Carnegie Mellon University
HeinzCollege
INFORMATION SYSTEMS • PUBLIC POLICY • MANAGEMENT

Outlier Detection: Automation, Systems, and Applications

A DISSERTATION PRESENTED
BY
YUE ZHAO
TO
HEINZ COLLEGE OF INFORMATION SYSTEMS AND PUBLIC POLICY

COMMITTEE MEMBERS:
PROF. LEMAN AKOGLU (CHAIR)
PROF. ZHIHAO JIA (CO-CHAIR)
PROF. GEORGE H. CHEN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
IN THE SUBJECT OF
INFORMATION SYSTEMS AND MANAGEMENT

CARNEGIE MELLON UNIVERSITY
PITTSBURGH, PENNSYLVANIA
MAY 2023

Yue Zhao

Outlier Detection: Automation, Systems, and Applications

Abstract

Outlier detection (OD), also known as anomaly detection (AD), aims to identify samples that differ significantly from the general data distribution. It is a crucial machine learning task with numerous finance, security, and healthcare applications. In the past few decades, many *unsupervised* OD algorithms have been proposed where no ground truth labels are needed. However, using them in real-world applications still faces challenges in (*i*) how to select a good OD model to deploy *without* ground truth labels for evaluation and (*ii*) how to build efficient learning systems to support large-scale detection tasks. This thesis (partially) addresses these challenges by proposing automatic learning paradigms and scalable systems for OD.

In the first part of the thesis, I present four papers on how to select a good detection model as well as tune its hyperparameters without using labels. More specifically, Chapter 1 surveys and compares internal performance measures (IPM) for unsupervised outlier model selection (UOMS), which shows both its limitations and potential. In Chapter 2 and 3, I present two principled methods for UOMS (i.e., MetaOD and ELECT), which are based on meta-learning. In a nutshell, meta-learning facilitates the model selection on a new dataset by transferring knowledge from similar historical datasets. In Chapter 4, I extend the use of meta-learning to optimize the hyperparameters for unsupervised OD.

In the second part of the thesis, I introduce the design of three major OD systems that support large-scale OD tasks, which have been used in thousands of applications with millions of downloads. Chapter 5 introduces PyOD, a scalable CPU system, which equips with just-in-time compilation when possible. Chapter 6 introduces SUOD, which addresses the scheduling challenges in training multiple detection algorithms in a distributed fashion along with a case study for medical claim analysis. I finish the second part with Chapter 7, where I design TOD—the first multi-GPU OD system. Uniquely, TOD has a novel programming interface to represent diverse and complex detection algorithms as shared, basic tensor operators for GPU acceleration. Along with a newly designed quantization technique, TOD is on avg. 10.9 times faster than PyOD.

In the third part of the thesis, I present benchmarks and applications of OD. Chapter 8 shows the most comprehensive comparative study for outlier detection, ADBench, by including 30 datasets on 57 real-world datasets. Driven by industry need, ADBench unlocks the insights for detection algorithms under varying levels of supervision, different types of outliers, and data corruption and noise. Chapter 9 focuses on a specific security application, i.e., identifying email rule manipulation via outlier detection. In this work, we propose a detection framework ADMoE to detect security breaches by leveraging multiple sources of noisy labels.

I conclude the thesis with promising future directions of OD in Chapter 10.

Contents

Listing of figures

List of Tables

To XY — it was so lucky to meet you.

Acknowledgments

I want to first thank Prof. Leman Akoglu, Prof. Zhihao Jia, and Prof. George Chen, who have been on my committee(s) through the first paper presentation (i.e., the qualification exam in other Ph.D. programs). Leman taught me how to be a competent researcher. Her passion for research, efficiency in work, and patience in discussions are great motivations. I also appreciate the guidance and mentorship from Zhihao, who is always intelligent, cheering, and supportive. I will always remember how he encouraged me after (each) paper rejection and helped revise my job talk slides rounds after rounds. I also want to thank George, who has spent much time sharing research philosophy and paper writing suggestions with me. I would not make it without your support!

I am also fortunate to collaborate externally with Prof. Jure Leskovec from Stanford, Prof. Philip S. Yu from UIC, and Prof. Xia Hu from Rice University. These great mentorships and collaborations expand my research and will keep inspiring me. Throughout the years, I have been delighted to work and discuss with my great friends. An incomplete list includes Zain Nasrullah (U of Toronto), Zheng Li (Arima), Tianfan Fu (George Tech), Kexin Huang (Stanford), Yingtong Dou (VISA), Kay Liu (UIC), Kaize Ding (ASU), Changlin Wan (Genentech), Henry Lai (Rice), Daochen Zha (Rice), Minqi Jiang (SUFE), Xueying Ding (CMU), Lingxiao Zhao (CMU), Shubhranshu Shekhar (CMU), Cheng Cheng (CMU), Jeremy Lee (CMU). Of course, our life is beyond research collaborations, where we had fun hangouts and great game nights with Xiaobin Shen (CMU), Hongbo Fang (CMU), Shihan Li (CMU), Deying Song (CMU). This list can go longer, but all I want to say is "great to have you"!

I also want to say thanks to the support from Heinz College–the best interdisciplinary school in the world, which makes my Ph.D. smooth and enjoyable. Dean. Ramayya Krishnan, Prof. Martin Gaynor, Prof. Woody Zhu, and Ms. Michelle Wirtz have made my Ph.D. life easier and more manageable. Heinz always has a great environment that feels like home, and its openness and freedom for interdisciplinary research are unique.

I have lovely family and friends, who always tell me "it is fine to give up!" Prof. Jone Wen-ben has been my "life advisor" since I took his Intro to Digital Design at the University of Cincinnati. I am indebted to getting his wisdom during the hard times. My families are always supportive—I have been on international calls with my mom every day. Of course, my largest gain during my Ph.D. is forming a lovely family with my spouse. I would be in a much worse shape if I did not have your shoulder while crying 😭. Likely that will happen again and again in our life. Be prepared 😁

0 Introduction

0.1 What is Outlier Detection (OD)?

Outlier detection (OD), also known as Anomaly detection (AD)*, is the process of identifying unusual or unexpected events in a dataset [99, 238, 247, 268]. It is a widely used technique in many different fields, including anti-money laundering [162], rare disease detection [345], social media

*We use outlier detection and anomaly detection interchangeably in this thesis, while clear definition differences are rather subtle and "controversial". See [7] and [329] for more involved discussions.

analysis [333, 340], and intrusion detection [155]. Numerous methods have been developed in the last few decades [7, 150, 181, 183, 228, 257, 305, 347]. Among them, the majority are designed for tabular data (i.e., no time dependency and graph structure). Thus, we focus on the *tabular* OD algorithms and datasets in this thesis.

By the availability of supervision, OD can be roughly categorized into three settings:

- **Unsupervised Methods by Assuming Anomaly Data Distributions**. *Unsupervised OD methods are proposed with different assumptions of data distribution* [7], e.g., anomalies located in low-density regions, and their performance depends on the agreement between the input data and the algorithm assumption(s). Many unsupervised methods have been proposed in the last few decades [7, 32, 228, 254, 347], which can be roughly categorized into shallow and deep (neural network) methods. The former often carry better interpretability, while the latter handles large, high-dimensional data better. Please see the recent book [7], and surveys [228, 254] for additional information. **Unsupervised OD** often presents a collection of n samples $\mathbf{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \in \mathbb{R}^{n \times d}$, where each sample has d features. Given the inductive setting, the goal is to train an OD model M to output anomaly score $\mathbf{O} := M(\mathbf{X}) \in \mathbb{R}^{n \times 1}$, where higher scores denote for more outlyingness. In the inductive setting, we need to predict on $\mathbf{X}_{\text{test}} \in \mathbb{R}^{m \times d}$, so to return $\mathbf{O}_{\text{test}} := M(\mathbf{X}_{\text{test}}) \in \mathbb{R}^{m \times 1}$.

