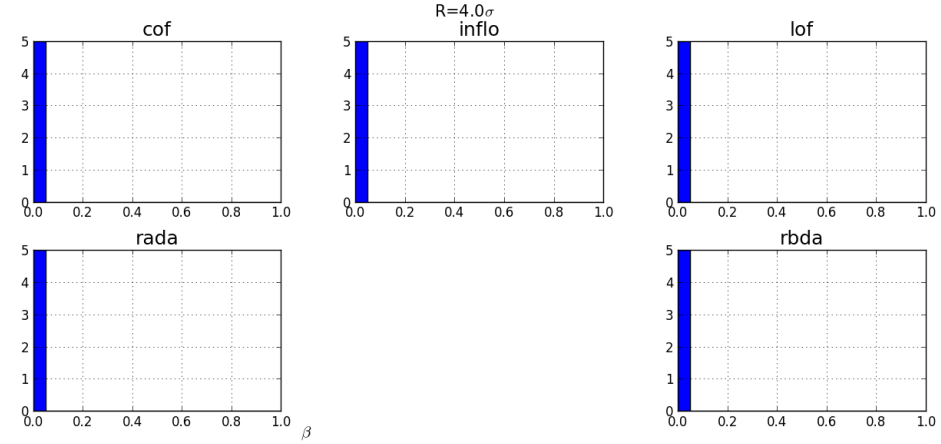
the top-ranked data points, requiring another algorithm to discriminate among these. Our sequential algorithms proposed in Algorithm 4 and Algorithm 5 intend to solve this problem. In the following, we evaluate both of our algorithms on multiple benchmark datasets.

3.3.4 Evaluation of Sequential-1 Method

In our empirical study, we observe that COF or LOF followed by RADA generates the best results among all the combinations for sequential algorithms. The reason is that RADA is from a different family of algorithms from LOF and COF, also, in general RADA generates the best solutions among all the individual algorithms in general as shown in the Chapter 2 in Table 2.1. This shows that using a more accurate and diverse algorithm as the second



(a) $R = 3.0\sigma$

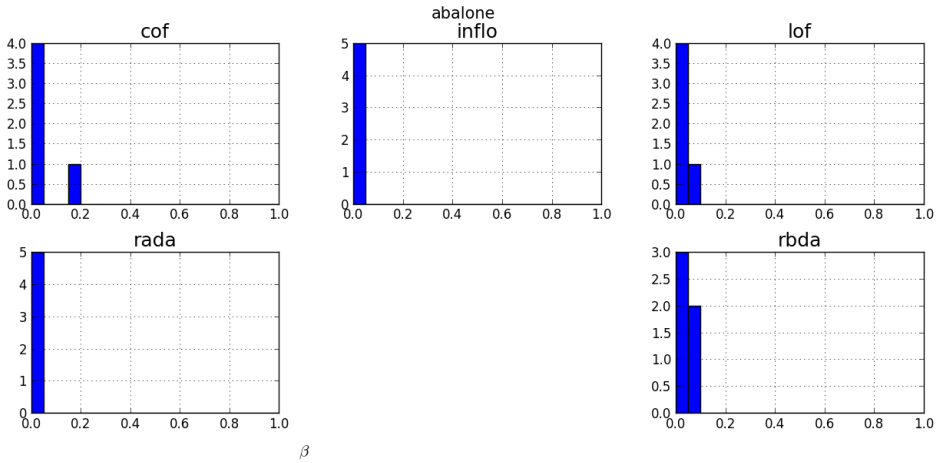


(b) $R = 4.0\sigma$

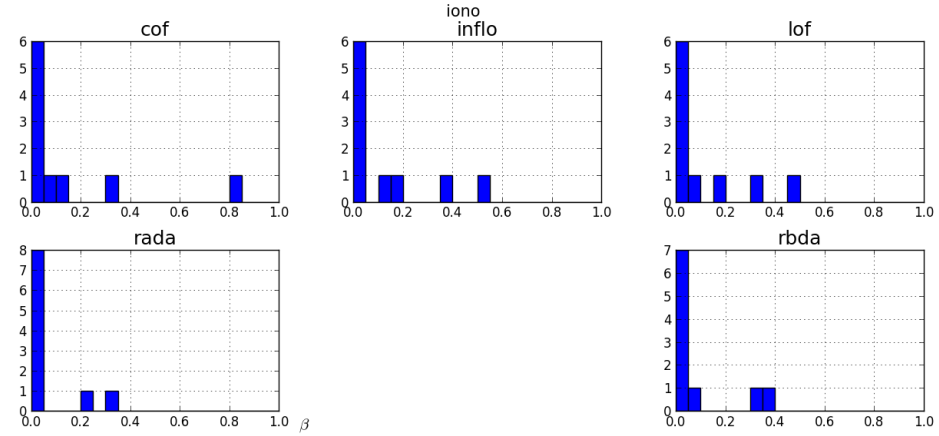
Fig. 3.4: Relationship between detection rate and β on synthetic data

algorithm to 'correct' first algorithm's errors has the potential of detecting more anomalies than application of individual algorithm.

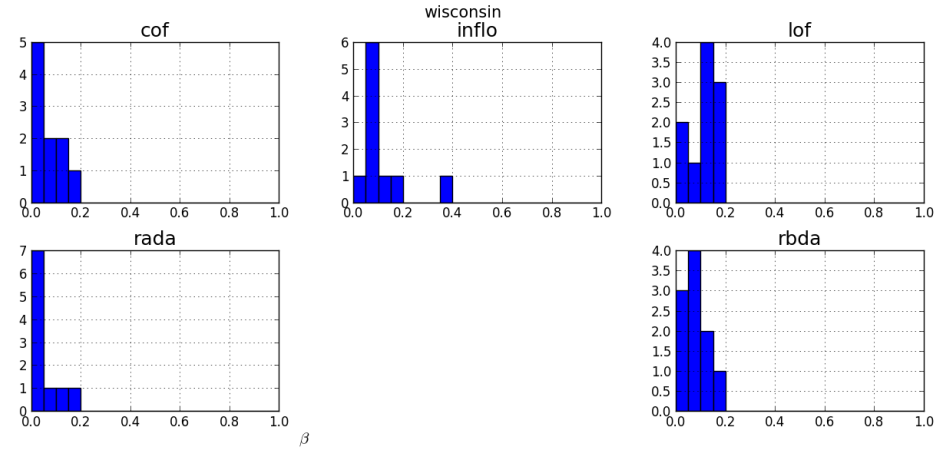
Our empirical study shows that when β ranges from 0.1 to 0.3, the pairs (LOF, RADA) and (COF, RADA) perform best. In most datasets, the Sequential-1 algorithm results in better detection performance than each of the individual algorithms. Note that on abalone dataset, the base algorithms are already very accurate, therefore, by applying sequential method, it is difficult to get higher accuracy.



(a) abalone dataset



(b) ionosphere dataset



(c) wisconsin dataset

Fig. 3.5: Relationship between detection rate and β on real-world data

Table 3.1: Summary of Sequential-1 algorithm for different β values when k=5

k=5	Base Algorithms			$\beta = 0.1$		$\beta = 0.2$		$\beta = 0.3$	
Dataset	LOF	COF	RADA	L-R	C-R	L-R	C-R	L-R	C-R
Abalone	0.980	0.962	0.991	0.948	0.964	0.973	0.977	0.975	0.976
Activity	0.528	0.534	0.413	0.962	0.998	0.764	0.702	0.754	0.727
Eeg	0.710	0.467	0.846	0.987	0.930	0.987	0.965	0.971	0.857
Glass	0.457	0.492	0.691	0.826	0.817	0.853	0.855	0.911	0.920
Iono	0.895	0.869	0.949	0.692	0.767	0.923	0.949	0.972	0.969
Lympho	0.978	0.718	0.966	0.191	0.928	0.539	0.945	0.685	0.851
Nba	0.690	0.573	0.896	0.941	0.884	0.826	0.896	0.788	0.912
Sat	0.398	0.399	0.369	0.851	0.354	0.305	0.322	0.487	0.336
Pec	0.664	0.678	0.971	0.995	0.995	0.996	0.996	0.995	0.994
Popularity	0.820	0.447	0.664	0.558	0.545	0.355	0.614	0.399	0.633
Wisconsin	0.744	0.418	0.900	0.961	0.978	0.975	0.975	0.965	0.977
Wine	0.734	0.382	0.955	0.999	0.993	0.790	0.995	0.731	0.977

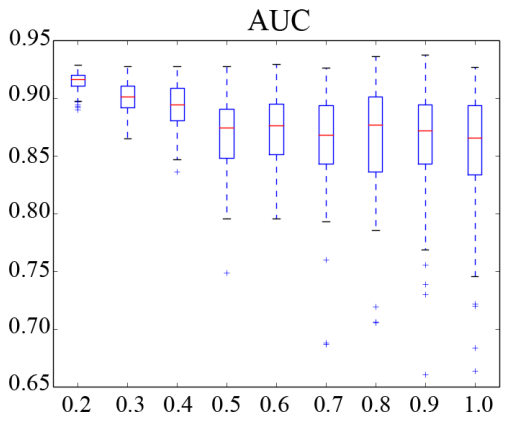
3.3.5 Sub-sampling Approach (Sequential-2 Method)

To evaluate if $p \in D$ is an anomaly Zimek *et al.* [75] have proposed to repeatedly (typically 25 times) draw a sample of size 100 $\times\theta\%$ from the original dataset D. We select a sample of size 100 $\times\theta\%$ from D_α and, as in their approach, compute the anomaly score for each object with respect to the sub-sampled data. Zimek *et al.* have argued that the performance will be good when random sampling is performed 25 times or more. Our experiment indicates that it is sufficient to use as few as 5 subsamples from D_α. We use $k = 2$ and LOF to provide initial rankings and evaluate the performance for multiple values of $\alpha = 0.2$, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0 to measure the effect of this parameter[1]. For comparison, we run each experiment 100 times; Figure 3.6 represents the corresponding box-plots for AUC scores.

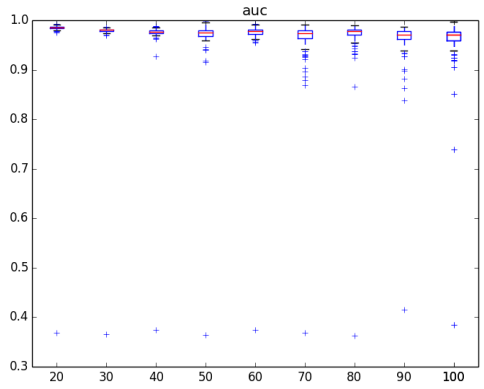
Comparison of Sequential-1 and Sequential-2

Now we compare the performance in AUC score for Sequential-1 algorithm and Sequential-2 algorithm. We choose $\gamma = 0.1$ because in our previous evaluations for bootstrapping that

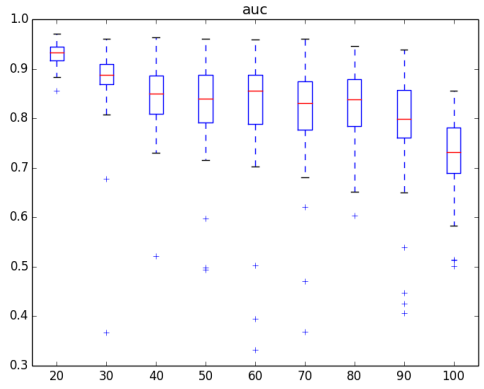
[1]When α=1.0, our approach is equivalent to Zimek's random selection from the whole dataset.



(a) AUC of Wisconsin



(b) AUC of KDD



(c) AUC of PEC

Fig. 3.6: Performance of subsampling approach when samples are drawn from D_β for different values of β, for three datasets.

drawing multiple samples with sampling rate 0.1 is the best. We draw 10 subsamples from the top $\beta = 0.3$, and repeat Sequential-2 algorithm 30 times for each data set. The results

Table 3.2: AUC Comparison for Seq2 an Seq1 when $\beta = 0.3$, $\gamma = 0.1$; the algorithms (COF, RADA) is used

Data Set	Sequential - 2	Sequential - 1
Abalone	0.947 ± 0.022	0.976
Act	$\mathbf{0.995 \pm 0.002}$	0.727
Eeg	$\mathbf{0.974 \pm 0.010}$	0.857
Glass	0.724 ± 0.035	0.920
Lympho	$\mathbf{0.892 \pm 0.063}$	0.851
Iono	0.830 ± 0.028	0.969
Nba	$\mathbf{0.934 \pm 0.017}$	0.912
Sat	$\mathbf{0.645 \pm 0.089}$	0.336
Pec	0.986 ± 0.002	0.994
Pop	$\mathbf{0.814 \pm 0.049}$	0.633
Wine	0.987 ± 0.011	0.977
Wisconsin	0.978 ± 0.004	0.977

are summarized in Table 3.2. The bold numbers show that if the performance of Sequential-2 algorithm is at least $\mu + \sigma$ improvement from Sequential-1 algorithm. We notice that for almost half of the data sets that the performance of Sequential-1 algorithm is similar (or better) to Sequential-2 algorithm. However, on Sat data set where Sequential-1 cannot improve the base method much but using multiple samples, *i.e.*, Sequential-2 approach, increases the AUC from the best base performance 0.399 (shown in Table 3.1) to 0.645. We consider the anomaly detection problem on such data sets as the difficult problems, and using multiple subsampling can help improve the sequential ensemble more. Though using Sequential - 2 algorithm is more time consuming (multiple subsamples), the execution on each subsample anomaly detection can be easily adopt to a parallel computation environment.

3.3.6 Evaluation for Multi-layer Sequential Method

In this section, we evaluate the performance of a multi-layer sequential algorithm; see Figure 3.2. We consider two algorithms at first layer, in particular, LOF and COF, followed by RADA. The outputs of LOF and COF on the entire dataset D, are ensembled as follows:

- Denote the top $\beta \times 100\%$ of the two algorithms' outputs, based on the scores, as S_1 and S_2.
- Next, we take the union $(S_1 \cup S_2)$ and remove the duplicates.
- The resulted set is used for anomaly detection by algorithm RADA; we do not use sub-sampling.

The AUC of each dataset is summarized for both methods in Table 3.3.

Table 3.3: Multi-layer sequential compared with sequential-1 method, $\beta = 0.3$ and $k = 5$

DataSet	Multi-seq	Seq - 1 (L - R)	Seq - 1 (C - R)
Abalone	0.983	0.975	0.976
Act	0.727	0.754	0.727
Eeg	0.922	0.971	0.857
Glass	0.902	0.911	0.920
Iono	0.972	0.972	0.969
Lympho	0.826	0.685	0.851
Nba	0.832	0.788	0.912
Pec	0.998	0.995	0.994
Pop	0.496	0.399	0.633
Sat	0.445	0.487	0.336
Wine	0.847	0.731	0.977
Wisconsin	0.960	0.965	0.977
AVG	0.826	0.803	0.844
STD	0.184	0.201	0.195

We notice that for 6 out of 12 dataset Seq-1 (C-R) performs the best, for 4 out of 12 datasets Seq-1(L-R) performs the best and in three out of 12 Multi-layer has the best performance (for dataset Iono Multi-seq and Seq-1 (L-R) have the same AUC value. In brief, it appears that multi-sequential approach has limited advantage over the Seq-1 approach.

Table 3.4: Average correlation between pair of algorithms over all datasets

	LOF	INFLO	COF	RBDA	RADA
LOF	1.000	0.910	0.622	0.666	0.559
INFLO	0.910	1.000	0.566	0.636	0.536
COF	0.622	0.566	1.000	0.482	**0.390**
RBDA	0.666	0.636	0.482	1.000	0.874
RADA	0.559	0.536	0.390	0.874	1.000

3.3.7 Selection of Pair of Algorithms for Sequential Application

It has been argued that diversity of algorithms is very important [9, 74] when combining different anomaly detection algorithms. Since correlation coefficient provides a measure of similarity, we employ it towards this goal. We calculate the Pearson correlation coefficient between each pair of algorithms using the associated score vectors. Table 3.4 summarizes the average correlation between every pair of algorithms over nine different datasets. Although all correlation coefficients are large, the smallest average value corresponds to COF and RADA. In other words, we expect that, in general, COF followed by RADA (or in reverse order) should give good performance. Largest average correlations are observed between three algorithms of rank-family, *i.e.*, between RBDA, RADA, and ODMR; implying that selecting two algorithms among these will not perform as well as if one is chosen out of these three and the other from the distance based algorithm.

We note that the pair (COF, RADA) has the least average correlation and therefore compare the performance of sequential application of these two algorithms with sequential application of the pair (RBDA, RADA). Table 3.5 summarizes the results, bold items represent superior performance.

We observe that, using AUC metric, pair (COF, RADA) performs better than the pair (RBDA, RADA) for five out of nine datasets, worse for two, and equal for two.

Table 3.5: Performance (AUC score) over different pair of algorithms

DataSet	(COF, RADA)	(RBDA, RADA)
PEC	**0.996**	0.988
Wisconsin	**0.993**	0.989
KDD	**0.991**	**0.991**
Iris(1)	**1.000**	0.816
Iris(2)	**1.000**	**1.000**
Iono(1)	**0.973**	0.805
Iono(2)	**0.989**	0.979
Eeg	0.983	**0.987**
Segment	0.980	**0.985**

3.4 Conclusion

In this chapter, we proposed new sequential ensemble algorithms where the second algorithm builds upon the findings of the first algorithm. We consider single-layer sequential methods based on the sieve method which takes the output from one base algorithm to filter out the non-anomalous observations, then compare the suspect anomalies with a second algorithm. Our empirical study suggests that either LOF or COF followed by RADA achieve very good performance in most cases. It has been argued that an anomaly detection algorithm performs better on a subsample of the dataset. We incorporate the sampling concept in this proposed sequential methods. In our multi-layer sequential method, we propose to use both LOF and COF at the first layer, then combine the outputs from both algorithms as the inputs for next layer algorithm, namely RADA, results in similar performance but lower variance. To select two (or more) algorithms to apply in sequential methods, we propose to use the algorithms which have higher diversity among themselves as the base algorithms.

CHAPTER 4
ADAPTIVE SAMPLING AND LEARNING FOR ANOMALY DETECTION

The popular ensemble methods for generating more accurate classifiers are bagging and boosting. In Chapter 2, we have evaluated different bagging approaches for anomaly detection. In this chapter, we discuss the application of boosting ensemble techniques for unsupervised anomaly detection.

4.1 Boosting Approaches

A well-known ensemble boosting approach is the 'AdaBoost' supervised classification algorithm [29], which trains T rounds of weak learners over the training set, in each round focusing on the 'hard' examples from the previous round, then combines their outputs by weighted majority voting. AdaBoost calls a 'booster' in each round to draw a subsample D_t from D with a set of sampling probabilities, initially uniform. A weak learner or base method h_t is trained over D_t, then the probabilities for inclusion in the next sample are increased for incorrectly classified examples. After T rounds, each $weak\ learner\ h_t$ is assigned with a weight α_t which is lower if error is higher. The final decision output is made

by applying weighted majority voting over all h_t for $t = 1, 2, ..., T$. The training error will be substantially reduced if all the weak learners perform better than random guessing. AdaBoost has gained considerable popularity for its elegance and performance.

Algorithm 6 AdaBoost Algorithm

Input: Labeled training dataset $D = \{(x_1, y_1), ..., (x_N, y_N)\}$, and detection algorithm A.

Initialize the weight vector: $w_{x_i}^0 = 1/|D|$ for each point x_i;

for $t = 1, 2, ..., T$ **do**

1. Set:

$$\mathbf{p^t} = \mathbf{w^t} / \sum_i w_{x_i}^t$$

2. Draw N observations from D using $\mathbf{p^t}$, remove the duplicates, denote this set of observations as D_t

3. Execute algorithm A on D_t, compute $h_t(x_i) \in [0, 1]$ for each object x_i

4. Calculate the error at h_t,

$$\epsilon_t = \Sigma_{i=1}^N p_i^t \cdot |h_t(x_i) - y_i|$$

5. Calculate weight of h_t to be

$$\beta_t = \frac{\epsilon}{1 - \epsilon}$$

5. Set the new weights vector to be:

$$w_{x_i}^{t+1} = w_i^t \cdot \beta_t^{1-|h_t(x_i)-y_i|}$$

end

Output: Make the final decision

$$H(x_i) = \begin{cases} 1, \text{ if } \Sigma_{t=1}^T (\log \beta_t)^{-1} h_t(x_i) \geq \frac{1}{2} \Sigma_{t=1}^T (\log \beta_t)^{-1} \\ 0, \text{ otherwise} \end{cases}$$

For clustering problems, an Adaptive Clustering algorithm was proposed in [67], with an adaptive sampling ensemble method for better partition generation. They use a consis-

tency index to determine the sampling probability and adaptively focus more on points in regions with inconsistent cluster labels. A consensus function is used for the final decision output, using an agglomerative algorithm applied on the co-association similarity matrix. Empirical study has shown improved performance compared to the classical k-means clustering algorithm.

The application of ensemble methods for unsupervised outlier detection is an emerging research topic, addressed in only a few works so far, such as [6, 9, 39, 44, 74]; of these the work in [6] is the only attempt to apply AdaBoost to solve the outlier detection problem. Their proposed method converts the outlier detection problem to an AdaBoost-solvable classification problem by drawing some number of points from an underlying distribution, and marking them as outliers, whereas all the points in the original data set are considered to be inliers. Their resampling method is based on *minimum margin active learning* by iteratively assigning lower sample rejection probabilities to points with low margin values, because they have less consistency. The weights for each classifier are adjusted by the classification error rate in each subsample.

By contrast to the above approaches, we propose a novel adaptive learning algorithm for unsupervised outlier detection which uses the score output of the base algorithm to determine the 'hard' examples, and iteratively resamples more points from such examples in a completely unsupervised context. The method is evaluated on multiple well-known datasets, and our simulations show better results than the base algorithm as well as other existing outlier detection approaches.

The next section describes the adaptive sampling approach. This is followed by discussing the combination method and the proposed algorithm. Experimental simulations and results on benchmark problems are then presented.

4.2 Adaptive Sampling

The main idea of adaptive sampling is to give greater emphasis to the examples that are hard to identify. To extract such examples from the data set, we need to answer two questions:

1. How can we determine whether an object cannot be easily classified as an inlier or an outlier?

2. How can we combine the outputs from different iterations?

In this section, we focus on the first question. We first review how classification and clustering ensemble methods can be used to determine the hard-to-identify objects and adjust their sampling weights. We then discuss our proposed adaptive sampling method for outlier detection.

The popular ensemble classification method AdaBoost is adaptive with respect to the training errors of various weak hypotheses, and successively (in each iteration) increases sampling probability for points with a high error rate.

In the adaptive clustering ensemble approach, clustering consistency is used to estimate the difficulty of assigning a clustering label to an object. Objects near cluster boundaries are assigned higher sampling probabilities, and the boundaries are successively refined in later iterations.

In AdaBoost, the classification decision boundaries are refined gradually by resampling the objects that are more likely to be near the decision boundaries. Adaptive clustering is also focused on refining cluster boundaries. The outlier detection problem is more difficult: unlike classification and clustering, there is no clear boundary in outlier detection. Instead, most outlier detection algorithms output a score for each object indicating the probability of that object being an outlier; we now discuss our proposed method to use outlier scores to determine the 'decision boundary' in outlier detection adaptive learning.

4.2.1 Justification with local density-based kNN algorithms

Local density based methods consider the outlier score of a point o to be proportional to the density of its k nearest neighbors over the local density of o. The outlier score $Score(o)$ of an object o can be rewritten as follows:

$$Score(o) \propto \frac{DEN_k(N_k(o))}{den_k(o)},$$

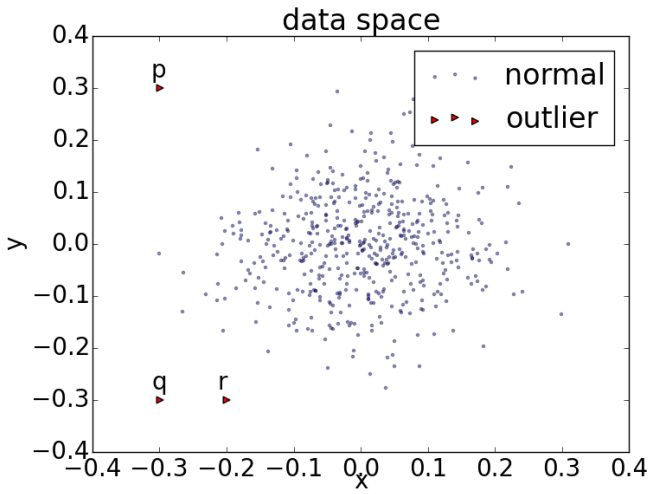
where $N_k(o)$ is the set of k nearest neighbors of o, $den_k(o)$ is the local density of o, and $DEN_k(N_k(o))$ is the averaged local density of o's k nearest neighbors. For an inlier i, $Score(i) \approx 1$; for an outlier o, $Score(o) \gg 1$.

Many algorithms have been proposed to solve the outlier detection problem in an unsupervised context, but all have high false positive rates. An algorithm misidentifies an inlier p to be an outlier when p's neighborhood is dense; then $Score(p)$ is high due to other inliers in this neighborhood.

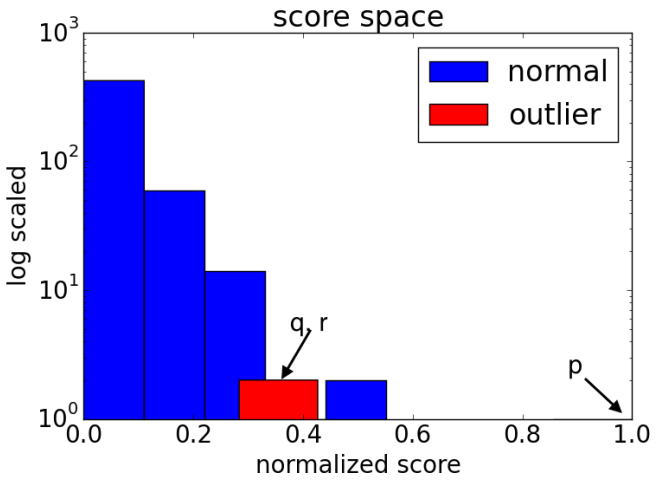
Likewise, if an outlier q is in a sparse neighborhood, other outliers in this neighborhood force the score of q to be small; and therefore q is declared to be an inlier.

Our sampling approach aims to sample more from the 'boundary' area, in order to resolve such problems.

In our approach, we remove the obvious inliers and obvious outliers from the dataset. The benefits from such sampling are: (1) removal of the outliers from $N_k(q)$ will increase $den_k(N_k(q))$; (2) removal of the inliers from $N_k(p)$ will decrease $den_k(N_k(p))$. Hence, after such subsampling, the 'boundary' points can be separated better, while the obvious outliers remain identifiable.



(a) Data Space



(b) Score histogram in log-scale

Fig. 4.1: 2D example with 3 outliers

4.2.2 Decision Boundary Points

Many outlier detection algorithms assign a score for each object, roughly indicating the relative probability of the object being an outlier (or inlier). Objects with higher scores are considered more likely to be outliers.

Most existing outlier detection algorithms perform well on capturing the obvious outliers, but may fail to capture the hard-to-detect outliers. We use the state-of-the-art local density based anomaly detection algorithm LOF [18] in the following as our base detection algorithm.

A simple example is illustrated in Figure 4.1a, where non-anomalous (inlier) data points occur within a two-dimensional Gaussian cluster, outside which three outliers have been inserted: points p, q, and r. We apply the LOF anomaly detection algorithm with $k = 3$, obtain outlier scores for each object, and normalize the scores to be in [0,1]. The histogram for normalized scores in log-scale is shown in Figure 4.1b.

We observe that the object on the left top corner (p) is the most obvious outlier among all of the three outliers, and was assigned the highest outlier score as shown Figure 4.1b; this illustrates that the detection algorithm performs very well when detecting the easy-to-detect outliers.

Next, we address how to use the score space to distinguish potential outliers and inliers. We use the obvious and often used method: choose a threshold θ such that all objects with an outlier score greater than θ will be identified as outliers and all other objects are marked as inliers; θ determines the decision boundary. In the example of Figure 4.1b, if we choose $\theta = 0.3$, then two real outliers p,q are identified, but five false positives are introduced, and we have one false negative r (undetected outlier). Points near the threshold boundary are hard to identify, *i.e.*, whether they are true outliers.

Our approach is to re-evaluate such points that are near a decision boundary. In the following, we describe our method of how to adjust the sampling weights.

4.2.3 Sampling Weights Adjustment

As stated above, we consider normalized scores for each object x such that $Score(x) \in [0, 1]$; $Score(x) \approx 0$ identifies a clear inlier, $Score(x) \approx 1$ identifies a definite outlier, and $Score(x) \approx \theta$, implies that x is more likely to be a boundary point, where $\theta \in [0, 1]$ is an appropriately chosen threshold.

Most false positives and false negatives are expected to occur at boundary points, with existing algorithms. In our approach, a new weight is assigned to each object o, roughly measuring the expectation that it is a boundary point, as follows:

$$WB(x) = \begin{cases} \frac{Score(x)}{\theta} & \text{, if } Score(x) < \theta, \\ \frac{1-Score(x)}{1-\theta} & \text{, if } Score(x) \geq \theta. \end{cases} \tag{4.1}$$

This weight assignment increases sampling probability for boundary points and decreases the sampling probability for points that are easy to detect (clear inliers and outliers).

Instead of iteratively refining the boundaries as in classification or clustering problems, we iteratively re-evaluate the points near the boundaries in outlier detection, in order to reduce the number of false positives.

4.3 Final Outputs Combination with Different Weighting Schemes

It is very important in ensemble methods to determine how to combine different outputs from all the weak learners (individual modules). A commonly used method is weighted majority voting in which the weight of a learner is assigned by measuring the quality of its results. At iteration t, where $t = \{1, 2, ..., T\}$, for object x, if a learner's output is $h_t(x)$,

the final output is denoted by:

$$H(x) = \sum_{t=1}^{T} \beta_t h_t(x)$$

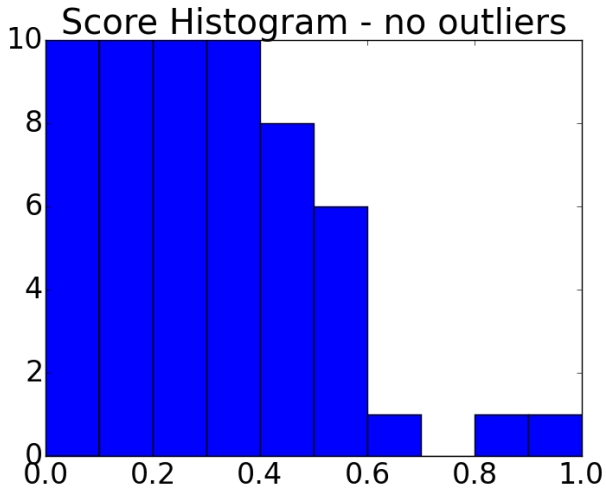
where β_t indicates how important the learner h_t is in the final combination. In classification, one can use the trained error as an indication of how well a learner performed. The absence of labeled training data makes it difficult to measure the goodness of a clustering or outlier detection algorithm in an ensemble approach. If the outputs from all the iterations are independent, we can use the correlation between them as an indication of how well one individual learner performs. However, in the adaptive learning process, the input of each learner is dependent on the previous one, and this makes the process of selection of weights (for each learner) even harder. We now propose heuristics for how to assign weights β_t to the different learners and evaluate them empirically.

The problem of measuring how well a learner performs in an iteration is essentially the question of how many 'real' outliers were captured in that iteration. Obviously, the objects with higher scores are more anomalous than the ones with lower scores. But a more pertinent question is that of determining the size of the gap between scores of anomalous vs. non- anomalous objects.

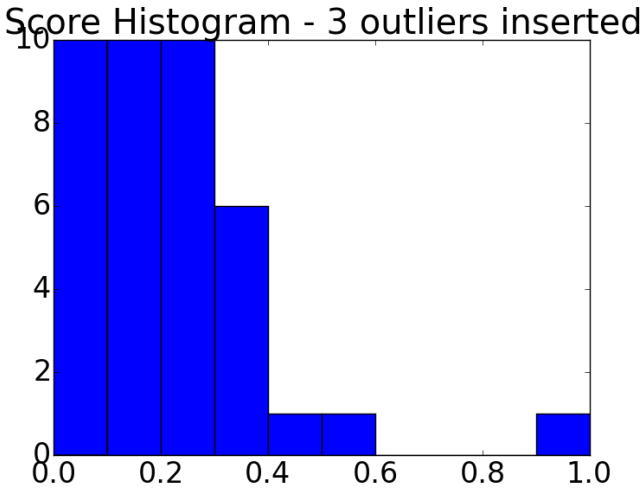
In Figure 4.2, we show the zoomed histograms of normalized scores for two datasets: (1) with no outlier and (2) for the same dataset with 3 outliers inserted. We observe that the outliers get larger scores, and also the gap in scores between the inliers and outliers increases. In Figure 4.2a, the 'gap' is from 0.7 to 0.8 while in Figure 4.2b, the 'gap' is from 0.6 to 0.9. Using these concepts, we have evaluated three alternative weight assignments for β_t:

a) The simplest approach is to take the arithmetic average for all values of t. Thus, we assign:

$$\beta_t = 1/T; \text{for each } t \in \{1, 2, ..., T\}$$



(a) Dataset without Outliers



(b) Dataset with 3 Outliers

Fig. 4.2: Zoomed Score Histogram Example

b) Select a score threshold θ_s, and at each iteration t, calculate a_t= the number of objects with score greater than θ_s. Using this, we assign:

$$\beta_t = 1 - \frac{a_t}{|D_t|}$$

where $|D_t|$ is the size of the sample in the t^{th} iteration .

c) At each iteration t, obtain the histogram of the score output, calculate the 'gap' b_t

between the right-most bin and the second right-most bin, and set:

$$\beta_t = \frac{b_t + 1}{\sum_{t=1}^{T} b_t + 1}.$$

4.4 Adaptive Sampling Algorithm

The algorithm begins by giving equal sampling weight to all points in the original dataset D, such that for every point x_i, $w^0_{x_i}$=$1/|D|$. At each iteration, we draw N observations following the sampling distribution $\mathbf{p^t}$. In the set of observations, duplicates are removed and the scores are re-evaluated; the resulting set of observations is denoted as D_t. We adjust the sampling weights for all the points in D_t as mentioned above, and normalize their sampling weights, dividing by the sum of all the sampling weights in D_t; for unsampled data, the sampling weights remain unchanged. This process continues for $t = 1, 2, ..., T$.