- **Supervised Methods by Treating Outlier Detection as Binary Classification**. *With the accessibility of full ground truth labels (which is rare), supervised classifiers may identify known anomalies at the risk of missing unknown anomalies*. Arguably, there are no specialized supervised outlier detection algorithms, and people often use existing classifiers for this purpose [7, 294] such as Random Forest [44] and neural networks [94]. One known risk of supervised methods is that ground truth labels are not necessarily sufficient to capture all types of anomalies during annotation. These methods are therefore limited to detecting un-

known types of anomalies [7]. Recent machine learning books [8, 94] and scikit-learn [235] may serve as good sources of supervised ML methods. **Supervised OD** also has the (binary) ground truth labels of $\mathbf{X}$, i.e., $\mathbf{y} \in \mathbb{R}^{n \times 1}$. A supervised OD model M is first trained on $\{\mathbf{X}, \mathbf{y}\}$, and then returns anomaly scores for the $\mathbf{O}_{\text{test}} := M(\mathbf{X}_{\text{test}})$.

- **Semi-supervised Methods with Efficient Use of Labels.** *Semi-supervised OD algorithms can capitalize the supervision from partial labels, while keeping the ability to detect unseen types of anomalies.* To this end, some recent studies investigate using partially labeled data for improving detection performance and leveraging unlabeled data to facilitate representation learning. For instance, some semi-supervised models are trained only on normal samples, and detect anomalies that deviate from the normal representations learned in the training process [14, 15, 335]. In ADBench, semi-supervision mostly refers to *incomplete label learning* in weak-supervision (see [358]). **Semi-supervised AD** only has the partial label information $\mathbf{y}^l \in \mathbf{y}$. The OD model M is trained on the entire feature space $\mathbf{X}$ with the partial label $\mathbf{y}^l$, i.e., $\{\mathbf{X}, \mathbf{y}^l\}$, and then outputs $\mathbf{O}_{\text{test}} := M(\mathbf{X}_{\text{test}})$.

Among the three settings, ***unsupervised OD*** is the primary one due to the following reasons:

1. **Data imbalance and lack of representative samples**: outliers, by definition, are rare events that differ from most of the data. Thus, collecting enough representative samples to create a (balanced) labeled dataset for supervised ML can be challenging.

2. **Subjectivity of labeling**: outliers are often defined in relation to some criteria, such as a statistical threshold or domain-specific knowledge. However, what may be considered an outlier to one person may not be considered an outlier to another. This subjectivity can make it challenging to create a consistent and reliable set of labels for outlier detection.

3. **Cost and time**: labeling data can be a time-consuming and expensive process. This is es-

pecially true for tasks that require domain expertise or specialized knowledge such as OD. Note that most OD samples are around insurance fraud and financial breaches where strong expertise is assumed.

4. **Exploration of new patterns**: unsupervised approaches may play a critical role in identifying unseen outlying patterns, which can be important in many applications.

Notably, existing literature primarily focuses on improving detection performance [76, 51, 105], which does not provide the full picture for real-world OD applications that are constrained by time budget and hardware as well. To make OD more practical, we provide a deep dive into two aspects, namely **automation** (see Part I) and **systems** (see Part II), in this thesis. Also, we present benchmarks and new OD algorithms for real-world applications in Part III.

Key aspects of unsupervised OD. we find it is important to select suitable methods for the underlying OD application without using any labels. Meanwhile, we should also ensure these unsupervised OD algorithms can efficiently execute on "big data". We further highlight the challenges associated with these two aspects in §0.2-0.5.

0.2 Automation Challenge 1: Unsupervised Outlier Model Selection (UOMS)

0.2.1 Importance of UOMS

Model selection aims to select a model from a set of candidate models for a task, given data. We consider the model selection problem for the *unsupervised* outlier detection task. Specifically, given a dataset for unsupervised OD, how can we identify – *without using any labels* – which outlier model (a detection algorithm and the value(s) of its hyperparameter(s)) performs better than the others on the input dataset? Importantly, note that as the outlier detection task is unsupervised, so is the model selection task. That is, an outlier model is to be selected without being able to validate any

candidate models on hold-out labeled data. The notion of a universally "best" outlier model does not exist; rather the best-performing model depends on the given data. On the other hand, model selection is a nontrivial one, provided there are numerous outlier detection algorithms based on a variety of approaches: distance-based [140, 243], density-based [46, 92, 285], angle-based [145], ensemble-based [8, 180, 236], most recently deep neural network (NN) based [55, 71, 254, 302], and so on. To add to this "choice paralysis", most models are sensitive to their choice of hyperparameters (HPs) with significant variation in performance [93], even more so for *deep* neural network (NN) based outlier models that have a long list of HPs [71]. Unsupervised model selection will likely be an increasingly pressing problem for deep detectors, as their complexity and expressiveness grow. Recent work hold out some labeled validation data for tuning such deep outlier models [172, 173, 254], which however is not feasible for fully unsupervised settings. These factors make outlier model selection a problem of utmost importance.

Obviously, unlike supervised model selection that can leverage a hold-out dataset for validation, In unsupervised learning, the algorithms must identify patterns and relationships in the data without the benefit of pre-labeled examples. This lack of prior information about the expected output makes it challenging to select a model that will accurately detect outliers. Additionally, there are many different models available for outlier detection, each with its own strengths and weaknesses, making it challenging to determine which one is best suited for a particular dataset. Finally, the definition of an outlier can vary depending on the context and the application, adding another layer of complexity to the model selection process. All of these factors contribute to the difficulty of selecting the right model for unsupervised outlier detection.

Despite its importance, the problem of Unsupervised Outlier Model Selection (UOMS) is a notoriously challenging one. Mainly, the absence of validation data with labels makes the problem hard. Moreover, there does not exist a universal or well-accepted objective criterion (i.e. loss function) for outlier detection, which makes model comparison infeasible. Besides its unsupervised na-

ture, the search space for UOMS can be quite large with the arrival/popularity of deep NN-based models with many HPs.

0.2.2 Related Work on UOMS

Perhaps due to these challenges, UOMS remains a vastly understudied area. Most work in the outlier mining literature focus on designing new detection algorithms, such as those for unique settings: streaming [101, 195, 284], contextual [176, 199], human-in-the-loop [62, 152] detection, etc.

Supervised Model Selection: In supervised learning, model selection can be done via performance evaluation of each trained model on a labeled hold-out set, either by simply searching among models defined over a static grid or via more sophisticated dynamic search techniques such as bandit-based strategies [168] and Bayesian hyperparameter optimization [120, 286] – the latter of which are effectively used in the AutoML literature [111]. We remark that these techniques cannot (at least directly) be used in UOMS as they require model evaluation using ground-truth labels. There exist some recent work on automating OD by Li *et al.* [172, 173], which however also relies on labels, inheriting the aforementioned limitations. Meta-learning has also been employed [82, 315], e.g., to find promising initialization for (i.e. warm-starting) BO [83, 312]. Note that *all of these approaches rely on multiple model evaluations* (i.e., performance queries) for various HCs, and hence cannot be applied to the *unsupervised* outlier model selection problem.