The result of our sampling makes the sampling weights for possible boundary points (hard-to-identify points) higher in the following iterations, so the 'effective' sample size will decrease over iterations. To prevent the sample size from reducing too fast, we select $N = 2|D|$ in our experiments. Details of the algorithm (Algorithm 7) are shown below.

Algorithm 7 Adaptive Sampling Algorithm

Input: Dataset D, detection algorithm A.

Initialize the weight vector: $w_{x_i}^0 = 1/|D|$ for each point x_i;

for iteration t = 1,2,...,T:

1. Set:

$$\mathbf{p^t} = \mathbf{w^t} / \sum_i w_{x_i}^t$$

2. Draw N observations from D using $\mathbf{p^t}$, remove the duplicates, denote this set of observations as D_t

3. Run A on D_t, get a score vector $\mathbf{Score^t}$, normalize the scores to [0,1]

4. $h_t(x_i)$= normalized score of x_i

5. Set the new weights vector to be:

$$w_{x_i}^{t+1} = (1-\alpha) * w_{x_i}^t + (\alpha) * WB(x_i);$$

where

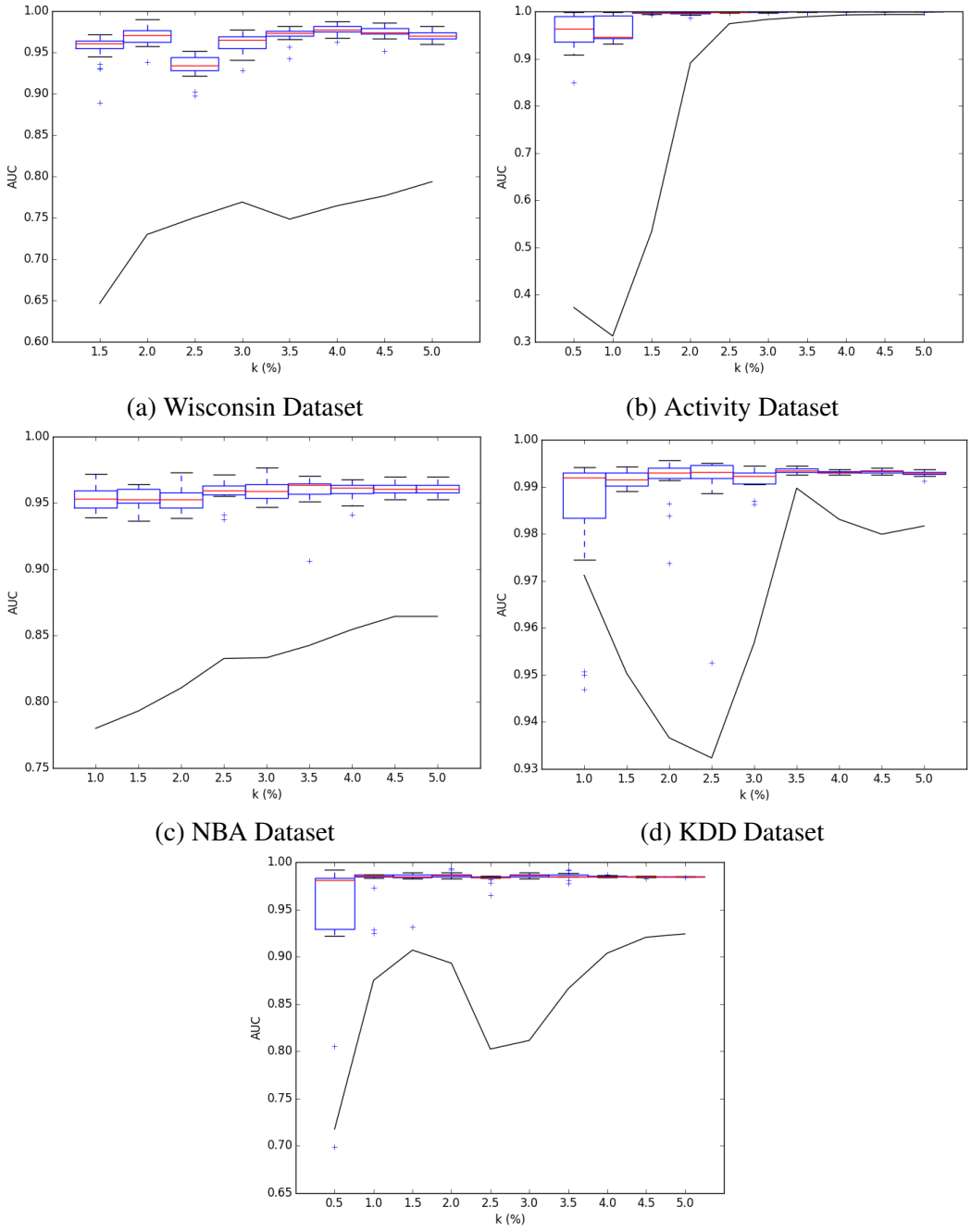
$$WB(x_i) = \begin{cases} \frac{Score(x_i)}{\theta} & \text{, if } Score(x_i) < \theta, \\ \frac{1-Score(x_i)}{1-\theta} & \text{, if } Score(x_i) \geq \theta. \end{cases}$$

Output: Make the final decision

$$H(x_i) = \sum_{t=1}^{T} \beta_t h_t(x_i)$$

4.5 Experiments and Results

This section describes the datasets used for simulation, simulation results, and discussions of model parameters. In our results, the AUC defined in Section is used for evaluation.



(a) Wisconsin Dataset

(b) Activity Dataset

(c) NBA Dataset

(d) KDD Dataset

(e) PEC Dataset

Fig. 4.3: AUC Performance Comparison with Base Algorithm (LOF) over Different k

4.5.1 Dataset Description

Five well-known datasets were used in our simulations, which are defined in Section 1.5.1.

4.5.2 Performance Comparisons

We evaluated our ensemble approach with base algorithm LOF for different k values (the number of local neighbors). In Figure 4.3, we plot AUC for various k values, where the x-axis shows the ratio of k to the size of the dataset. A solid line shows the AUC values for base algorithm LOF, and a box plot shows the results of our ensemble approach using $\beta_t = 1 - \frac{a_t}{|D_t|}$, using 25 iterations in this experiment. Results show that for all k values, our approach outperforms the base algorithm for all the 5 datasets. The ensemble approach has large variations when k is small, which decreases as k increases.

We also compared our proposed ensemble approach with the Active-Outlier approach proposed in [6], Feature Bagging approach proposed in [44], and HiCS (high contrast subspace) outlier detection method proposed in [39]. In Active-Outlier approach, [6] propose to use AdaBoost with decision tree classifier (CART) as the base classifier, and the synthetic outliers are generated from a specific distribution (Uniform, Gaussian). In Feature Bagging, we use the same number of ensemble members and use LOF as base method. In HiCS, we use LOF as base method. Feature Bagging and HiCS approaches are implemented in ELKI toolbox [7], results for three different k values are reported. In our ensemble approach, we do not add any additional observations; we use LOF as the base outlier detection algorithm with $\alpha = 0.2$, $\tau = 0.95$ and results are reported for three different values of k. For fair comparison, we use the same number of iterations in all methods.

Table 4.1 presents the averaged AUC for performance over 20 repeats. Active-Outlier approach is denoted as *ActOut*, Feature Bagging is denoted as *FB*, while our proposed ensemble approach is denoted as *Adaptive*. As can be seen that the performance of our proposed ensemble approach is better in almost all cases and increases as k increases.

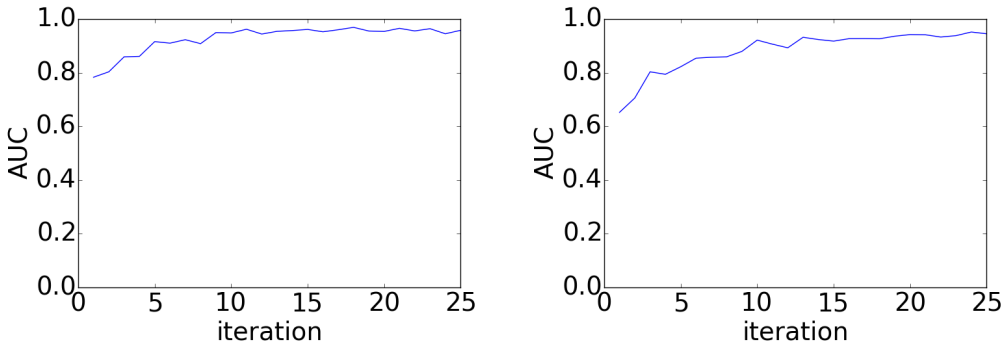
Table 4.1: Performance Comparison: Averaged AUC over 20 Repeats

Data	ActOut		k=1%			k=3%			k=5%		
	Unif	Gauss	FB	HiCS	Adaptive	FB	HiCS	Adaptive	FB	HiCS	Adaptive
WIS	0.865	0.921	0.446	0.564	**0.963**	0.407	0.539	**0.965**	0.448	0.761	**0.978**
NBA	0.903	0.810	0.822	0.709	**0.920**	0.855	0.864	**0.959**	0.884	0.885	**0.961**
ACT	0.5	**0.980**	0.833	0.922	**0.961**	0.977	**1.000**	0.999	0.993	**1.000**	**1.000**
PEC	0.907	0.686	0.762	0.768	**0.975**	0.775	0.808	**0.986**	0.701	0.894	**0.985**
KDD	0.873	0.685	0.844	0.778	**0.985**	0.780	0.836	**0.993**	0.787	0.795	**0.993**

4.5.3 Effect of Model Parameters

Effect of Number of Iterations:

In Fig. 4.4, we plot the AUC value versus the number of iterations for a synthetic dataset and a real dataset. As we can see, on the synthetic dataset, performance stabilized after 10 iterations and on the real dataset, performance stabilized after 20 iterations.



(a) Synthetic Dataset (b) Real Dataset

Fig. 4.4: AUC Performance vs. Number of Iterations

Effect of Combination Approach:

The simulations for comparing different possible combination approaches were set up as follows:

1. For each dataset, for each k value, apply the algorithm with each of the three different combination approaches.

2. Repeat the whole process 20 times, report the mean of AUC.

3. For each dataset, for each k value, rank the AUC performance of each combination approach, so the best one will have a rank of 1, the worst one has a rank of 3.

4. For each dataset, calculate the sum of the above ranks for each combination approach, so the best one has the lowest sum.

In Table 4.2, we summarize the accumulated sums of AUC rankings over all different k values for the three approaches and denoted as Sum_a,Sum_b, and Sum_c respectively. The one with best detection performance will have the lowest Sum over the three different combination approaches.

Table 4.2: Performance over Sum of AUC Rankings with Different Combination Approaches

DataSet	Sum_a	Sum_b	Sum_c
WIS	42	**33**	73
NBA	55	52	**43**
ACT	41	**27**	73
PEC	59	45	**33**
KDD	**33**	41	57

We observe that combination approach a ranks once as the best, twice as the second, and twice as the third; combination approach c ranks twice as the best, three times as the third; while the combination approach b ranks twice as the best, three times as the second and none as the third. So we can say that combination approach b, where $\beta_t = 1 - \frac{a_t}{|D_t|}$ outperforms the other two combination approaches over the 5 datasets we considered.

4.6 Conclusion

In this chapter, we have proposed a novel adaptive sampling approach for unsupervised outlier detection. We use the normalized score output to select a decision boundary where the hard-to-detect anomalies are contained, design a sampling weight assignment function which re-draw and re-evaluate the points in such boundaries iteratively. We then design three combination approaches for combining the score output from each iteration, results show that by assigning more weights on iterations output more skewed scores that best performance are achieved. Simulation results on five well-known data sets show the relative success obtained with the new approach, compared to the LOF base algorithm as well as other recent approaches.

CHAPTER 5
ENSEMBLE METHODS FOR ANOMALY DETECTION ON STREAMING DATA

In the previous chapters, we discussed how to use ensembles of various algorithms for detecting anomalies in static datasets. In this chapter, we focus on how to use random forests based methods to improve the anomaly detection rate for streaming datasets.

The key concept in a current work [64] is to build a random forest where in each tree, at any internal node, a feature is randomly selected and the associated data space is partitioned in half. To improve the efficiency of a forest, we make the following contribution in this chapter:

- We give mathematical justification of required tree height and number of trees by casting the problem as a classical coupon collector problem in Section 5.2.2.
- We design a majority voting score combination strategy in Section 5.2.3.
- We apply feature clustering to group the correlated features together in order to find the anomalies jointly determined by subsets of features in Section 5.2.4.
- To better partition the data space for anomaly detection, we adopt an Evolutionary algorithm to maximize the chance of separating anomalies in Section 5.3.

5.1 Anomaly Detection for Streaming Data

Consider a data stream arriving at time stamps $t_1, t_2, ..., t_n, ...$, where each data $\mathbf{X_t}$ is a high-dimensional data point containing d features and a label; *i.e.* $\mathbf{X_t} = (x_t^1, x_t^2, ..., x_t^d, label)$, where $label \in \{normal, abnormal\}$ but is unknown. The problem of finding anomalies from streaming data is to separate the data points with $label = abnormal$ from the majority data points with $label = normal$. We use F to denote the set of features, namely, $F = \{f_i | i = 1, ..., d\}$. By definition, abnormal points are fewer than the normal points. As discussed in the previous chapters, in unsupervised anomaly detection, labels are unknown, and the task is to find a set of rules to separate anomalous observations from normal observations.

A recent survey for anomaly detection on temporal data can be found in [32]. In [8], Aggarwal proposes a statistical profiling method for detecting the deviations for new data observations from the expected values. This method builds a regression model for the historical data, and compares the observed values with expected values for anomaly detection. An illustration can be found in Figure 5.1, in which $\hat{y}_{t+1}$ is the predicted values for datastream at time $t + 1$, and y_{t+1} is the observed value.

Previous work in Half-Space Trees [64] adopts an ensemble method which combines

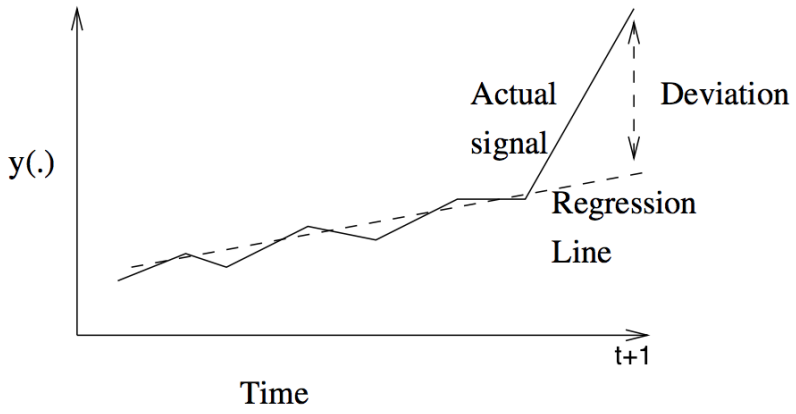


Fig. 5.1: An illustration of detecting the deviations in a data stream [8]

results from a set of T full binary trees. To build a full binary tree, a feature from F is randomly selected at each node and the existing space is partitioned in two equal parts.. Note that each feature can be selected multiple times. An illustration can be found in Fig. 5.2. In the figure, the data space is shown on the left, while a full binary tree is built on the right. The tree has 7 nodes, each tree path forms a partition. For example, tree leaf node IV is associated with a region in the dataspace withfeature value $Y < 0.25$, and there are 2 such points; tree leaf node VI is associated with a region with $X < 0.5$ and $Y < 0.5$, and there are 33 such points. The idea is that a point in a (leaf) node's region with more observations should be less anomalous than a point in the region of a leaf with fewer observations.

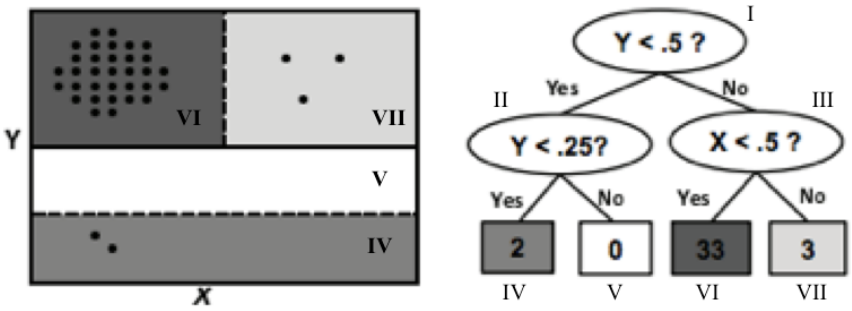


Fig. 5.2: An illustration of dataspace partition by one HSTree [64]

The basic idea of this method is to partition the data space into random half-space and the points belonging to sparse partitions are possible anomalies. Tan *et al.* [64] proposed to use mass profile associated with each tree node to record the number of points falling in that partition. A mass profile for each *node* contains two parts: *node.latest* is used for recording the newly arrived information during the testing phase and *node.reference* is used for for anomaly detection obtained from training data. Batch learning is used for streaming anomaly detection, *node.reference* is used for recording the information for the last batch, therefore, each newly arrived observation is compared with this information while this observation is updated in the latest profile. This method has three phases: tree building, training and testing, as described below:

- **Tree building:**

  in this phase, T full binary trees with height of h are built without any data;

- **Training:**

  for each tree:

  - the first ϕ data are taken to construct the *reference* mass profile for each node;
  - set the *latest* mass profile to 0 for each node;

  these steps are repeated for the T trees;

- **Testing:**

  for each observation x, for the i^{th} tree, $i = 1, ..., T$:

  - x is fed into i^{th} tree, along the path followed by x, update each node's *latest* mass profile, *node.latest*, by increasing it by 1; when it arrives at a leaf *node*, calculate the anomaly score for x as:

$$score_i(x) = node.reference * 2^{node.height}. \tag{5.1}$$

  The final anomaly score for x is an average over the T trees:

$$Score(x) = \sum_{i=1}^{T} score_i(x).$$

  For each set of ϕ data points, for each *node*, swap their $node.reference$ and $node.latest$, set $node.latest = 0$; when a new observation arrives, continue the testing procedure.

This method has shown its advantage in detecting anomalies compared to other state-of-the-art detection algorithms, and is competitive with supervised learning methods. However, the authors confine the evaluations to low-dimensional data streams [64]; and this method has the following deficiencies:

- the tree height and the number of trees are selected without any mathematical justification;
- final score combination is simple averaging although the detection power of each tree is different;
- if the number of noisy features is large, more tree nodes are needed for detecting anomalies; and
- in our view, some anomalies can only be jointly determined by a subset of features, consequently, building random trees over entire feature space might be wasteful.

Also, a different systematic approach to build trees will be beneficial over random trees generation.

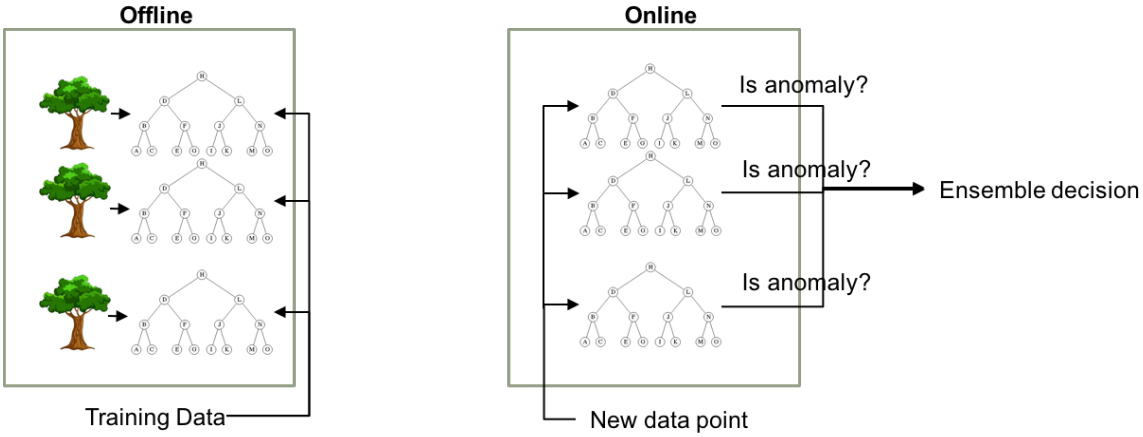


Fig. 5.3: A framework for streaming anomaly detection

5.2 Analysis of Random Trees

In this section, we first analyse how random trees perform when number of features increases. Then, we derive the expected number of trees and height of trees using the theories from coupon collector problem. Next, we design two score combination methods for the final decision making process which alleviate a problem of using score averaging. Finally,

we propose to apply feature clustering to group the correlated features together, then build random trees with respect to the features within each feature cluster, and our evaluations on both synthetic and real-world datasets show improvement over building random trees on the entire feature set.

5.2.1 Performance of Random Trees and Number of Features

We believe that the performance of random forest will decrease if the number of features increase (if the number of trees and their heights remain fixed). We construct two synthetic datasets to illustrate this problem. We choose two datasets which are used in [64]. **Http dataset** contains 567,497 data points, 3 features and 0.4% of them are anomalies, **CoverType dataset** contains 286,048 data points, 10 features and 0.9% of them are anomalies, as described in Chapter 1.5.2. To illustrate performance deterioration, we continuously add noisy features which are drawn from a uniform distribution to the two data sets. With noisy features added to **Http dataset** and **CoverType dataset**, we denote them as *syn-1* and *syn-2*, respectively. In the following experiments, we use the AUC (defined in Section 4.5) to measure the performance of anomaly detection algorithms.

We designed 3 sets of experiments:

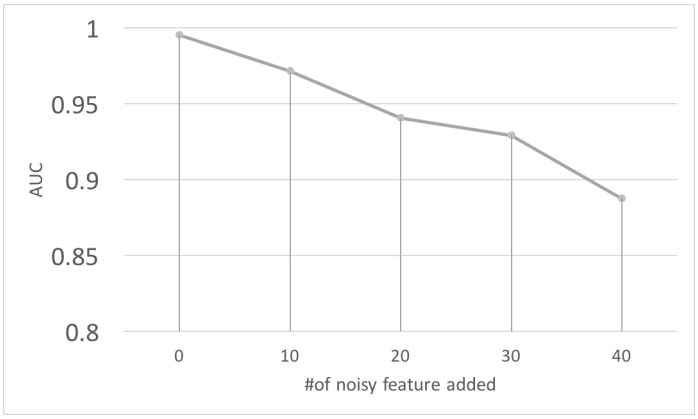
- We fix the number of trees and tree height, vary the number of noisy features and compute the AUC performance as the number of noisy features increases. The results, shown in Figure 5.4, indicate that when the number of noisy features increases, the performance decreases.

- We fix the number of noisy features to be 40, fix the tree height, and vary the number of trees, then compute the AUC. The results, shown in Figure 5.5, indicate that we need more trees to detect anomalies in the presence of noisy features.

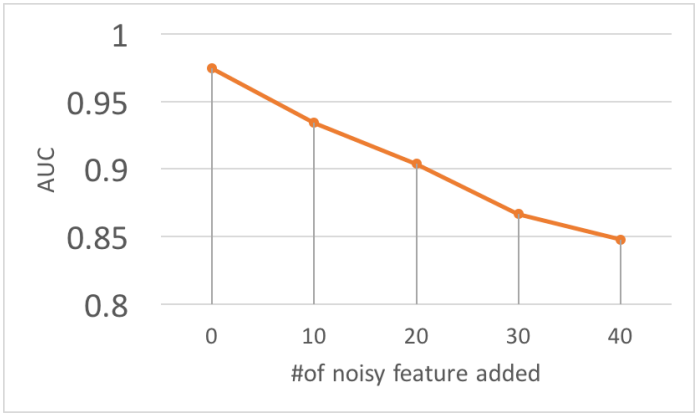
- We use the original dataset, and fix the number of trees used, vary the tree height and compute AUC. The results, shown in Figure 5.6, indicate that performance increases

when trees are ‘taller’.

From these results, we conclude that the number of trees and associated heights should be determined carefully and consistent with the number of features. We give theoretical justification of how to select these numbers in Section 5.2.2.



(a) Performance with syn-1

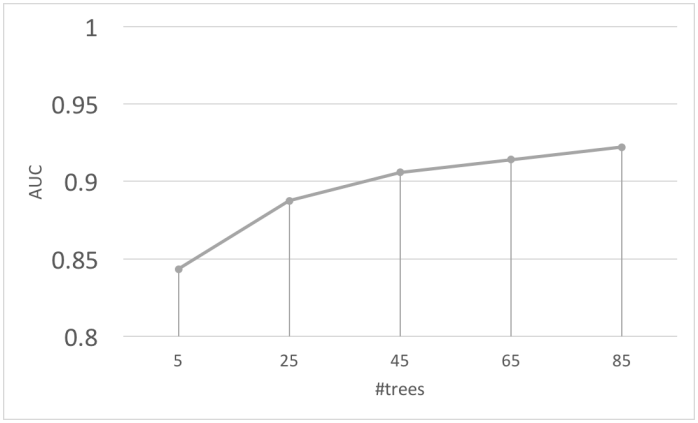


(b) Performance with syn-2

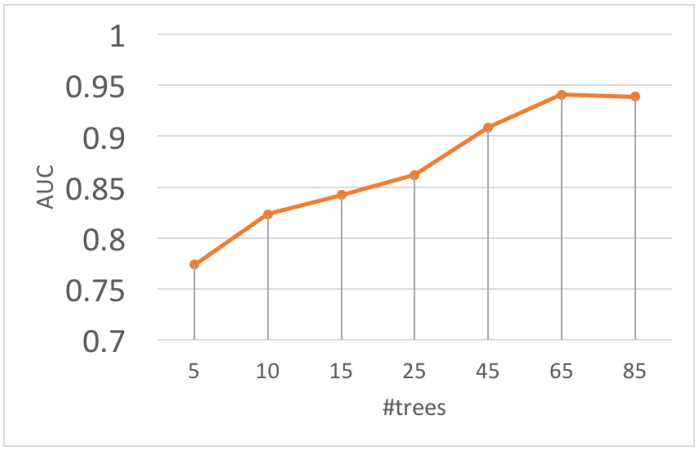
Fig. 5.4: Variation of AUC with the number of noisy features in the syn-1 and syn-2 datasets

5.2.2 Deriving the Number of Trees and Height of Trees using Theory of Coupon Collector Problem

In the previous section, we observe that the performance of random trees will be decreasing when the number of features increases. In this section, we intend to derive a relationship



(a) Performance with syn-1



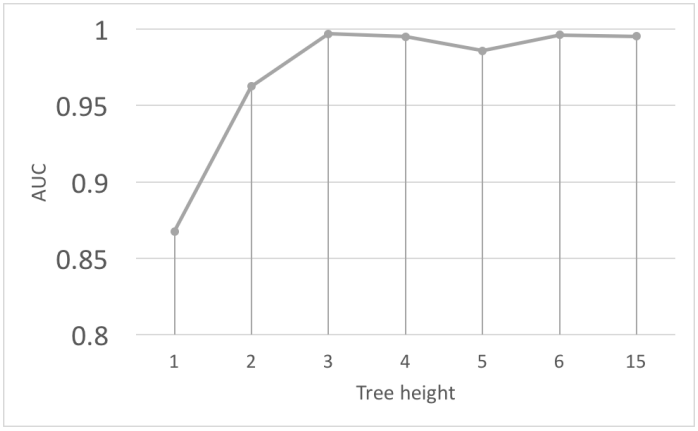
(b) Performance with syn-2

Fig. 5.5: Variation of AUC with the number of trees used in the syn-1 and syn-2 datasets

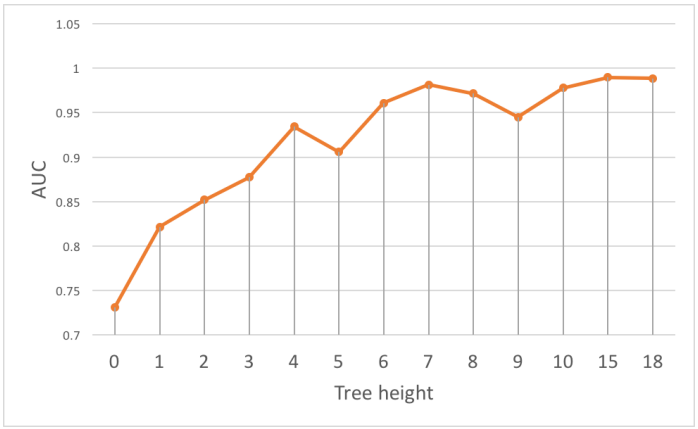
between the number of trees and height of trees required in term of number of features present in a dataset by casting the random trees to coupon collector problem.

Number of Nodes Needed with Noisy Features

Not all features are informative, only a subset I of F are meaningful whereas $F \setminus I$ are spurious. Let $m = |F|$ be the total number of features, and $k = |I|$ be the number of informative features, so $m - k = |F| - |I|$ is the number of noisy features which are not contributing to anomaly detection. In constructing a random forest, at each node, a feature is randomly selected from m features. The probability that an informative feature will be



(a) Performance with syn-1



(b) Performance with syn-2

Fig. 5.6: Variation of AUC with the height of the trees used in the syn-1 and syn-2 datasets

selected at each tree node is:

$$Pr(SelectInform) = k/m.$$

If h is the height of each tree, then the number of nodes in it is $node(trees_i) = 2^h - 1$ since $trees_i$ is a complete binary tree. Let $\mathcal{F}(trees_i)$ be the number of informative features covered by tree $trees_i$, then

$$\mathcal{F}(trees_i) \sim B(2^h - 1, k/m).$$

The expected number of informative features selected is $(2^h - 1)\frac{k}{m}$ will be small if $\frac{k}{m}$ is small. If we wish to construct an ensemble of trees such that each informative feature is presented at least in one tree, then, we need more nodes to cover more informative features. If the heights of trees are the same, then we will need more trees in our forest to cover as many informative features.

We now discuss how to find the expected number of trees and height for the trees when the number of features are fixed.

The coupon collector's problem - Analysis of tree height

In the coupon collector's problem [47], there are d types of coupons and they are drawn at random at each trial. Let r be the number of trials for one to collect at least one copy of each of the d types of coupons. The goal of the coupon collector's problem is to find out what is the relationship between r and d.

The similarity between the random trees and the coupon collector's problem is that if we treat each feature as a type of coupon, each detection path in the tree can be treated as an experiment with n trials – where n is the number of nodes in a random tree of height h. If an anomaly is jointly described by d features, each tree should capture at least one copy of each of the d features. To study the relationship between the number of nodes n and number of features present, we adopt the theoretical results for the coupon collector's problem.

We show that when the tree height is

$$h = \log_2(\beta d \ln d + 1),$$

the probability that at least one of the features is not captured is bounded by $d^{-(\beta-1)}$, where $\beta > 1$.

Let X_d be a random variable defined to be the number of nodes required to collect at

least one copy of each type of the d features. The expected number of nodes is

$$E[X_d] = d\sum_{i=1}^{d}\frac{1}{i} = dH_d$$

where H_d is the harmonic sum [47].

Let σ_i^n be the event that feature i is not selected nuin n nodes, the probability of this event is:

$$Pr[\sigma_i^n] = (1-\frac{1}{d})^n \leq e^{-\frac{n}{d}},$$

for $n = \beta d \ln d$, this bound is $d^{-\beta}$, where $\beta > 1$ is a constant.

Thus, the probability that at least one of the features is not captured in the n nodes is

$$Pr[\cup_{i=1}^{d}\sigma_i^n] \leq \sum_{i=1}^{d} Pr[\sigma_i^n] \leq \sum_{i=1}^{d} d^{-\beta} = d^{-(\beta-1)},$$

for a random tree with number of nodes $n = \beta d \ln d$, consequently, the tree height $h = \log_2(n+1) = \log_2(\beta d \ln d + 1)$.

Number of trees T for a given tree height h and number of features d

Given tree height h, each tree has $n = 2^h - 1$ nodes. For T such trees, the total number of nodes is nT. Number of trees T is chosen such that the probability that each feature occurs at least in one of the T trees should be larger than $1-\nu$. From the results from the coupon collector problem, we have:

$$Pr(X_d = k) = \sum_{j=0}^{d-1}(-1)^j \binom{d-1}{j}(1-\frac{1+j}{d})^{k-1}.$$

It is desired that:

$$1 - (1 - Pr[d \leq X_d \leq nT]) \geq 1 - \nu \tag{5.2}$$

$$Pr[d \leq X_d \leq nT] \geq 1 - \nu \tag{5.3}$$

$$\sum_{i=d}^{nT} Pr(X_d = i) \geq 1 - \nu \tag{5.4}$$

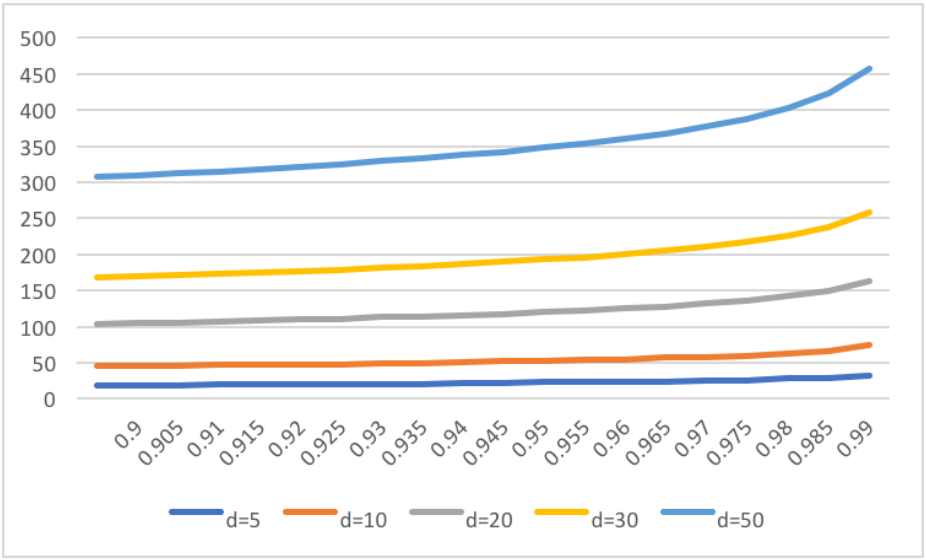
$$\sum_{i=d}^{nT} \sum_{j=0}^{d-1} (-1)^j \binom{d-1}{j} \left(1 - \frac{1+j}{d}\right)^{k-1} \geq 1 - \nu \tag{5.5}$$

This is a combinatorial problem, and numerical solutions are shown in Figure 5.7.