Unsupervised Model Selection: Unsupervised ML tasks (e.g., clustering) poses additional challenges for model selection [80, 290]. Nonetheless, those exhibit *established objective criteria* that enable model comparison, *unlike* OD. For example, BO methods still apply where the surrogate can be trained on ⟨HC, objective value⟩ pairs, for which meta-learning can provide favorable priors.

Task-independent meta-learning [3], that simply identifies the globally best model on historical tasks, applies to the unsupervised setting and hence OD. This can be refined by identifying the best

model on not all, but *similar* tasks, where task similarity is measured in the meta-feature space via clustering [132] or nearest neighbors [217]. This type of similarity-based recommendations points to a connection between algorithm selection and collaborative filtering (CF), first recognized by Stern *et al.* [282]. The most related to UOMS is CF under cold start, where evaluations are not-available (in our case, infeasible) for a new user (in our case, task). There have been a number of work using meta-learning for the cold-start recommendation problem [38, 160, 295], and vice versa, using CF solutions for ML algorithm selection [206, 328].

There exists a small body of work, specifically addressing UOMS, that proposes *internal* (i.e. unsupervised) model evaluation strategies to assess the quality of a model and its output. These are called internal strategies as they use heuristic measures that solely make use of the input data and/or output outlier scores. To our knowledge, there are only three such techniques (in chronological order) [197, 91, 216]. However, they employ their proposed strategies to select only among 2-3 detectors on 8-12 real-world datasets. More problematically, they do not systematically compare to one another, nor do they use the same datasets. This makes it difficult to fully understand the strengths and limitations of these existing methods, and ultimately the extent to which progress has been made on this subject. In Chapter 1, we perform a comprehensive survey on internal evaluation strategies, especially their usage in UOMS.

0.2.3 Definition of UOMS

Here we provide a more formal definition of UOMS as follows. Model selection concerns picking a model from a pool of candidate models. Let $\mathcal{M} = \{M_i\}_{i=1}^{N}$ denote a pre-specified pool of N models. Here each model M_i is a $\{$`detector`, `HPconfiguration`$\}$ pair, where `detector` is a certain outlier detection algorithm (e.g. Local Outlier Factor (LOF) [46]) and `HPconfiguration` is a certain setting of the values for its hyperparameter(s) (e.g. for LOF, value of `n_neighbors`: number of nearest neighbors to consider, and function of choice for `distance` computation). As such, $\{$`LOF`, `k=5`$\}$

Table 1: Outlier Detection Models; see hyperparameter definitions from PyOD

Detection algorithm	Hyperparameter 1	Hyperparameter 2	Total
LOF [46]	n_neighbors: [1, 5, 10, 15, 20, 25, 50, 60, 70, 80, 90, 100]	distance: ['manhattan', 'euclidean', 'minkowski']	36
kNN [243]	n_neighbors: [1, 5, 10, 15, 20, 25, 50, 60, 70, 80, 90, 100]	method: ['largest', 'mean', 'median']	36
OCSVM [262]	nu (train error tol): [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]	kernel: ['linear', 'poly', 'rbf', 'sigmoid']	36
COF [285]	n_neighbors: [3, 5, 10, 15, 20, 25, 50]	N/A	7
ABOD [145]	n_neighbors: [3, 5, 10, 15, 20, 25, 50]	N/A	7
iForest [180]	n_estimators: [10, 20, 30, 40, 50, 75, 100, 150, 200]	max_features: [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]	81
HBOS [92]	n_histograms: [5, 10, 20, 30, 40, 50, 75, 100]	tolerance: [0.1, 0.2, 0.3, 0.4, 0.5]	40
LODA [236]	n_bins: [10, 20, 30, 40, 50, 75, 100, 150, 200]	n_random_cuts: [5, 10, 15, 20, 25, 30]	54
			297

and {`LOF`, `k=10`} are examples to two different models in a pool.

In this thesis, $\mathcal{M}$ is composed by pairing 8 popular outlier detection algorithms to distinct hyperparameter choices, comprising a total of $N = 297$ models, as listed in Table 1. All models are trained based on the Python Outlier Detection Toolbox (PyOD) [347] on each dataset.

Let $\mathcal{D} = \{D_t\}_{t=1}^{T}$ denote the set of outlier detection datasets (i.e. tasks), where $D_t = \{\mathbf{x}_j^{(t)}\}_{j=1}^{n_t}$, $n_t = |D_t|$ is the number of samples and o_t is the true number of ground-truth outliers in D_t. We denote by $\mathbf{s}_i^{(t)} \in \mathbb{R}^{n_t}$ the list of outlier scores output by model M_i when employed (i.e. trained[†]) on D_t, and $s_{ij}^{(t)} \in \mathbb{R}$ to depict individual sample j's score. We omit the superscript when it is clear from context. W.l.o.g. the higher the s_{ij} is, the more anomalous is j w.r.t. M_i.

Problem 1 (Unsupervised Outlier Model Selection) (UOMS) *The model selection problem for unsupervised outlier detection can be stated as follows.*

Given an unsupervised detection task $D = \{\mathbf{x}_j\}_{j=1}^{n}$,
all models in $\mathcal{M}$ trained on D
with corresponding output scores $\{\mathbf{s}_i\}_{i=1}^{N}$;
Select a model $M' \in \mathcal{M}$,
such that $\mathbf{s}'$ yields good detection performance.

Note that the detection performance is to be quantified *post* model selection, where ground-truth

[†]Note that as we consider unsupervised outlier detection, model "training" does not involve any ground-truth labels.

labels are used only for evaluation (and **not** for model training or model selection).

Also note that not all the models in $\mathcal{M}$ should be trained to select a good one. This effort can be reduced by techniques like meta-learning, which we further elaborate this in Chapter §2 and 3.

0.2.4 Our Proposals for UOMS

In Chapter 1, we perform a comprehensive survey on internal evaluation strategies, especially their usage in UOMS. More recently, we have designed a series of new approaches for UOMS leveraging two key concepts; In Chapter 2 and 3)[348, 342, 349], we introduce new UOMS approaches using meta-learning. The idea is to boost the relatively weak internal model performance signals from these heuristic measures via meta-learning from historical tasks that have labels. As such, any future work on designing new internal model evaluation strategies and improving their effectiveness and speed would directly feed into and advantage these meta-learning based UOMS approaches.

0.3 Automation Challenge 2: Unsupervised OD Hyperparameter Optimization (HPO)

0.3.1 Importance of HPO for OD

Although a long list of unsupervised outlier detection (OD) algorithms have been proposed in the last few decades [10, 51, 228], how to optimize their hyperparameter(s) (HP) remains underexplored. Without hyperparameter optimization (HPO) methods, practitioners often use the default HP of an OD algorithm, which is hardly optimal given many OD algorithms are sensitive to HPs. For example, a recent study by Zhao *et al.* reports that by varying the number of nearest neighbors in local outlier factor (LOF) [46] while fixing other conditions, up to 10× performance difference is observed in some datasets [348]. The literature also shows that HP sensitivity is exacerbated for deep OD models with more 'knobs' (e.g., HPs and architectures) [71], which we also observe in this

study—deep robust autoencoder (RAE) [356] exhibits up to 37× performance variation under different HPs.

In supervised learning, one can use ground truth labels to evaluate the performance of an HP, including simple grid and random search [34] as well as more efficient Sequential Model-based Bayesian Optimization (SMBO) [131]. Unlike the simple methods, SMBO builds a cheap regression model (called the surrogate function) of the expensive objective function (which often requires ground truth labels), and uses it to iteratively select the next promising HP for the objective function to evaluate. Notably, learning-based SMBO is more efficient and effective than simple, non-learnable methods [78].

However, unsupervised OD algorithms face evaluation challenges—they do not have access to (external) ground truth labels, and most of them (e.g., LOF and Isolation Forest (iForest) [180]) do not have an (internal) objective function to guide the learning either. Even for the OD algorithms with an internal objective (e.g., reconstruction loss in RAE), its value does not necessarily correlate with the actual detection performance [71]. Thus, HPO for unsupervised OD is challenging and underexplored.

0.3.2 Definition of HPO for OD

We consider the hyperparameter optimization (HPO) problem for unsupervised outlier detection (OD), referred to as *HPO for OD* hereafter. Given a new dataset $\mathbf{D}_{\text{test}} = (\mathbf{X}_{\text{test}}, \emptyset)$ *without any labels* and an OD algorithm M with the HP space Λ, the goal is to identify a HP setting $\lambda \in \Lambda$ so that model M_λ (i.e., detector M with HP λ) achieves the highest performance. HPs can be discrete and continuous, leading to an infinite number of candidate HP configurations. For instance, given h hyperparameters $\lambda_1 \ldots \lambda_h$, with domains $\Lambda_1, \ldots, \Lambda_h$, the hyperparameter space Λ of M is a subset of the cross product of these domains: $\Lambda \subset \Lambda_1 \times \cdots \times \Lambda_h$. Eq. (1) presents the goal formally.

$$\arg\max_{\lambda \in \Lambda} \quad \text{perf}(M_\lambda, \mathbf{X}_{\text{test}}) \tag{1}$$

Problem 2 (HPO for OD) Given *a new input dataset (i.e., detection task*[‡]*)* $\mathcal{D}_{test} = (\mathbf{X}_{test}, \emptyset)$ *without any labels,* pick *a hyperparameter setting* $\lambda \in \Lambda$ *for a given detection algorithm* M *to employ on* $\mathbf{X}_{test}$ *to maximize its performance.*

It is infeasible to evaluate an infinite number of configurations with continuous HP domain(s), and thus a key challenge is efficiently searching the space. As *HPO for OD* does not have access to ground truth labels $\mathbf{y}_{\text{test}}$, HP performance (perf.) cannot be evaluated directly.

0.3.3 Our Proposal for HPO for OD

In Chapter 4. we present the first continuous HPO framework for unsupervised OD termed HPOD. The core idea is to use meta-learning to transfer useful knowledge for optimizing HPs on a new dataset without using any labels.

0.4 System Challenge 1: Accelerating Heterogeneous Outlier Ensembles

Most of the existing OD algorithms are unsupervised due to the high cost of acquiring ground truth. Model selection and hyperparameter tuning are feasible (see Chapter 1-4), while still being gradually adapted by the industry. To reduce the risk and instability of using a single OD model, practitioners prefer to build a large corpus of OD models with variation and diversity [9], e.g., different algorithms, varying parameters, distinct views of the datasets, etc.

This approach is known as *heterogeneous* ***outlier ensembles***. Ensemble methods that select and combine diversified base models can be leveraged to analyze heterogeneous OD models [7, 359,

[‡]Throughout text, we use *task* and *dataset* interchangeably.

9], and more reliable results may be achieved. The most straightforward combination is to take the average or maximum across all the base models as the final result [9], along with more complex combination approaches in both unsupervised [346] and semi-supervised manners [344].

However, training and prediction with many heterogeneous OD models are computationally expensive on high-dimensional, large datasets. For instance, proximity-based algorithms, assuming outliers behave differently in specific regions [7], can be prohibitively slow or fail to work under this setting. Representative methods such as k nearest neighbors (kNN) [243], local outlier factor (LOF) [46], and local outlier probabilities (LoOP) [143], operate in Euclidean space for distance/density calculation, suffering from the curse of dimensionality [261]. Numerous works have attempted to tackle this scalability challenge from various angles, e.g., data projection [136], subspacing [180], and distributed learning for specific OD algorithms [188, 218]. However, none provides a comprehensive solution considering all aspects of large-scale heterogeneous OD, leading to limited practicability and efficacy.

Chapter 6 presents SUOD, a specialized CPU system for accelerating heterogeneous OD ensembles via designs in data, model, and execution.

0.5 System Challenge 2: Accelerating OD Algorithms with Modern Accelerators

Numerous OD algorithms have been proposed recently to detect outliers for different types of data (e.g., tabular data [7, 175, 346], time series [150], and graphs [182]). Although there is no shortage of detection algorithms, OD applications face challenges in *scaling to large datasets*, both in terms of execution time and memory consumption, which prevents OD algorithms from being deployed in data-intensive and/or time-critical tasks such as real-time credit card fraud detection. To address these challenges, recent work focuses on both developing distributed OD algorithms on

CPUs [188, 220, 24, 37, 218, 327, 288, 345, 338] and accelerating certain OD algorithms on GPUs [25, 157]. However, existing GPU-based OD solutions only target specific (families of) OD algorithms and cannot support *generic* OD computations. For instance, Angiulli et al. [25] showcases an example of using GPUs for distance-based algorithms, while how to handle linear and probabilistic OD algorithms remains unclear in the proposed solution.

Chapter 7 presents TOD, the first multi-GPU distributed system for diverse OD algorithms. A key idea behind TOD is decomposing complex OD applications into a small collection of basic tensor algebra operators. This decomposition enables TOD to accelerate OD computations by leveraging GPUs for fast tensor computation.

0.6 Thesis Organization and Summary of Contributions

0.6.1 Overall Structure and Included Papers

The thesis is split into three parts. In Part I of automation, I present a set of works on selecting a good OD model as well as tuning its hyperparameters *without* using labels, which are crucial for real-world deployment. In Part II of systems, I discuss critical OD systems that support scalable learning tasks, which have been used in thousands of applications with millions of downloads. In Part III of applications, I present benchmarks and applications of OD. Finally, I conclude the thesis by providing promising future directions for outlier detection. A more detailed breakdown can be found below.

Automation (Part I):

1. **Chapter 1** introduces and analyzes a set of unsupervised (internal) evaluation strategies for outlier detection, where we also formally define the unsupervised outlier model selection (UOMS) problem. This chapter is primarily based on:

[190] Ma, Martin Q.*, **Yue Zhao***, Xiaorong Zhang, and Leman Akoglu. "The Need for Unsupervised Outlier Model Selection: A Review and Evaluation of Internal Evaluation Strategies." *ACM SIGKDD Explorations Newsletter*, 2023. (*Co-first Author; equal contribution)

2. **Chapter 2** proposes the first systematic solution for UOMS by meta-learning, called MetaOD. This chapter is primarily based on:
   [348] **Yue Zhao**, Ryan Rossi, and Leman Akoglu. "Automatic Unsupervised Outlier Model Selection." *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

3. **Chapter 3** introduces another UOMS solution by meta-learning. As a follow-up work on UOMS, the proposed ELECT achieves better empirical performance. This chapter is primarily based on:
   [349] **Yue Zhao***, Sean Zhang, Leman Akoglu. "ELECT: Toward Unsupervised Outlier Model Selection". *IEEE International Conference on Data Mining (ICDM)*, 2022.