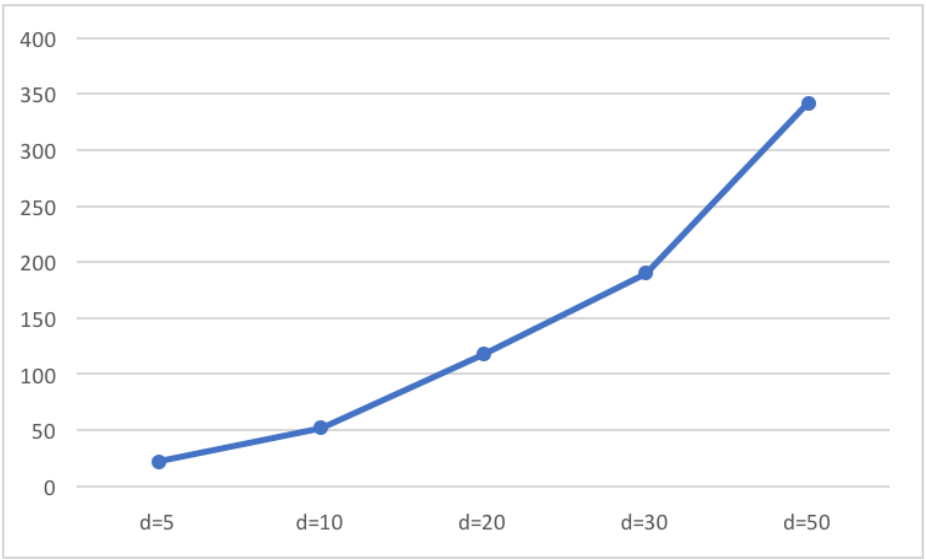
5.2.3 Discussion on Score Combination

One of the most important issues in designing an ensemble algorithm is output combination. In this section, we use the mass profile as the anomaly scores. As described before, the smaller the score is, the more anomalous the observation is. Although the method is designed to handle streaming data processing, however for the purpose of evaluation, we evaluate the score assignment and compare the performance for each observations within the same batch.

In the previous paper [64], the final score for each object is averaged among all the random trees. However, as we see later, each tree has different power of capturing anomalies, therefore, simple averaging might diluting the final result. For example, if we have three trees with height of 2, the scores for each tree can be represented as a vector of length 4. Then, consider a dataset with 100 objects, and we build three trees where the score outputs from them are: (33,33,33,**1**), (**60**,20,15,5), (25,50,20,**5**), where the bold font represents the leaf node that contains the anomaly. If we consider the simple arithmetic average, then, the leaf with a score of 60 will significantly affect the final result though two out of three trees say the point is an anomaly. This effect indicates that we need other score combination methods other than simple arithmetic averaging. In this section, we discuss two combina-



(a) Y-axis is nT, X-axis is $1 - \nu$



(b) Y-axis is nT, X-axis is number of dimensions

Fig. 5.7: Numerical results for the number of trees and tree height

tion methods, including the minimum score method and the majority voting method.

The minimum method

In Chapter 2, we observe that using a minimum ranking method for independent results combination generates the best performance. However, as we evaluated in Section 2.4.2 that when the base algorithm is the same, then, using maximum score method gives the best

performance. We use the minimum score method because in the previous algorithms, the higher the score is, the more anomalous the observation; while in random trees algorithm, lower score indicates a higher degree of anomalousness. As before, the idea of adopting a minimum approach is that an object will be detected as an anomaly even if one of the random trees say so. Such an approach would be beneficial when the system allows some false positive but requires low false negative rate. The final score for each object x over T random trees is:

$$Minimum(x) = \min_{i=1}^{t} score_i(x)$$

where $score_i(x)$ is the anomaly score for x on the i^{th} random tree as defined in Equation (5.1).

Majority voting approach using score discretization

The output for each object is a numeric score. In order to adopt the majority voting approach, we need to discretize the score before the combination.

We now discuss how to find the cutoff point for discretizing the score vector to a binary representation. If data is uniformly distributed, without any anomalies, then the number of objects at each level would be uniform. For example, if we have 100 objects and 4 leaves, the score vector would be (25,25,25,25) and no anomalies can be detected. Anomalies are expected to be associated with a leaf node with a very small number of observations, while the normal objects are grouped densely together. We use the η-quantile as an indication for anomalies, denote as q_η. For example, if we use 5% quantile as a cutoff point, then the vector (50,44,2,4) will be converted to (0,0,1,1). The converted score for each object x is calculated below:

$$b(x) = \begin{cases} 1, & \text{if } score(x) < q_\eta; \\ 0, & \text{otherwise.} \end{cases}$$

The result obtained from majority voting of each object x is denoted by $Majority(x)$, and

can be calculated in below:

$$Majority(x) = \begin{cases} 1, & \text{if } \sum_{i=1}^{T} b_i(x) > T/2 \\ 0, & \text{otherwise.} \end{cases}$$

Where $b_i(x)$ is the converted binary output for x in the i^{th} tree.

Experimental Results

We tested three different combination approaches on a synthetic dataset with 5000 observations and one anomaly introduced. We use 5 random trees and height = 5. We select $\eta = 5\%$. Majority gives us the best result as shown in Table 5.1.

Table 5.1: Rank of anomaly for different combination approaches

	Avg	Minimum	Majority
RANK	508	321	**240**

5.2.4 Building Detection Trees using Feature Clustering

The advantages of building random trees are: (1) they are very fast, (2) they do not require any prior knowledge about the data distribution. In the previous section, we analyzed how the random trees will perform with high-dimensional features. In this section, we discuss why random trees fail in situations where some anomalies can only be detected by one subset of features, and other anomalies are better detected by other subsets of features. Consider the example in Fig.5.8, where anomalies are represented with red triangles, while the normal observations are shown with blue dots.

Features 0, 1, and 2 are strongly correlated while 4,5, and 6 are strongly correlated, but the two subsets are not correlated. The correlation matrix is represented in Table 5.2. Also note that an anomaly can only be detected with certain combination of features. To find such anomalies which can only be captured using the correlated features, we propose

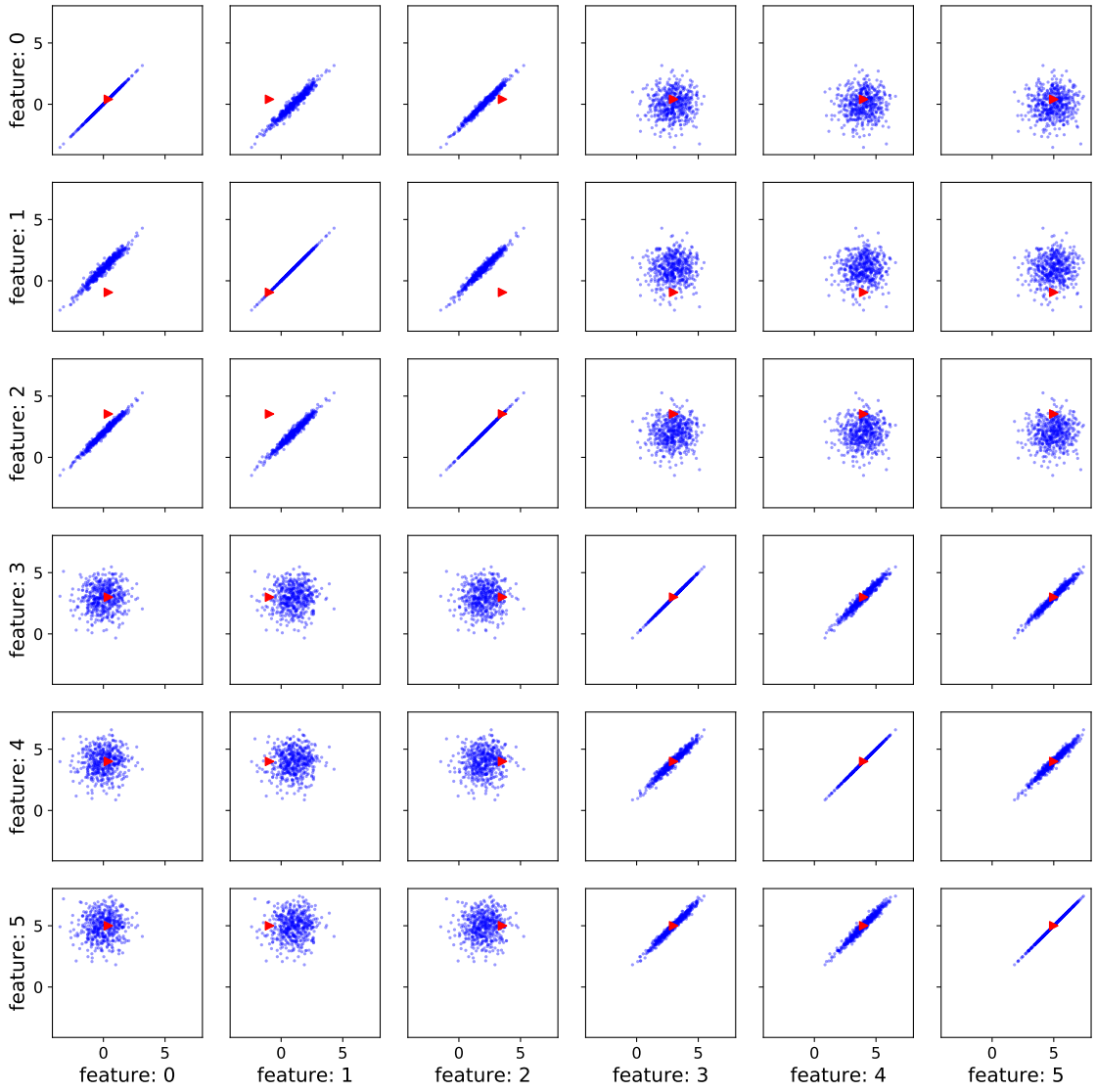


Fig. 5.8: A synthetic data where feature are interacted

a two step approach in which we first group the correlated features together, then apply the random trees detection algorithm in each feature cluster separately. We first discuss how to measure whether a tree can potentially detect anomalies using an information theoretic technique.

If an observation o is a true anomaly, *i.e.* is most anomalous, then, there exists some partition such that the neighborhood of o has the lowest density (most sparse) compared to normal observations. Assume the cells obtained from the HS tree offer such a partition;

Table 5.2: Correlations between features for data shown in Figure 5.8

feature	0	1	2	3	4	5
0	1.00	0.97	0.99	0.03	0.03	0.03
1	0.97	1.00	0.98	0.04	0.04	0.03
2	0.99	0.98	1.00	0.03	0.03	0.03
3	0.03	0.04	0.03	1.00	0.98	0.99
4	0.03	0.04	0.03	0.98	1.00	0.98
5	0.03	0.03	0.03	0.99	0.98	1.00

then, there exists a tree depth h such that the cell containing o has the smallest number of observations (namely, the probability of it being anomaly is the largest).

For simplicity, we consider a dataset with only two features X and Y that are uniformly distributed. First we consider the worst case when the true anomaly cannot be detected over all possible partitions, that is when all cells have the same probability, that is when X and Y are independent. Consider a toy example with tree depth = 2, where each dimension is split into two parts, the probability of each cell is 0.25, and the anomaly (orange observation) cannot be detected. In this case, data has the most uncertainty, that is $H(X, Y) = H(X) + H(Y)$. Now we consider the best case where all normal observations fall in one dense cell and the anomaly falls in any other cell. In this case, there is no data uncertainty, *i.e.* $H(X, Y) = 0$.

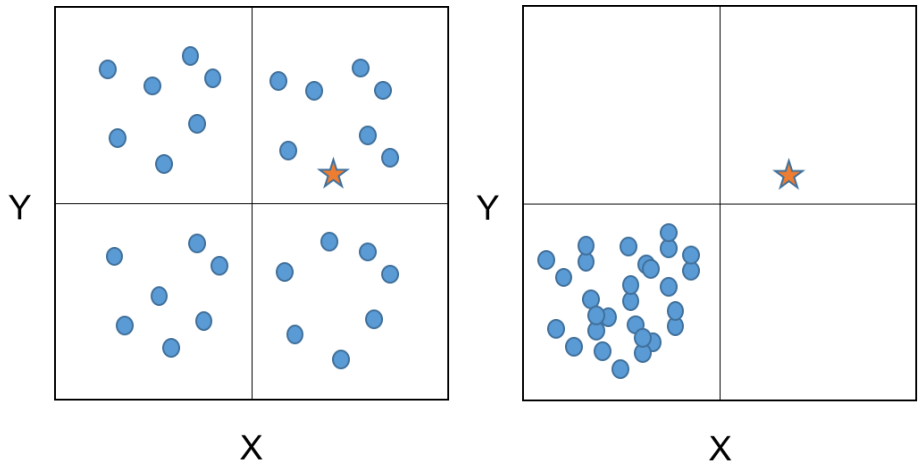
Therefore, we use the following measure to evaluate the discriminative power of each tree:

$$\frac{H(X, Y)}{H(X) + H(Y)}.$$

In general, for a tree with m features and height = h, we measure:

$$\frac{H(X_1, X_2, ..., X_m)}{\sum_{i=1}^{m} H(X_i)}$$

This measurement will be lower when anomalies can be detected from the normal observations.



(a) tree leaves with most uncertainty (b) tree leaves with least uncertainty

Fig. 5.9: Two cases with outlier in different positions with respect to normal data

Effect of dependency on tree

For simplicity, we now consider the effect of linear dependency in a problem with two features. In this case, $H(X,Y) = H(X) + H(Y) - I(X;Y)$, and

$$\frac{H(X,Y)}{H(X)+H(Y)} = 1 - \frac{I(X;Y)}{H(X)+H(Y)},$$

therefore, the discriminative power is negatively correlated with the mutual information between X and Y.

Why use feature clustering?

Given m features, we want to partition the feature spaces into different cells such that each cell contains features that are highly dependent with each other. Therefore, the intra-tree dependency will be maximized for the trees built on each partition.

The problem of selecting k meaningful feature sets from m features can be represented as an optimization problem:

$$arg\max_{FS} \sum_{i=1}^{k} \sum_{X \in FS_i} I(X; \lambda_i)$$

where λ_i is the representative feature from the feature set FS_i.

As a result, the problem of partitioning the feature spaces is similar to the classical clustering problem. The only difference is that instead of using the classical Euclidean distance, we use the feature dependency as a similarity metric.

The Affinity Propagation (AP) [30] algorithm identifies exemplars among data points and forms clusters of data points around these exemplars. It operates by simultaneously considering all data points as potential exemplars, and exchanging messages between data points until a good set of exemplars and clusters emerges. We use Affinity Propagation in our work, given the following advantages compared to k-means clustering:

- AP accepts similarity matrix instead of distance matrix, so that no conversion is needed.
- AP finds actual feature exemplar while k-means finds the 'mean' in a cluster which may not be a data point.
- AP does not require initial set of exemplars and users do not need to explicitly specify the number of clusters.

5.2.5 Experimental Results

For the synthetic data shown in Fig 5.8, after we apply clustering, the cluster labels are (0,0,0,1,1,1), which means features 0,1 and 2 are clustered together while features 3,4, and 5 are clustered together. We measure the ranks of the anomalies and also the AUC score. We build 10 trees using all of the 6 features and 5 trees each for the two feature clusters for fair comparison. Results are shown in Figure 5.10, in which the results obtained from

feature clustering are indicated with "clus". For the same number of trees and the same height of trees, feature clustering results in better solutions than purely random trees.

In Section 5.2.2, we justified that we need more number of trees or 'taller' trees to cover more features. By applying feature clustering, for each cluster, the number of features are decreased. We design two sets of experiments to compare the performance of our clustering based algorithm and random trees algorithm:

- Fix the height of each tree, vary the number of trees used for our method and random trees method. Results, shown in Figure 5.11(a), indicate that our method outperforms random trees on all values of T, *i.e.* number of trees used.

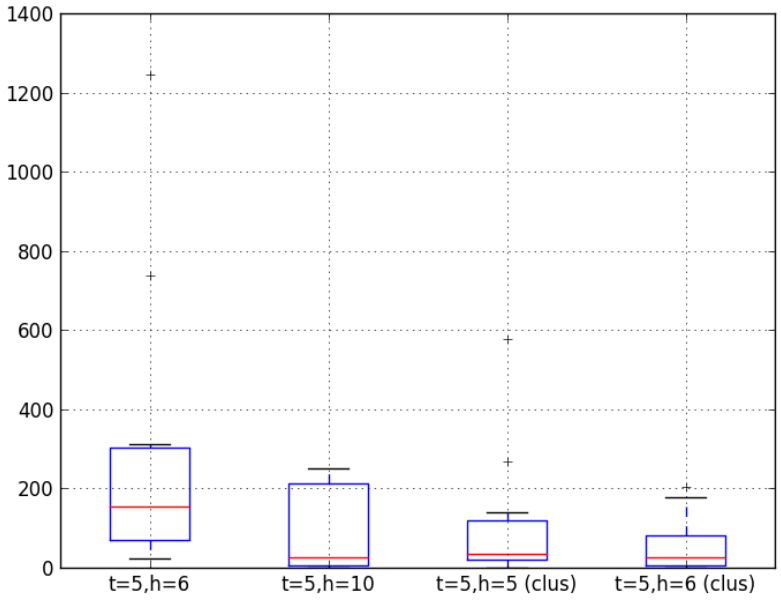
- Fix the number of trees used, vary the height of each tree, result shown in Figure 5.11(b) indicate that our method outperforms random trees method and becomes very stable when the height of tree is 'taller', *i.e.*, $h = 7$.