4. **Chapter 4** tackles another important problem in OD automation—hyperparameter optimization for OD algorithms. As the first of its kind, the proposed HPOD provides a meta-learning solution for this important problem. This chapter is primarily based on:
   [342] **Yue Zhao***, Leman Akoglu. "Towards unsupervised hpo for outlier detection." *arXiv preprint arXiv:2208.11727*, 2023.

Systems (Part II):

1. **Chapter 5** describes a comprehensive and scalable detection system called Python Outlier Detection (PyOD) library, which has been widely used in thousands of applications. This chapter is primarily based on:

[347] **Yue Zhao***, Zain Nasrullah, and Zheng Li. "PyOD: A Python Toolbox for Scalable Outlier Detection." *Journal of Machine Learning Research (JMLR)*, 2019.

2. **Chapter 6** introduces an acceleration system on top of PyOD for training a group of heterogeneous OD models. The proposed SUOD system leverages {data, model, system} level technique to make OD faster. This chapter is primarily based on:
   [345] **Yue Zhao***, Xiyang Hu*, Cheng Cheng, Cong Wang, Changlin Wan, Wen Wang, Jianing Yang, Haoping Bai, Zheng Li, Cao Xiao, Yunlong Wang, Zhi Qiao, Jimeng Sun, and Leman Akoglu. "SUOD: Accelerating Large-scale Unsupervised Heterogeneous Outlier Detection." *Conference on Machine Learning and Systems (MLSys)*, 2021.

3. **Chapter 7** designs the first distributed multi-GPU detection system, called TOD. In this work, we introduce a novel tensor programming interface to make diverse OD algorithms available for GPU acceleration, as well as come up with new techniques for deeper operator optimization. This chapter is primarily based on:
   [343] **Yue Zhao***, George H. Chen, Zhihao Jia. "TOD: GPU-accelerated Outlier Detection via Tensor Operations." *International Conference on Very Large Data Bases (VLDB)*, 2023.

Benchmarks and Applications (Part III):

1. **Chapter 8** describes ADBench, the most comprehensive benchmark for anomaly detection. Notably, ADBench's design is driven by real-world applications, which consider the detection performance with regard to (i) varying levels of supervision, (ii) different types of anomalies, and (iii) noisy and corrupted data. This chapter is primarily based on:
   [105] Songqiao Han*, Xiyang Hu*, Hailiang Huang*, Minqi Jiang*, and **Yue Zhao***†. "ADBench: Anomaly Detection Benchmark." *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. (*Co-first Author; †Corresponding Author)

2. **Chapter 9** introduces a novel learning paradigm called ADMoE, which is motivated by industry settings with access to multiple sets of noisy labels. This work has already been used for email policy rule manipulation at a major email client vendor. This chapter is primarily based on:

   [350] **Yue Zhao**, Guoqing Zheng, Subhabrata Mukherjee, Robert McCann, and Ahmed Awadallah. "ADMoE: Anomaly Detection with Mixture-of-experts from Noisy Labels." *AAAI Conference on Artificial Intelligence (AAAI)*, 2023.

In addition to the above chapters, I also co-authored the following papers during my Ph.D. study (which are not included in this thesis).

Anomaly and Outlier Detection:

[129] Minqi Jiang*, Chaochuan Hou*, Ao Zheng*, Xiyang Hu*, Songqiao Han, Hailiang Huang, Xiangnan He, Philip S. Yu and **Yue Zhao**†. "Weakly Supervised Anomaly Detection: A Survey" *arXiv preprint arXiv:2302.04549*, 2023. (*Co-first Author; †Corresponding Author)

[175] **Yue Zhao***, Zheng Li*, Xiyang Hu, Nicola Botta, Cezar Ionescu, George H. Chen. "ECOD: Unsupervised Outlier Detection Using Empirical Cumulative Distribution Functions." IEEE Transactions on Knowledge and Data Engineering (TKDE), 2022. (*Co-first Author; equal contribution)

[182] Kay Liu*, Yingtong Dou*, **Yue Zhao***, Xueying Ding, Xiyang Hu, Ruitong Zhang, Kaize Ding et al. "BOND: Benchmarking Unsupervised Outlier Node Detection on Static Attributed Graphs." *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. (*Co-first Author; equal contribution)

[150] Kwei-Herng Lai, Daochen Zha, Junjie Xu, **Yue Zhao**, Guanchu Wang, and Xia Hu."Revisiting time series outlier detection: Definitions and benchmarks." *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[174] Zheng Li, **Yue Zhao**, Nicola Botta, Cezar Ionescu, and Xiyang Hu (2020). "COPOD: Copula-Based Outlier Detection." IEEE International Conference on Data Mining (ICDM), 2020.

AI for Science:

[119] Kexin Huang, Tianfan Fu, Wenhao Gao, **Yue Zhao**, Yusuf Roohani, Jure Leskovec, Connor W. Coley, Cao Xiao, Jimeng Sun, and Marinka Zitnik. "Artificial intelligence foundation for therapeutic science." Nature Chemical Biology, 2022.
[118]Kexin Huang, Tianfan Fu, Wenhao Gao, **Yue Zhao**, Yusuf Roohani, Jure Leskovec, Connor W. Coley, Cao Xiao, Jimeng Sun, and Marinka Zitnik. "Therapeutics Data Commons: Machine Learning Datasets and Tasks for Drug Discovery and Development." *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

0.7 Associated Codes and Datasets

Table 2 provides the links to all the associated codes and datasets.

Table 2: Summary of codes and datasets.

Chapter	Method	Reference	Repo
Chapter 1	IPM	[190]	https://github.com/yzhao062/UOMS
Chapter 2	MetaOD	[348]	https://github.com/yzhao062/metaod
Chapter 3	ELECT	[349]	https://github.com/yzhao062/ELECT
Chapter 4	HPOD	[342]	https://github.com/yzhao062/hpod
Chapter 5	PyOD	[347]	https://github.com/yzhao062/pyod
Chapter 6	SUOD	[345]	https://github.com/yzhao062/suod
Chapter 7	TOD	[343]	https://github.com/yzhao062/pytod
Chapter 8	ADBench	[105]	https://github.com/Minqi824/ADBench
Chapter 9	ADMoE	[350]	https://github.com/microsoft/admoe

Part I

Automation

1

UOMS via Internal Evaluation Strategies

This chapter is primarily based on:

Ma, Martin Q.*, **Yue Zhao***, Xiaorong Zhang, and Leman Akoglu. "The Need for Unsupervised Outlier Model Selection: A Review and Evaluation of Internal Evaluation Strategies." *ACM SIGKDD Explorations Newsletter*, 2023. (*Co-first Author; equal contribution)

1.1 Highlight

This chapter focuses on *internal model evaluation strategies* for selecting a model for UOMS (see problem definition in §0.2.3). These so-called internal strategies solely rely on the input data (without labels) and the output (outlier scores) of the candidate models.

Our goal is to survey internal model evaluation strategies, and systematically evaluate and compare their effectiveness. To this end, we first bring under one umbrella the aforementioned three existing UOMS methods, adapt two state-of-the-art unsupervised model selection techniques originally proposed for deep representation learning [75, 177], and design two new internal model selection methods inspired by various consensus algorithms.

To compare their effectiveness systematically, we construct a large testbed of $T = 39$ real-world outlier detection datasets from two different repositories (See Sec. 1.4.1). That is, we perform UOMS using each technique 39 times, to select one model from the pool of 297. Given that the datasets are independent, a large testbed enables paired statistical tests that conclusively identify significant differences between these techniques and various simple baselines. We put them to test on a large testbed against simple baselines, including random model selection as well as the popular isolation Forest (iForest) [180], with default HPs. To our knowledge, this is the first work to systematically review and evaluate the internal evaluation strategies toward UOMS.

We summarize the contributions and findings of this chapter as follows.

- **Unified Comparison:** We identify (to our knowledge) all existing internal model evaluation strategies for UOMS. For the first time, we systematically compare them on their ability to discriminate between models w.r.t. detection performance, as well as w.r.t. running time, on the same testbed.

- **Large-scale Evaluation:** Our testbed consists of 8 state-of-the-art detectors, each configured

by a comprehensive list of hyperparameter settings, yielding a candidate pool of 297 models. We perform the model selection task on 39 independent real-world datasets from two different public repositories. We compare different strategies through paired statistical tests to identify significant differences, if any. We find that all three existing strategies specifically designed for UOMS are ill-suited. Alarmingly, none of them is significantly different from random selection (!)

- **New UOMS Techniques:** All three existing methods specifically designed for UOMS are *stand-alone*; evaluating each model individually, independent of others. In addition to those, we repurpose four *consensus-based* algorithms from other areas for UOMS; utilizing the agreements among the models in the pool. We find that consensus-based methods are more competitive than stand-alone ones, and all of them achieve significantly better performance than random. However, they are not different from iForest (the best detector in our pool), thus, would not be employed (on a pool) over training a single (iForest) model.

1.2 Review of Internal Model Evaluation Strategies

Internal strategies evaluate the goodness of a model without using any external information, especially with no access to ground-truth labels. The internal information being used is solely limited to (*i*) the input samples (feature values only), (*ii*) the trained models in the candidate pool and the outlier scores as output by these trained models. The common thread among all internal model evaluation strategies in this study is an estimated heuristic *internal measure* of "model goodness". Model selection is then addressed by top-1 selection: i.e. picking the model with the highest value of the respective measure.

We categorize the 7 strategies we studied into two, depending on how they estimate their internal measure: (1) **stand-alone** and (2) **consensus-based**. (See Table 1.1.) Stand-alone strategies solely

rely on each model and its output individually, independent of other models. All three existing methods proposed specifically for UOMS fall into this category. On the other hand, consensus-based strategies leverage agreement between the models in the pool and hence utilize candidate models collectively. Four strategies we adopt and adapt* from other areas all fall into this latter category.

In the following, we provide a short description of each strategy (and refer to the original articles for full details). We also remark on the computational complexity of some methods as they demand considerable running time. The ideal is to have a lightweight and effective selection method with low overhead incurred on top of model training. In the experiments, we compare these methods w.r.t. their selection performance as well as running time.

1.2.1 Stand-alone Internal Evaluation (Existing)

1.2.1.1 IREOS

The first known index proposed for the internal evaluation of outlier detection results is by Marques et al., called Internal, Relative Evaluation of Outlier Solutions (IREOS) [197]. While their initial index is designed only for binary solutions (referred to as "top-n" detection), their recent work [196] generalized to numeric outlier scorings, which is the setting considered in this study.

Their intuition is that an outlier should be more easily separated (discriminated) from other samples than an inlier. Then, a model is "good" the more it identifies as outlier those samples with a large degree of separability. They propose to assess the separability of each individual sample using a maximum-margin classifier (and specifically use nonlinear SVMs).† The IREOS score of a model M_i on a given dataset is computed as

*We *adopt* two strategies originally proposed for unsupervised model selection for deep representation learning "as is", and *adapt* two techniques (from information retrieval and ensemble learning) by repurposing them to UOMS problem with small modifications.

†Note that collective outliers (forming micro-clusters, or clumps) do not have high separability. IREOS accounts for this effectively, provided a user-specified `clump_size`. For details, we refer to the original articles.

Table 1.1: Overview of UOMS methods studied in this survey.

Method	Type	Based on	Strategy
XB,RS,... [216]	Stand-alone	Outlier scores	Cluster quality
EM, MV [91]	Stand-alone	Outlier scores	Level sets
IREOS [197]	Stand-alone	O. scores + Input	Separability
UDR [75]	Consensus	Outlier scores	One-shot
MC [177], MC_S	Consensus	Outlier scores	One-shot
HITS [139]	Consensus	Outlier scores	Iterative
ENS [359]	Consensus	Outlier scores	Iterative

$$\text{IREOS}(\mathbf{s}_i) = \frac{1}{n_\gamma} \sum_{l=1}^{n_\gamma} \frac{\sum_{j=1}^{n} p(\mathbf{x}_j, \gamma_l) w_{ij}}{\sum_{j=1}^{n} w_{ij}} \tag{1.1}$$

where $p(\mathbf{x}_j, \gamma_l)$ is the separability of sample j as estimated by a nonlinear SVM with kernel bandwidth (a hyper-parameter) γ_l, and n_γ is the number of different bandwidth values used from the interval $[0, \gamma_{\max}]$.[‡] They convert outlier scores $\{s_{ij}\}_{j=1}^{n}$ to probability weights $\{w_{ij}\}_{j=1}^{n}$ using the approach by [144] to push inlier scores toward zero so that they do not in aggregate dominate the weighted sum. IREOS tends to give high scores to those models whose outlier scores correlate well with the separability scores by a nonlinear SVM.

Computationally, IREOS is quite demanding as it requires training of a nonlinear classifier *per sample*. Their source code provides ways to approximate IREOS scores, mainly estimating separability via nearest neighbor distances, which however are also expensive to compute.

1.2.1.2 Mass-Volume (MV) and Excess-Mass (EM)

Goix [91] proposed using statistical tools, namely MV and EM curves, to measure the quality of a scoring function. Formally, a scoring function $s : \mathbb{R}^d \mapsto \mathbb{R}_+$ is any measurable function integrable w.r.t. the Lebesgue measure $\text{Leb}(\cdot)$, whose level sets are estimates of the level sets of the density.

[‡]They use heuristics to automatically set $\gamma_{\max}$ in code.

Outliers are assumed to occur in the tail of the score distribution as produced by a scoring function, where the *lower* $s(\mathbf{x})$ is, the more abnormal is $\mathbf{x}$.

Given a scoring function $s(\cdot)$ (in our context, an outlier model), the MV measure is defined as follows.

$$\widehat{MV}_s(\alpha) = \inf_{u \geq 0} \text{Leb}(s \geq u) \; s.t. \; \mathbb{P}_n(s(\mathbf{X}) \geq u) \geq \alpha \tag{1.2}$$

where $\alpha \in (0, 1)$, and $\mathbb{P}_n$ is the empirical distribution; $\mathbb{P}_n(s \geq v) = \frac{1}{n} \sum_{j=1}^{n} 1_{s(\mathbf{x}_j) > v}$.