We apply our clustering based algorithm on the Polish Bankruptcy dataset which has 5910 companies, 410 bankrupted companies (anomalies) became bankrupted after one year, as described in Section 1.5.2. Since there are many missing values in the original dataset, we replace missing values by different methods: column mean, column median and replace by the nearest neighbor by sorting based on gross profit. To compare the performance with Tan *et al.*'s work, we perform two sets of experiments:

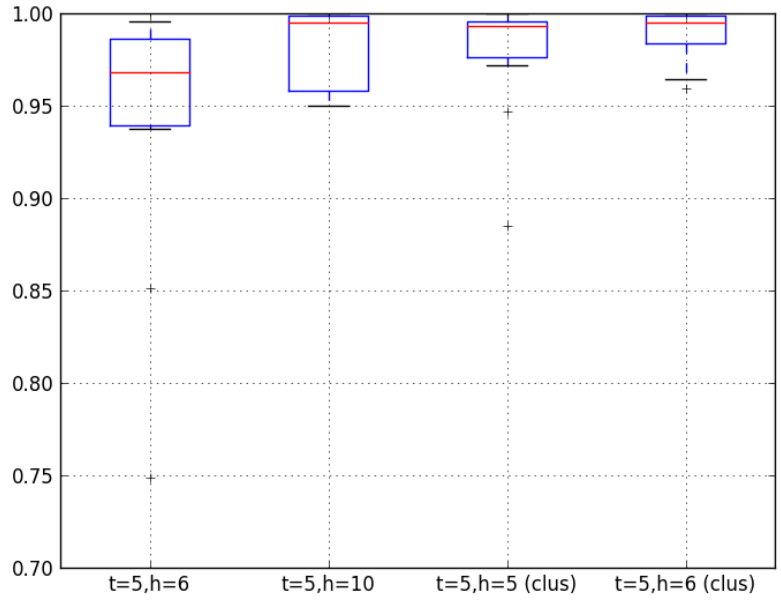
- Using 50 trees in total, each tree of height 10; for our method, 50 trees are equally distributed in each cluster.

- Using 100 trees in total, each tree of height 15; for our method, 100 trees are equally distributed in each cluster.

These experiments are repeated 30 times for each of the different methods to fill in the missing value. Results in Table 5.3 show that we are able to increase the AUC performance by at least around 20% for all the three datasets, after missing values are replaced. We observe from the results that:

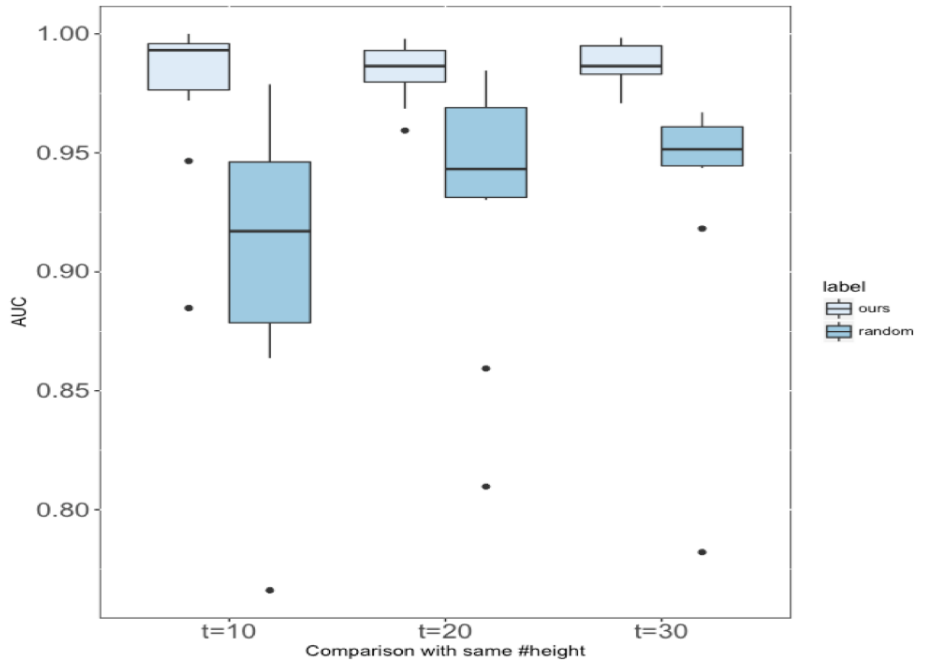


(a) Rank of anomalies

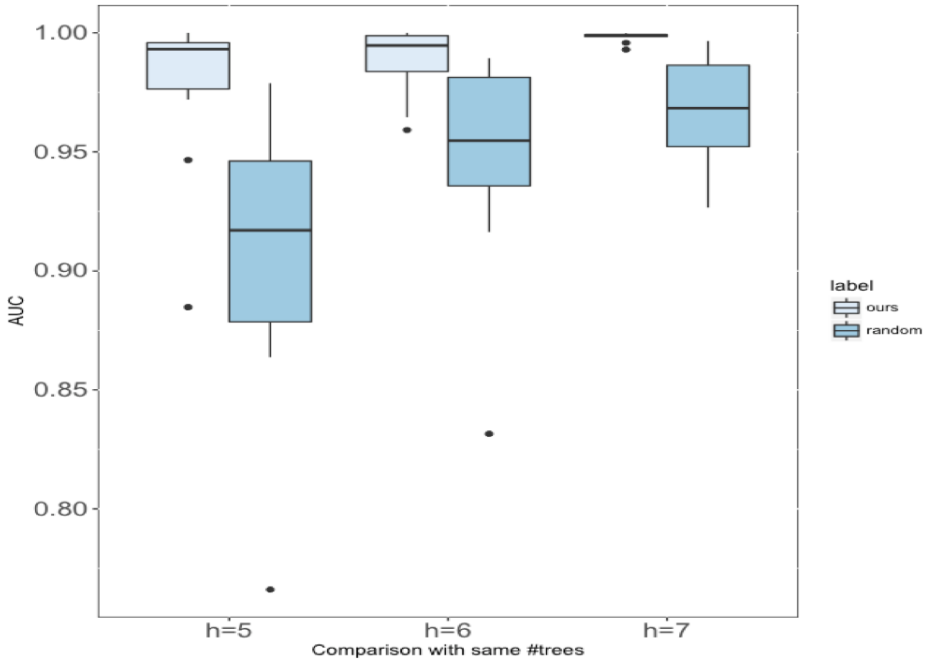


(b) AUC

Fig. 5.10: Performance comparison for our feature clustering method and random trees method on a synthetic dataset



(a) AUC comparison when height of trees is fixed



(b) AUC comparison when number of trees is fixed

Fig. 5.11: Performance comparison for our feature clustering method and random trees method for number of trees and height of trees on a synthetic dataset

- Our method achieves better results when using the same cost of building trees.
- The clustering cost is less than 5% of the overall time.

Table 5.3: Averaged AUC for Bankruptcy Dataset over 30 trials

	Mean		Median		Sorted	
	random	clus	random	clus	random	clus
t=50,h=10	0.436	**0.602**	0.452	**0.689**	0.527	**0.564**
t=100,h=15	0.556	**0.709**	0.554	**0.709**	0.581	**0.656**

5.3 Evolutionary Algorithm for Partitioning Data Space

Currently, in our model, when we build detection trees, each tree denotes a random partition. In each partition, features are selected randomly and then split into half. We want to further improve the partition generation process given that picking up irrelevant features might dilute the general detection performance, and instead of splitting the feature into half spaces, we want to find a better split point where extreme values (anomalies) can be better separated from the normal data.

5.3.1 How to partition the data space to separate outliers

Outliers reside in the sparse parts of a dataset, while the normal observations in the denser area. For a dataset D, we want to find a $partition = \{c_1, ..., c_p\}$ which best separates the anomalies from the normal objects, ***i.e.***, the normal objects are grouped together in some partitions while each outlier is separated with one partition. We define:

$$Density(\{c_1, ..., c_p\}) = \sum_{i=1}^{p} density(c_i) \tag{5.6}$$

Claim: All the outliers are separated in a data space DS if Density d is the maximum over the data space DS.

Proof by contradiction: Suppose d is maximum and there exists at least one outlier not separated.

$\exists o \in Outliers$ such that o is grouped with a normal group $\mathcal{N}$ and form a partition $\mathcal{N}' = \{o\} \cup \mathcal{N}$. By definition, $density(\mathcal{N}') < density(\mathcal{N})$ since outlier lies in a low density

area. Let the Density over $DS \setminus \mathcal{N}'$ be $d_0 = d - density(\mathcal{N})$. There exists a partition $\{DS \setminus \mathcal{N}', o, \mathcal{N}\}$ such that the cost over it is

$$\begin{aligned} d_1 &= d_0 + density(\mathcal{N}) \\ &= d - density(\mathcal{N}') + density(\mathcal{N}) \\ &> d \end{aligned}$$

which contradicts the assumption that d is the maximum possible density.

In order to find the best partition for a dataset with n data points, it requires $O(2^n)$ time complexity. Therefore, we consider EA as the optimization tool to find the best partitions.

5.3.2 Space-partitioning Forest

Each detection tree partitions the data space into parts where the normal objects are grouped in denser areas, and outliers reside in sparse parts. Finding one optimal tree may suffer from over-fitting problem. To avoid over-fitting, we aim at building a collection of T space-partitioning trees, *i.e.* a space-partitioning forest. Therefore, for T full binary trees of the same height h, the query for each data object is $O(T \cdot h)$.

Individual Representation

The goal is to find a collection of trees to better capture anomalies. Each individual is a collection of T trees, and each tree is represented in its *level-order* traversal representation. Each tree of height h (start from 1) consists of $2^h - 1$ interior nodes, each interior node is represented as a tuple: *(attId, splitVal)*, represents the id of the feature and the cutoff value at that node. Thus, each tree of height h is represented as a vector of nodes:

$$< node_1, node_2, ..., node_{2^h-1} >$$

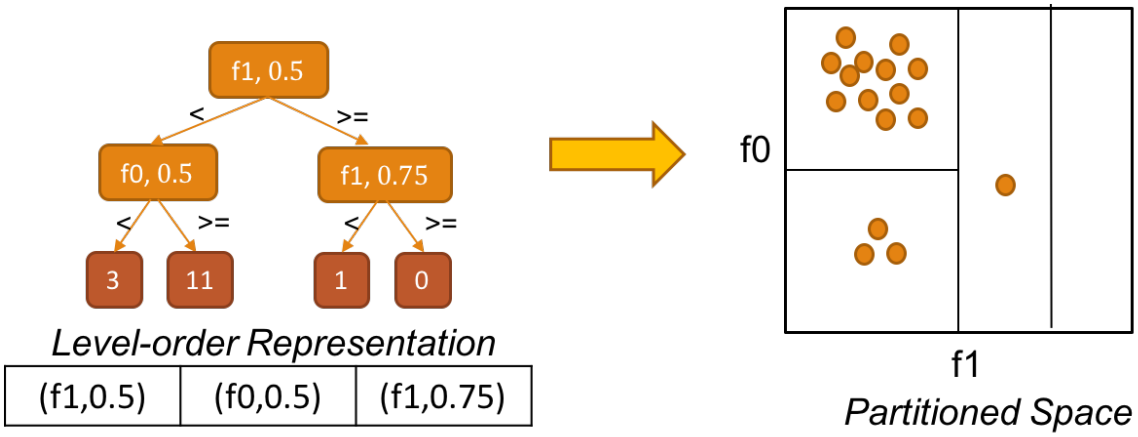


Fig. 5.12: An illustration of individual representation used in EA

Each individual is a collection of T trees, is a set of T such vectors. An illustration can be found in Figure 5.12.

Cost function

We defined a density function in Equation (5.6) for a partition $\{c_1, ..., c_p\}$. To estimate the density inside a node, we use the maximum distance from any data point in the node to the centroid as an approximate. Then, we use minimizing the max distance to maximize density over each sub-partition (node):

$$MaxDist(node) = \text{Max}\{\text{distance}(centroid(node), node.data_i), i = 1...|node.data|\}$$

Cost function of each tree is defined as the averaged maximum distances among all its leaf nodes:

$$cost(tree) = \frac{1}{2^H} \sum_{l=1}^{2^H} \text{MaxDist}(leaf_l).$$

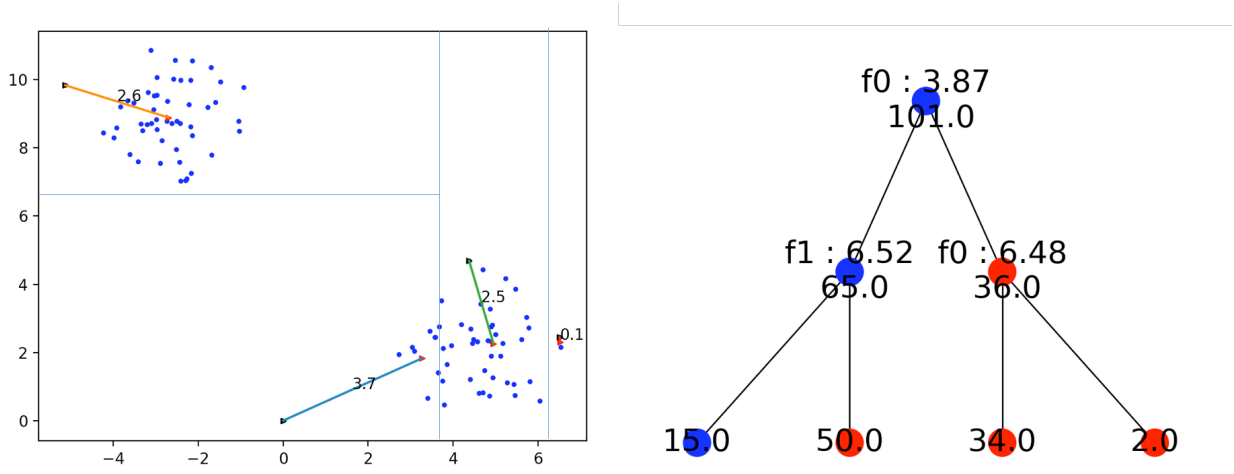
Each individual in EA is defined as a collection of T trees. The cost of an individual is defined as follows:

$$cost(individual) = \frac{1}{T} \sum_{i=1}^{T} cost(tree_i)$$

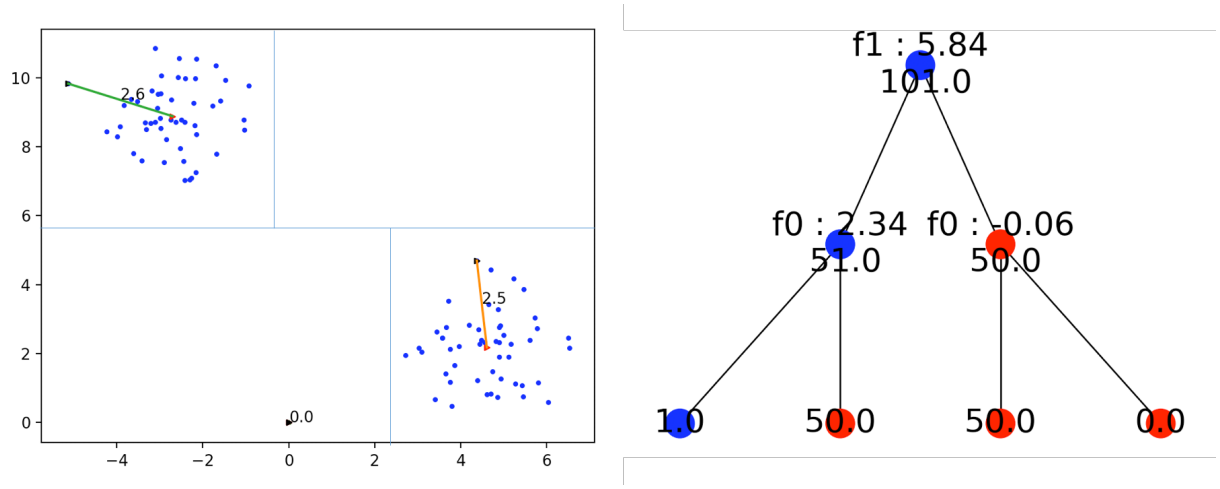
This cost function is to be minimized.

We first want to examine the effectiveness of our cost function in finding anomalies.

The following figures show two partitions created from two different detection trees.