For univariate real numbers, $\text{Leb}(\cdot)$ measures the length of the given interval. Let $s_{\max}$ denote the largest score produced by $s(\cdot)$. Then, empirically $\text{Leb}(s \geq u)$ is equal to the length $|s_{\max} - u|$. Given α, the u that minimizes the Lebesgue measure $\text{Leb}(s \geq u)$ in Eq. (1.2) would be equal to the outlier score at the $(1-\alpha)$-*th* quantile, i.e. $u = CCDF_s^{-1}(\alpha)$. Then, $|s_{\max} - u|$ would give the length of the range of scores for α fraction of the samples with scores larger than u. In their work, they consider $\alpha \in (0.9, 0.999)$.[§,¶] As they assume a *lower* score is more anomalous, the Lebesgue measure quantifies the length of the interval of scores for the inliers. The smaller MV is, the better the scoring function is deemed to be. Intuitively, then, MV measures the clusteredness of inlier scores (or the compactness of high-density level sets).

The EM measure is quite similar, and is defined as

$$\widehat{EM}_s(t) = \sup_{u \geq 0} \; \mathbb{P}_n(s(\mathbf{X}) \geq u) - t\text{Leb}(s \geq u) \tag{1.3}$$

for $t > 0$. Similarly, they consider $t \in [0, \widehat{EM}_s^{-1}(0.9)]$ with $\widehat{EM}_s^{-1}(0.9) := \inf\{t \geq 0, \widehat{EM}_s(t) \leq 0.9\}$.

Intuitively, EM would identify as small a u value as possible (so as to maximize the density mass

[§]Given fraction of outliers is bounded to 10% maximum.

[¶]Area under the MV-curve is estimated as the sum of empirical MV values by Eq. (1.2) for discretized values of α.

in the first term) such that the scores larger than or equal to u are as clustered as possible (so as to minimize the Lebesgue measure in the second term). The more clustered are the scores of the bulk of the samples (i.e. inliers), the larger EM gets, and the better the scoring function is deemed to be.

1.2.1.3 Clustering validation metrics

[216] point out that a drawback of IREOS, besides computational demand, is its dependence on classification – which itself introduces a model selection problem – since the results may depend on the selected classification algorithm and its hyper-parameter settings. Another paper [215] by the same authors proposed a classification based internal evaluation method, similar to IREOS. Their experiments show that the current internal measures do comparably well or better with less computational overhead, hence we omit [215] from this study. Despite citing IREOS, they do not compare in experiments.

Their key proposal is to apply internal validation measures for clustering algorithms to outlier detection. As clustering aims to ensure samples within each cluster are similar and different from samples in other clusters, these measures are mainly based on compactness (capturing within-cluster similarity) and/or separation (reflecting inter-cluster distance).

To that end, we split the outlier scores by a given model under evaluation for dataset D_t into two clusters, denoted C_o and C_i, respectively consisting of the highest o_t scores and the rest. According to those measures, an outlier model is "good" the more separated these two sets of scores are and/or the more clustered the scores within each set are.

In their study, they compared 10 different existing clustering quality measures, such as the Silhouette index [252], Xie-Beni index [323], etc. (See others in the original article.) One of the well-performing ones in our experiments, namely Xie-Beni index of a model M_i, denoted XB_i, is defined

as follows.

$$\text{XB}_i = \frac{\sum_{j \in C_o} d^2(s_{ij}, c_o) + \sum_{j' \in C_i} d^2(s_{ij'}, c_i)}{n_t \, d^2(c_o, c_i)} \tag{1.4}$$

where $c_o = \sum_{j \in C_o} s_{ij}/o_t$ and $c_i = \sum_{j' \in C_i} s_{ij'}/(n_t - o_t)$ depict the cluster centers and $d(\cdot, \cdot)$ is the Euclidean distance. This index can be interpreted as the ratio of the intra-cluster compactness to the inter-cluster separation.

Clustering quality based measures are typically easy to compute; most of them being linear in the number of samples.

1.2.2 Consensus-based Internal Evaluation (Repurposed)

1.2.2.1 UDR

The first consensus-based approach, namely Unsupervised Disentanglement Ranking (UDR), is adopted from deep learning and is "the first method for unsupervised model selection for variational disentangled representation learning" [75]. Each model in their case corresponds to a {`HPconfiguration`, `seed`} pair. Reciting Tolstoy who wrote "Happy families are all alike; every unhappy family is unhappy in its own way.", their main hypothesis is that a model with a good hyper-parameter (HP) setting will produce *similar results* under different random initializations (i.e. seeds) whereas for a poor HP setting, results based on different random seeds will look arbitrarily different.

In a nutshell, UDR follows 4 steps: (1) Train $N = H \times S$ models, where H and S are the number of hyperparameter settings and random seeds, respectively. (2) For each model M_i, randomly sample (without replacement) $P \leq S$ other models with the *same* HP as M_i, but *different* seeds. (3) Perform P pairwise comparisons between M_i and the models sampled in Step 2 for M_i. (4) Aggregate pairwise similarity scores (denoted $UDR_{ii'}$) as $UDR_i = \text{median}_{i'} \; UDR_{ii'}$, for $i = 1, \ldots, N$. Finally, they pick the model (among N) with the largest UDR_i. Intuitively, UDR selects a model with an HP setting that yields stable or *consistent* results across various seeds.

Notice that adopting UDR for the UOMS task is trivial by making the analogy between {`HPconfiguration`, `seed`} and {`detector`, `HPconfiguration`}. While trivially applied, one may question whether the implied hypothesis (that a good detector has consistent results across different HP settings) holds true for outlier models, since one of the key reasons for UOMS in the first place is that most detectors are sensitive to their HP settings [93].

The key part of UDR is how pairwise model comparisons are done in Step 3. We measure the output *ranking similarity* of the samples by two models, based on three well-known measures from information retrieval [178] (See Sec. 1.4.1).

1.2.2.2 MC

A follow-up work to UDR proposed ModelCentrality (MC), which is another consensus-based strategy for what they call "self-supervised" model selection for disentangling GANs [177].

Their premise is similar, that "well-disentangled models should be close to the optimal model, and hence also close to each other". Provided the similarity $B_{ii'}$ between two models M_i and $M_{i'}$ can be computed, ModelCentrality of M_i is written as $MC_i = \frac{1}{N-1} \sum_{i' \neq i} B_{ii'}$. They then select the model with the largest MC_i, which coincides with the medoid in the pool of models – hence the name MC.

Computationally, MC is quadratic in the number of models as it requires all pairwise comparisons. We also experiment with a lightweight version, called MC_S, where we randomly sample $P \leq N$ models and compute MC_i of M_i as the average of its similarities to P models, effectively reducing its complexity down to that of UDR.

In their experiments, [177] report that MC outperforms UDR schemes (Sec. 1.2.2.1). Our results are consistent with their finding, possibly because it is an unrealistic hypothesis for outlier models that a good model would have consistent results across HP settings.

1.2.2.3 Model Centrality by HITS

We can build on the idea of ModelCentrality through computing centrality in a network setting. Unlike MC that is computed in one shot, network centrality is *recursive*—wherein a node has higher centrality the more they point to nodes that are pointed by other high-centrality ones.