(a) Cost is higher when outlier is partitioned with normal objects



(b) Cost is lower when outlier is separated

Fig. 5.13: Cost computation for two trees with different degrees of separation of outliers from the other data points

We notice in Figure 5.13 that when cost is high, anomalies cannot be separated using the partition generated. For example, in the tree build in Figure 5.13(a), there are 101 data points generated, each tree leaf contains 15, 50, 34 and 2 points respectively; the partitioned data space is shown in the left, the cost for this tree is equal to (2.6+2.5+3.7+0.1)/4 = 2.225 while the tree built in Figure 5.13(b) partitions the outlier from the normal observations and has a cost of (2.6+2.5)/4 = 1.275, which is lower than before. This illustrates the effectiveness of our cost function in separating anomalies from normal objects.

Mutation

The idea of mutation is to not change the individual drastically, instead we modify one (or a few) nodes in one tree to keep the diversity. In order to find better solutions, we discard the offspring which gives us higher cost. Intuitively, this will lead to a hill-climbing searching procedure which could be computationally expensive, therefore, we add some constraints in our mutation procedure such that we set a counter and constrain it to be less than N times to find a better mutant. In our experiment, we tried $N = 0, 1, 5$ and $N \propto$ current generation. When $N \propto$ current generation used, it means we want finer tuning at the end of convergence. We observe that using the last strategy has higher cost reducing rate than the others. The results for a synthetic data are shown in Fig 5.14.

Algorithm 8 Mutation

Input: mutation rate p_m, individual $individual$, current generation gen

Output: return a mutated individual

$counter$ = 0 ;

$prevCost$ = cost(*individual*) ;

mutant = randomly change one node from a random tree in *individual* ;

while $counter < N$ *and cost(*mutant*)* $>= prevCost$ **do**

  Increment $counter$;

  mutant = randomly change one node from a random tree in *individual* ;

  $prevCost$ = cost(*individual*) ;

end

Crossover

In our algorithm, each individual is a set of independent detection trees. For crossover, we apply single-point crossover on the two parents sets. For example, for two individuals $\{T_1, T_2\}$ and $\{T_3, T_4\}$, after applying single-point crossover, we may obtain two offspring $\{T_1, T_4\}$ and $\{T_2, T_3\}$.

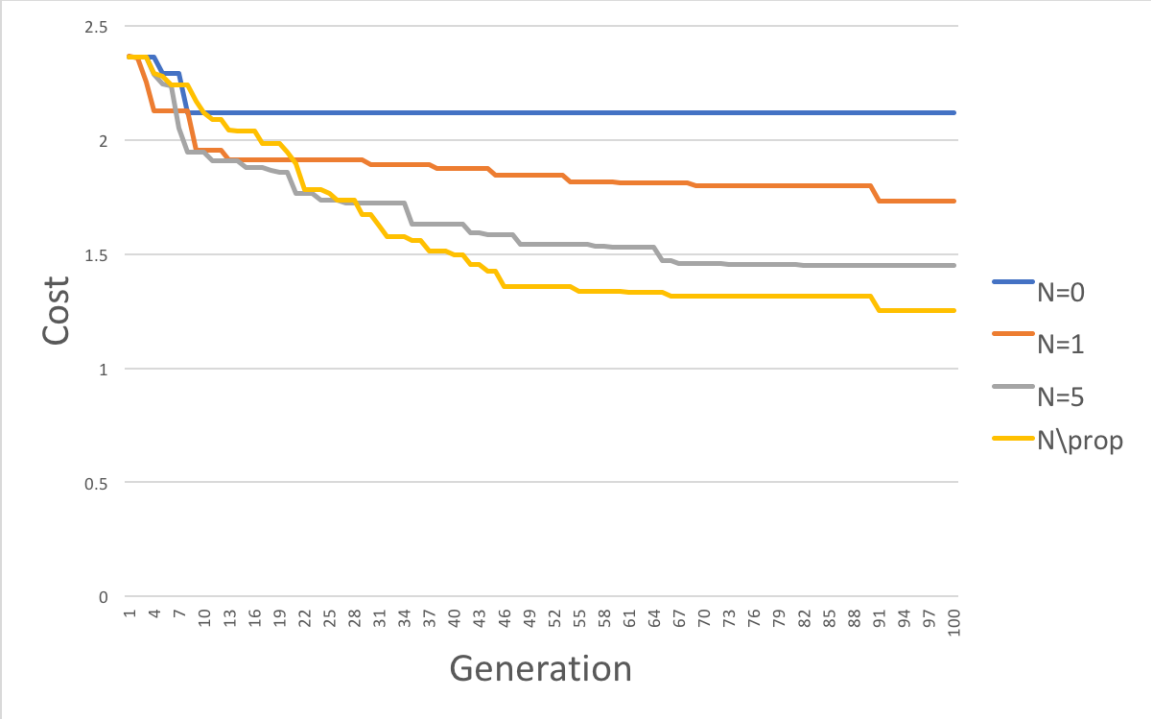


Fig. 5.14: Evolution of solution quality with number of iterations, with different strategies for synthetic dataset, when $p_c = 0.6$

The cost over iterations for different crossover probability p_c is shown in Fig 5.15.

Selection

We add elitism in our selection procedure. Which means in each iteration, we retain the best individuals (a fixed fraction e of the population size) in the next iteration. For parents selection, we use fitness proportion selection (inverse of cost). The cost over iterations for different e is shown in Fig 5.16, using e of 0.2, 0.4, or 0.8 seems to give reasonable performance, rather than a value of e that is too large or too small.

5.3.3 Algorithms

The overall algorithm is shown in Algorithm 9.

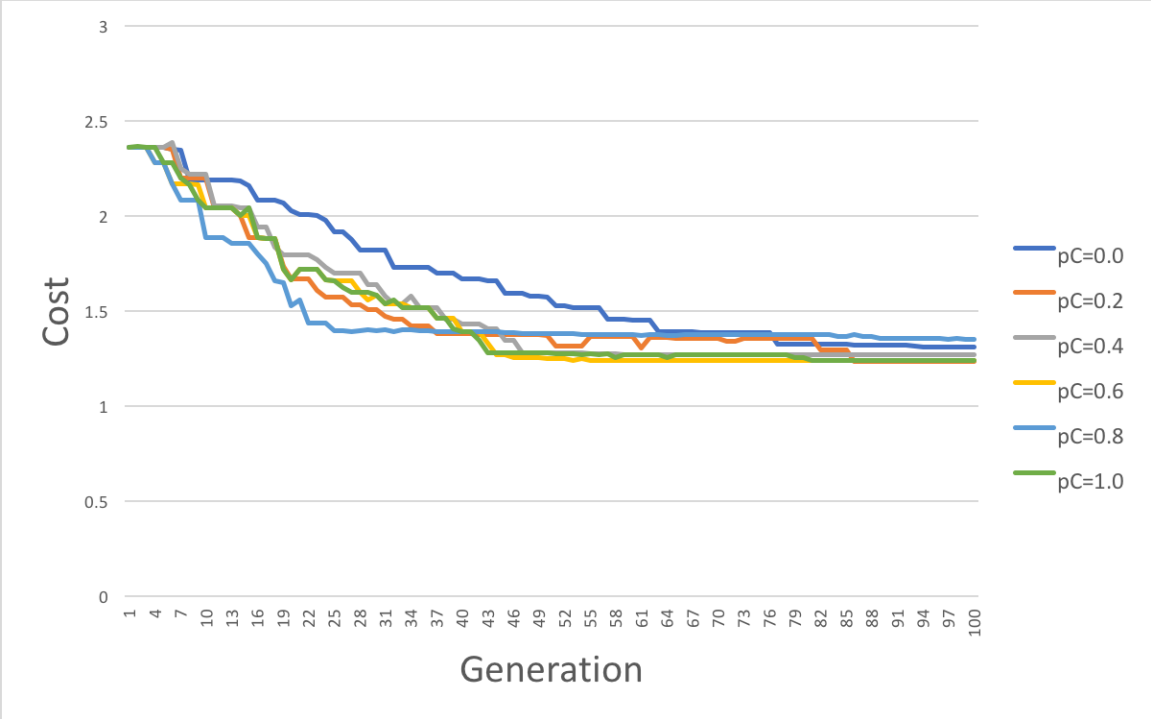


Fig. 5.15: Evolution of solution quality with number of iterations, for different values of crossover probability for synthetic data, $p_m = 0.2$

Algorithm 9 Evolutionary Algorithm

Input: population size N, mutation probability p_m, crossover probability p_c

Output: return the possible best individual

Pop = generate N random initial individuals ;

$isTerminated$ = False ;

$best$ = None ;

while *not isTerminated* **do**

  $C = \{\text{cost}(i) | \forall i \in Pop\}$;

  Pop' = $\emptyset$;

  while $|Pop'| < n$ **do**

    parents = select(Pop, C) ;

    offspring = reproduce(*parents*, p_c, p_c) ;

    add *offspring* to *Pop'* ;

  end

  Pop = *Pop'* ;

  $isTerminate$, $best$ = testTermination(Pop, $best$)

end

return $best$

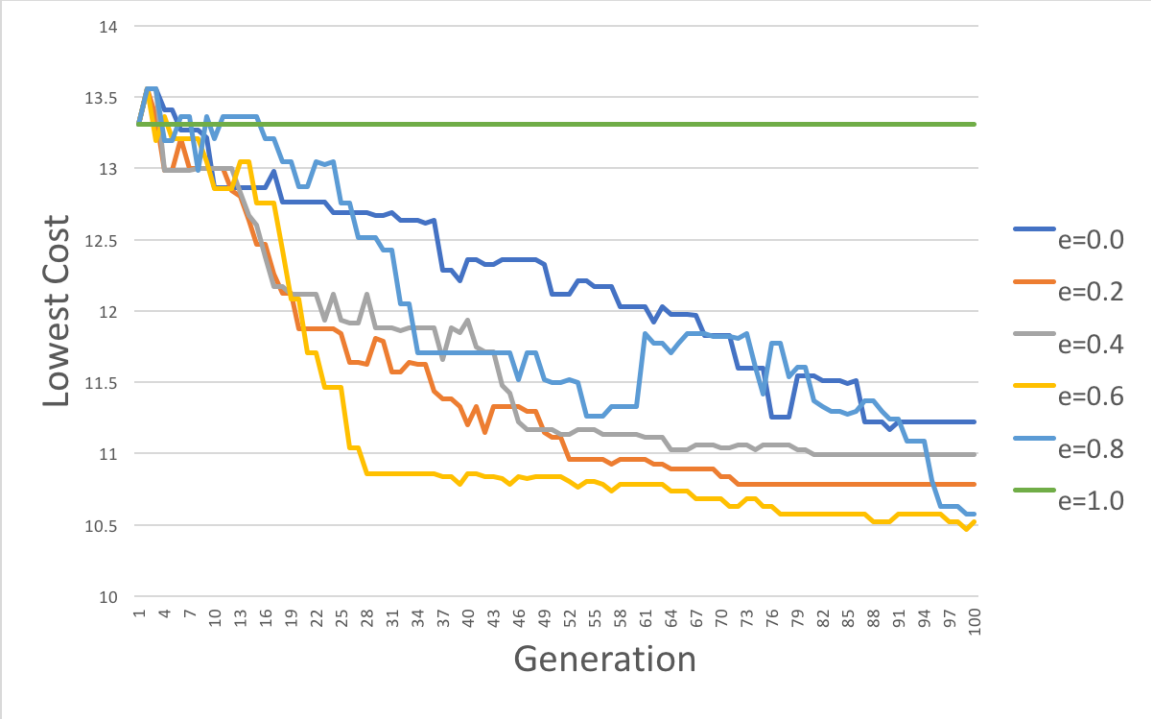


Fig. 5.16: Evolution of solution quality with number of iterations, for different values of elitism e with different for synthetic dataset

Algorithm 10 testTermination

Input: a population Pop, the best individual $best$ previously seen

Output: a tuple ($isTerminated$, $individual$) where $isTerminated$ is True if terminate, $individual$ is the best individual returned

individual = argmin$_i\{\text{cost}(i)|\forall i \in Pop\}$;

if *reach maximum generation* **then**
  return (True, *individual*) ;

end

else

  if $|cost(individual) - cost(best)| < \delta$ **then**
    return (True, *individual*) ;

  end

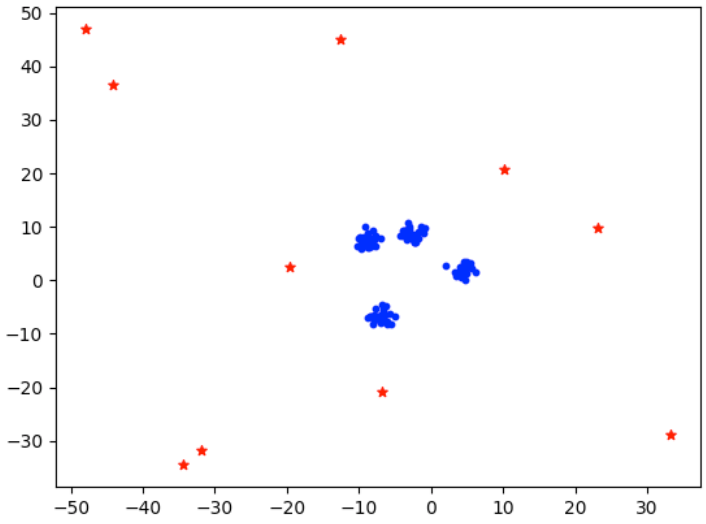
  else
    return (False, *individual*) ;

  end

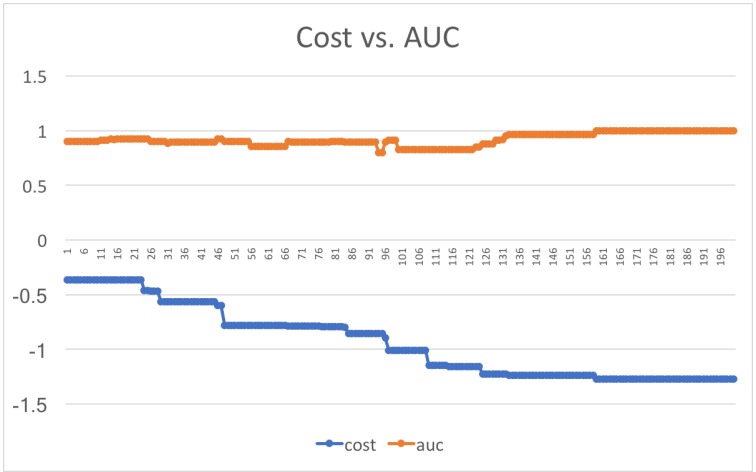
end

5.3.4 Preliminary Results for EA

In this section, we show some preliminary results for comparison of using EA to generate the random trees with pure random trees generation. We generate a synthetic dataset in which the normal data are drawn from a Gaussian distribution while outliers are uniformly distributed, the data is shown in Fig 5.17a. The results of using the EA are shown in Fig 5.17b. We observe that EA successfully finds all the outliers (AUC score is 1.0) when it converges.



(a) synthetic data – normal data from gaussian, outliers uniformly distributed



(b) Cost vs AUC for using EA to generate random trees

Fig. 5.17: Results of using EA on a synthetic dataset

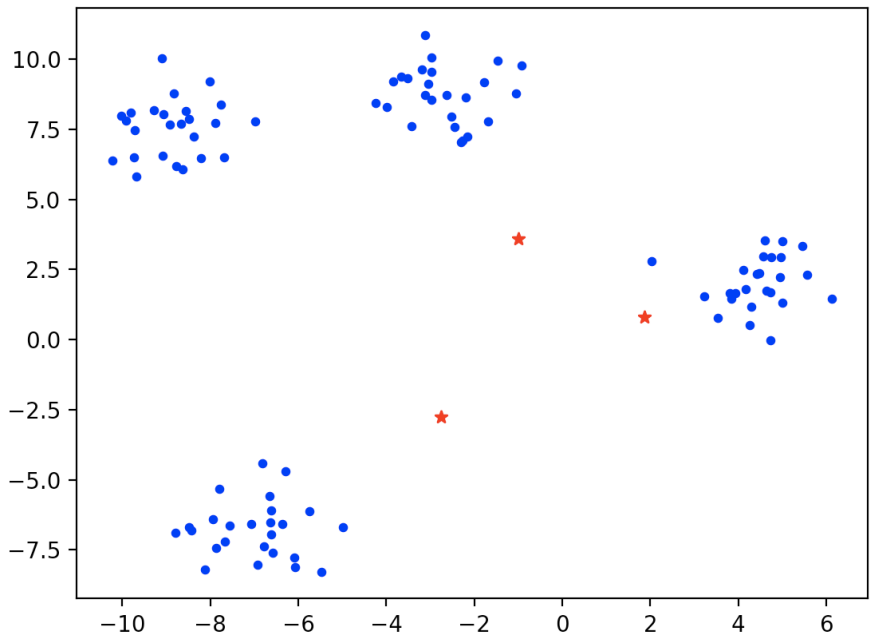


Fig. 5.18: Synthetic dataset with 4 clusters, outliers are inserted in between

Another synthetic dataset is shown in Fig 5.18. In this experiment, we want to see whether EA can better separate the outliers which are in between clusters, as compared to pure random trees. In our experiments, we fix the tree height to be 4 for both random and EA-generated trees. We observe that when using 10 trees, the AUC is 0.64 for random while is 0.94 for EA. We also notice that in our experiment that when we use large tree height and number of trees, the improvement of using an EA over random tree is not very significant. The reason is that if given enough tree cost (*i.e.* tree height and number of trees), the probability of covering all combinations of all features is high (discussed inp$_m$$Section$ 5.2.2).

5.4 Conclusion

In this chapter, we first reviewed the status of ensemble methods for streaming anomaly detection. We give a justification for the number of trees and tree heights one should use for random forest in Section 5.2.2, by converting this problem to the coupon collector

problem. For anomalies which are only detectable from a set of correlated features, we propose to apply feature clustering first, then build random trees on each of the clusters. Our evaluations on both synthetic and real-world datasets show performance improvements. To better separate data space for detecting anomalies, we propose to use (instead of pure randomly partition datasets using random trees) an Evolutionary Algorithm for minimizing partitioning cost based on the assumption that anomalies lie in the sparse area than normal observations. Our preliminary results show that using EA-based approach, anomalies can be better separated from the entire dataset.

CHAPTER 6
CONCLUSION AND FUTURE WORK

In this dissertation, we studied the application of ensemble learning in the context of anomaly detection for both static and streaming datasets. Chapters 2, 3, and 4 discussed how to design ensemble algorithms for static datasets while Chapter 5 introduced our solution for detecting anomalies on streaming data.

In this chapter, we summarize the results in previous chapters, and propose directions for future research.

6.1 Summary

In Chapter 2, we focused on two aspects: (1) how to design different strategies for combining decisions from different base algorithms; and (2) how to introduce ensemble diversity by detecting anomalies on multiple subsamples from the datasets. We design and evaluate multiple strategies using score normalization, rank aggregation and majority voting, to combine the results from 6 state-of-the-art base detection algorithms. Our evaluation on multiple real-world benchmark datasets show that by using ensembles, we achieve better detection performance than most of the base algorithms. By using *minimum ranking* and *maximum score*, we are able to achieve the best accuracy among all the algorithms. Us-

ing our proposed bootstrapping methods, we achieve results competitive with the current approach, but with a much more less computational cost.

In Chapter 3, we design a two-phase sequential method where two algorithms are used. The first algorithm is used to select a subset of observations in which most anomalies are suspected to be contained. Another algorithm is then executed such that all the observations are evaluated with respect to such subsets. Our evaluation results show that by selecting the top ranked observations, we are able to reach better solutions than base algorithms. Also, best results are obtained by combining algorithms that are substantially different.

In Chapter 4, a novel adaptive sampling and learning approach was designed and evaluated. In this approach, the score output of the base algorithm is used to determine the hard-to-detect examples, and iteratively resamples more points from such examples in a completely unsupervised context. The results show that using this idea, we achieved better solution than base algorithms and other ensemble methods.

In Chapter 5, we analyze the deficiencies of a recent anomaly detection algorithm based on the concept of random trees. We gave both theoretical and empirical analysis addressing how to choose parameters used in the model. Then, we proposed to partition the feature space into similar clusters, followed by independently constructing multiple random trees. Our evaluations on both synthetic and real-world datasets show a better detection accuracy. Instead of randomly separating the data space for finding anomalies, we proposed to use Evolutionary Algorithms in which the objective is to group the normal observations in denser area, such that the anomalies can be separated more efficiently. Our preliminary results show a potential performance improvement using this new approach.

6.2 Future Work

For the independent ensembles, currently, we have mainly explored combining the results from the nearest-neighborhood-based algorithms. A potential extension is to use other

learning algorithms such as Neural Networks and Support Vector Machines. Another direction is using probabilistic models for combining the decisions. The challenge is how to convert a raw score to a probability indicating the anomalousness of an object.

In our empirical evaluation, we observed that the base algorithm RADA achieves the best performance among all the individual base algorithms. Therefore, in our sequential learning, by using RADA as the second algorithm, we obtained best results among all the algorithms pairs. Meanwhile, we noticed that using a worse algorithm in the second stage generated worse performance. One of the future works should address how to select a more accurate algorithm among all the base algorithms to use as the refined approach. This might be difficult in a pure unsupervised context, however, if a small sample from the dataset contains labels, then one can perform a test on the labeled subsample first, then select the algorithms to use for sequential learning.

For sequential learning, currently, we explore the sequential learning, where the second algorithm is applied to the first algorithm's outputs. Multi-layer sequential learning can be explored as follows: at the first stage, a decision is generated from independent ensemble; at the next layer, subsampling is done with respect to the independent ensemble decision; next, another layer of decision-making involves applying another detection algorithm. This leads to a more complex algorithm that may generate more refined results.

For the streaming anomaly detection, we currently update our model parameters only considered the data distribution drift. We used a batch based method which updates the model periodically. The model update for data distribution is easier than model update with feature space drift. Since our method begins with feature clustering (preprocessing), if the feature space drifts, then our model needs to be able to modify the feature clusters under concept drift automatically. Also, the current evaluations with Evolutionary Algorithms are done on separating anomalies on data space. It would be worth exploring whether an Evolutionary strategy can be designed to update model parameters in a streaming context.

REFERENCES

[1] "Basketball statistics and history," http://databasebasketball.com/.

[2] "Classification of packed executables," http://roberto.perdisci.com/projects/cpexe.

[3] "Kdd cup 1999 data," http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[4] "Machine learning journal," https://mlwave.com/.

[5] "Netflix prize," https://en.wikipedia.org/wiki/Netflix_Prize.

[6] N. Abe, B. Zadrozny, and J. Langford, "Outlier detection by active learning," *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '06*, p. 504, 2006. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1150402.1150459

[7] E. Achtert, T. Bernecker, H.-P. Kriegel, E. Schubert, and A. Zimek, "Elki in time: Elki 0.2 for the performance evaluation of distance measures for time series," in *Advances in Spatial and Temporal Databases*. Springer, 2009, pp. 436–440.

[8] C. C. Aggarwal, "On abnormality detection in spuriously populated data streams," in *Proceedings of the 2005 SIAM International Conference on Data Mining*. SIAM, 2005, pp. 80–91.

[9] ——, "Outlier ensembles: position paper," *ACM SIGKDD Explorations Newsletter*, vol. 14, no. 2, pp. 49–58, 2013.

[10] ——, "Outlier analysis," in *Data mining*. Springer, 2015, pp. 237–263.

[11] F. J. Anscombe, "Rejection of outliers," *Technometrics*, vol. 2, no. 2, pp. 123–146, 1960.

[12] D. Barbará, Y. Li, J. Couto, J.-L. Lin, and S. Jajodia, "Bootstrapping a data mining intrusion detection system," in *Proceedings of the 2003 ACM symposium on Applied computing*. ACM, 2003, pp. 421–425.

[13] BBC News, "Cyber-attack: Europol says it was unprecedented in scale," http://www.bbc.com/news/world-europe-39907965, 2017, online; accessed 2017.

[14] R. J. Beckman and R. D. Cook, "Outlier ... ... s," *Technometrics*, vol. 25, no. 2, pp. 119–149, 1983.

[15] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is "nearest neighbor" meaningful?" in *International conference on database theory*. Springer, 1999, pp. 217–235.

[16] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[17] ——, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[18] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: identifying density-based local outliers," in *ACM Sigmod Record*, vol. 29, no. 2. ACM, 2000, pp. 93–104.

[19] J. B. Cabrera, C. Gutiérrez, and R. K. Mehra, "Ensemble methods for anomaly detection and distributed intrusion detection in mobile ad-hoc networks," *Information Fusion*, vol. 9, no. 1, pp. 96–119, 2008.

[20] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[21] CNBC, "'Unprecedented' cyberattack hits 200,000 in at least 150 countries, and the threat is escalating." http://www.cnbc.com/technology/?redirect=1, 2017, online; accessed 2017.

[22] N. De Condorcet *et al.*, *Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix*. Cambridge University Press, 2014.

[23] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the royal statistical society. Series B (methodological)*, pp. 1–38, 1977.

[24] U. o. U. Department of Mathematics, "Using the receiver operating characteristic (roc) curve to analyze a classification model: A final note of historical interest," retrieved Aug, 2017.

[25] B. Efron and R. J. Tibshirani, *An introduction to the bootstrap*. CRC press, 1994.

[26] J. P. Egan, "Signal detection theory and {ROC} analysis," 1975.

[27] T. Fawcett, "An introduction to roc analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

[28] K. Fernandes, P. Vinagre, and P. Cortez, "A proactive intelligent decision support system for predicting the popularity of online news," in *Portuguese Conference on Artificial Intelligence*. Springer, 2015, pp. 535–546.

[29] Y. Freund and R. Schapire, "A desicion-theoretic generalization of on-line learning and an application to boosting," *Computational learning theory*, vol. 55, pp. 119–139, 1995. [Online]. Available: http://link.springer.com/chapter/10.1007/3-540-59119-2_166

[30] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *science*, vol. 315, no. 5814, pp. 972–976, 2007.

[31] S. GreenAM, "Signal detection theory and psychophysics," 1966.

[32] M. Gupta, J. Gao, C. C. Aggarwal, and J. Han, "Outlier detection for temporal data: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 9, pp. 2250–2267, 2014.

[33] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques*. Elsevier, 2011.

[34] J. A. Hanley and B. J. McNeil, "The meaning and use of the area under a receiver operating characteristic (roc) curve." *Radiology*, vol. 143, no. 1, pp. 29–36, 1982.

[35] S. Hettich and S. Bay, "The uci kdd archive [http://kdd. ics. uci. edu]. irvine, ca: University of california," *Department of Information and Computer Science*, vol. 152, 1999.

[36] H. Huang, K. Mehrotra, and C. K. Mohan, "Algorithms for detecting outliers via clustering and ranks," in *Advanced Research in Applied Artificial Intelligence*. Springer, 2012, pp. 20–29.

[37] ——, "Rank-based outlier detection," *Journal of Statistical Computation and Simulation*, vol. 83, no. 3, pp. 518–531, 2013.

[38] W. Jin, A. K. Tung, J. Han, and W. Wang, "Ranking outliers using symmetric neighborhood relationship," in *Advances in Knowledge Discovery and Data Mining*. Springer, 2006, pp. 577–593.

[39] F. Keller, E. Müller, and K. Böhm, "Hics: high contrast subspaces for density-based outlier ranking," in *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*. IEEE, 2012, pp. 1037–1048.

[40] E. M. Knox and R. T. Ng, "Algorithms for mining distancebased outliers in large datasets," in *Proceedings of the International Conference on Very Large Data Bases*. Citeseer, 1998, pp. 392–403.

[41] T. A. Lasko, J. G. Bhagwat, K. H. Zou, and L. Ohno-Machado, "The use of receiver operating characteristic curves in biomedical informatics," *Journal of biomedical informatics*, vol. 38, no. 5, pp. 404–415, 2005.

[42] R. Laxhammar, G. Falkman, and E. Sviestins, "Anomaly detection in sea traffic-a comparison of the gaussian mixture model and the kernel density estimator," in *Information Fusion, 2009. FUSION'09. 12th International Conference on*. IEEE, 2009, pp. 756–763.

[43] A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava, "A comparative study of anomaly detection schemes in network intrusion detection," in *Proceedings of the 2003 SIAM International Conference on Data Mining*. SIAM, 2003, pp. 25–36.

[44] A. Lazarevic and V. Kumar, "Feature bagging for outlier detection," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. ACM, 2005, pp. 157–166.

[45] J. Lee, B. Kang, and S.-H. Kang, "Integrating independent component analysis and local outlier factor for plant-wide process monitoring," *Journal of Process Control*, vol. 21, no. 7, pp. 1011–1021, 2011.

[46] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[47] R. Motwani and P. Raghavan, *Randomized algorithms*. Chapman & Hall/CRC, 2010.

[48] E. Müller, M. Schiffer, and T. Seidl, "Statistical selection of relevant subspace projections for outlier ranking," in *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*. IEEE, 2011, pp. 434–445.

[49] K. Noto, C. Brodley, and D. Slonim, "Anomaly detection using an ensemble of feature models," in *Data Mining (ICDM), 2010 IEEE 10th International Conference on*. IEEE, 2010, pp. 953–958.

[50] N. C. Oza and K. Tumer, "Classifier ensembles: Select real-world applications," *Information Fusion*, vol. 9, no. 1, pp. 4–20, 2008.

[51] T. Pang-Ning, M. Steinbach, V. Kumar *et al.*, *Introduction to data mining*, 2006.

[52] A. Patcha and J.-M. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," *Computer networks*, vol. 51, no. 12, pp. 3448–3470, 2007.

[53] R. Perdisci, A. Lanzi, and W. Lee, "Classification of packed executables for accurate computer virus detection," *Pattern Recognition Letters*, vol. 29, no. 14, pp. 1941–1946, 2008.

[54] T. Pevnỳ and J. Fridrich, "Novelty detection in blind steganalysis," in *Proceedings of the 10th ACM workshop on Multimedia and security*. ACM, 2008, pp. 167–176.

[55] D. Pokrajac, A. Lazarevic, and L. J. Latecki, "Incremental local outlier detection for data streams," in *Computational Intelligence and Data Mining, 2007. CIDM 2007. IEEE Symposium on*. IEEE, 2007, pp. 504–515.

[56] Radware, "2015-2016 global application network security report," 2015, online; accessed 2017. [Online]. Available: www.radware.com/newsevents/pressreleases/radwares-2015-2016-global-applications-and-network-security-report

[57] J.-L. Reyes-Ortiz, L. Oneto, A. Sama, X. Parra, and D. Anguita, "Transition-aware human activity recognition using smartphones," *Neurocomputing*, vol. 171, pp. 754–767, 2016.

[58] L. Rokach and O. Maimon, *Data mining with decision trees: theory and applications*. World scientific, 2014.

[59] S. Sadik and L. Gruenwald, "Research issues in outlier detection for data streams," *ACM SIGKDD Explorations Newsletter*, vol. 15, no. 1, pp. 33–40, 2014.

[60] V. Sandulescu and M. Chiru, "Predicting the future relevance of research institutions-the winning solution of the kdd cup 2016," *arXiv preprint arXiv:1609.02728*, 2016.

[61] S. Sathe and C. Aggarwal, "Lodes: Local density meets spectral outlier detection," in *Proceedings of the 2016 SIAM International Conference on Data Mining*. SIAM, 2016, pp. 171–179.

[62] L. Shoemaker and L. O. Hall, "Anomaly detection using ensembles," in *Multiple Classifier Systems*. Springer, 2011, pp. 6–15.

[63] V. Spruyt, "The curse of dimensionality in classification," *URL: http://www. vi*, 2014.

[64] S. C. Tan, K. M. Ting, and T. F. Liu, "Fast anomaly detection for streaming data," in *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, vol. 22, no. 1, 2011, p. 1511.

[65] J. Tang, Z. Chen, A. W.-C. Fu, and D. W. Cheung, "Enhancing effectiveness of outlier detections for low density patterns," in *Advances in Knowledge Discovery and Data Mining*. Springer, 2002, pp. 535–548.

[66] L. Tenenboim-Chekina, L. Rokach, and B. Shapira, "Ensemble of feature chains for anomaly detection," in *Multiple Classifier Systems*. Springer, 2013, pp. 295–306.

[67] A. Topchy, B. Minaei-Bidgoli, A. K. Jain, and W. F. Punch, "Adaptive clustering ensembles," *Proceedings - International Conference on Pattern Recognition*, vol. 1, no. i, pp. 272–275, 2004.

[68] J. W. Tukey, *Exploratory data analysis*. Reading, Mass., 1977.

[69] Y. Wang, K. Wu, and L. M. Ni, "Wifall: Device-free fall detection by wireless networks," *IEEE Transactions on Mobile Computing*, vol. 16, no. 2, pp. 581–594, 2017.

[70] K. Yamanishi and J.-i. Takeuchi, "A unifying framework for detecting outliers and change points from non-stationary time series data," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2002, pp. 676–681.

[71] Y. Zhao, F. Balboni, T. Arnaud, J. Mosesian, R. Ball, and B. Lehman, "Fault experiments in a commercial-scale pv laboratory and fault detection using local outlier factor," in *Photovoltaic Specialist Conference (PVSC), 2014 IEEE 40th*. IEEE, 2014, pp. 3398–3403.

[72] Z.-H. Zhou, *Ensemble methods: foundations and algorithms*. CRC press, 2012.

[73] M. Zikeba, S. K. Tomczak, and J. M. Tomczak, "Ensemble boosted trees with synthetic features generation in application to bankruptcy prediction," *Expert Systems with Applications*, 2016.

[74] A. Zimek, R. J. Campello, and J. Sander, "Ensembles for unsupervised outlier detection: challenges and research questions a position paper," *ACM SIGKDD Explorations Newsletter*, vol. 15, no. 1, pp. 11–22, 2014.

[75] A. Zimek, M. Gaudet, R. J. Campello, and J. Sander, "Subsampling for efficient and effective unsupervised outlier detection ensembles," in *Proceedings of the 19th ACM*

SIGKDD international conference on Knowledge discovery and data mining. ACM, 2013, pp. 428–436.

[76] M. H. Zweig and G. Campbell, "Receiver-operating characteristic (roc) plots: a fundamental evaluation tool in clinical medicine." *Clinical chemistry*, vol. 39, no. 4, pp. 561–577, 1993.

VITA

NAME OF AUTHOR: Zhiruo Zhao

PLACE OF BIRTH: Beijing, China

DATE OF BIRTH: July 1, 1989

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

M.S. 2013, Syracuse University, United States

B.S. 2011, Northwestern Polytechnical University, China