One of the earliest methods for computing centrality, namely hubness h_p and authority a_p, of pages on the Web is the HITS algorithm [139], where

$$h_p \propto \text{sum of } a_i \text{ for all nodes } i \text{ that } p \text{ points to }, \text{ and}$$
$$a_p \propto \text{sum of } h_i \text{ for all nodes } i \text{ pointing to } p \ ,$$

which are estimated alternatingly over iterations until convergence. Besides ranking on the Web, HITS-like ideas have been used to estimate user trustworthiness in online rating platforms [148, 298], physician authoritativeness in patient referral networks [205], polarity of subjects in political networks [16], as well as truth discovery [331].

It is easy to adapt HITS for UOMS by constructing a complete bipartite network between the N models and n_t samples in a given dataset D_t. Then, the models can be evaluated by their hubness centralities. The analogous interpretation is that a a sample has higher authority (outlierness), the more trusted models (with high hubness) point to it (with large outlierness score, i.e. large edge weight). Then, a model is more central or trusted, the more it points (with large outlierness score) to samples with high authority.

Note that a by-product of this strategy is a consensus-based ranking of the samples based on authority scores (i.e. centrality-based outlierness) upon convergence. We compare this (aggregate) ranking, called HITS-AUTH, against selecting a (single) model by hubness in the experiments.

1.2.2.4 Unsupervised outlier model ensembling

HITS has a built-in advantage that is the iterative refinement of model trustworthiness. Specifically, given the trustworthiness of models, outlier scores can be better estimated by a trustworthiness-weighted aggregation of scores across models. Then, given those refined outlier scores, model trustworthiness can also be better estimated; where the more similar their output is to the updated scores, the more a model is deemed trustworthy.

Here we build on another iterative scheme, originally designed for unsupervised selective outlier model ensembling [246, 359]. The idea is to infer reliable "pseudo ground truth" outlier scores via aggregating the output of a carefully-selected subset of trustworthy models. The ensemble is constructed bottom-up in a greedy iterative fashion (see Alg. 1).

Similar to HITS, the "pseudo ground truth" and model trustworthiness are estimated alternatingly. The latter is computed as the ranking based similarity of a model's output to the "pseudo ground truth" (i.e. *target* in Alg. 1) at a given iteration. We adapt this framework to UOMS by using these similarities at convergence to evaluate the models. We call this strategy Ens. In experiments, we also compare the (aggregate) ranking by the ensemble (based on *target*), called Ens-pseudo, to selecting a (single) model (with highest similarity to *target*).

To wrap up, we give a summary of the 7 families of UOMS techniques as described in this section in Table 1.1.

1.3 Critique and Comparison of Existing Techniques

Internal model evaluation strategies for UOMS: Cluster quality based measures [216] and statistical mass based EM/MV methods [91] rely only on output scores. In contrast IREOS [196, 197] uses more information, that is both outlier scores and the original input samples (See Eq. (1.1)). Verifying that outlier scores align (correlate) with the separability of samples in the feature space is

Algorithm 1 Ensemble-based Internal Model Evaluation

Require: set of outlier scores from all models, $\{\mathbf{s}_i\}_{i=1}^{N}$
Ensure: internal scores for all models

$\mathcal{S} := \emptyset$, $\mathcal{E} := \emptyset$, $C := 0$
for $i = 1, \ldots, N$ **do** {convert scores to inverse rank}
    $\mathcal{S} := \mathcal{S} \cup \{1/\text{rank}(s_{ij})\}_{j=1}^{n}$
end for
target := avg($\mathcal{S}$) {initial pseudo ground truth scores}
repeat
    sort $\mathcal{S}$ by rank *corr*elation to *target* in desc. order
    $\{m, \text{corr}_m\} :=$ fetchFirst($\mathcal{S}$)
    if *corr*(avg($\mathcal{E} \cup m$), *target*) $\times$ $|E| \geq C$ **then**
        $\mathcal{E} := \mathcal{E} \cup m$, C+= corr_m
        target := avg($\mathcal{E}$) {pseudo ground truth by $\mathcal{E}$}
    end if
until {$\mathcal{S} = \emptyset$ or $\mathcal{E}$ is not updated}
return rank *corr*elation of $\mathbf{s}_i$ to *target*, $i = 1, \ldots, N$

potentially less error-prone than simply looking at whether outlier/inlier scores are well clustered or separated – e.g., a model that outputs a $\{0, 1\}$ score per point at random would be considered a good model by the latter. The trade-off is the computational overhead for quantifying separability per sample.

In their work, IREOS is employed for UOMS using only 2 detectors (LOF [46] and kNN [243]), each with 17 different HP configurations (for a total of 34 models) on 11 datasets. Being the seminal work, there is no comparison to any other techniques (existing or adapted). [216] acknowledge IREOS and criticize its computational demand, without any comparison. They also do not perform any UOMS in experiments, rather, they study the decay in internal measures as the ground truth ranking is contaminated via random swaps at the top based on 12 datasets. Finally, [91] performs UOMS using only and exactly 3 models (LOF, iForest [180], OCSVM [262]), each with a single (unspecified) HP configuration, on 8 datasets. None of these three compares to any other in their work. Moreover, because the datasets, experimental design, and the model pool specified by each

work is different, it is not possible to do any direct comparison. In this chapter, we do a systematic comparison for the first time, using a much larger testbed (8 detectors, 297 models, 39 datasets) than originally considered by any prior work.

Other internal model evaluation strategies repurposed for UOMS: All three existing methods for UOMS are stand-alone, evaluating a model independently from the others. Having trained all models among which to select from, it is reasonable to take advantage of the similarities/agreement among them. To this end, we have repurposed methods from unsupervised representation learning [75, 177], network centrality [139], and unsupervised ensemble learning [246, 359] all of which are based on the "collective intelligence" of the models in the pool.

As we will show in the experiments, these strategies produce superior outcomes than existing, stand-alone methods. As such, our study motivates future work on consensus-based strategies, and calls for the transfer of prominent ideas from other similar fields, such as truth discovery and crowdsourcing, toward tackling the important problem of UOMS.

1.4 Experiments

1.4.1 Setup

Datasets and Model Pool. We already discussed the real-world datasets and candidate models of this study in Sec. 0.2.3. We build the experiments on **39 widely used outlier detection benchmark dataset**. As shown in Table 1.2, 21 datasets are from the **ODDS Library** [245], and the other 18 datasets are from **DAMI datasets** [51]. The specifications for all N=297 models have been listed in Sec. 0.2.3 Table 1.

Baselines. We compare the model selected by each technique (Sec. 1.2) to two baselines across datasets.

- Random, whose performance is the average of all (297) models per dataset. This is equiva-

Table 1.2: Real-world dataset pool composed by ODDS library (21 datasets) and DAMI library (18 datasets).

	Dataset	Num Pts	Dim	% Outlier
1	annthyroid (ODDS)	7200	6	7.416
2	arrhythmia (ODDS)	452	274	14.601
3	breastw (ODDS)	683	9	34.992
4	glass (ODDS)	214	9	4.205
5	ionosphere (ODDS)	351	33	35.897
6	letter (ODDS)	1600	32	6.250
7	lympho (ODDS)	148	18	4.054
8	mammography (ODDS)	11183	6	2.325
9	mnist (ODDS)	7603	100	9.206
10	musk (ODDS)	3062	166	3.167
11	optdigits (ODDS)	5216	64	2.875
12	pendigits (ODDS)	6870	16	2.270
13	pima (ODDS)	768	8	34.895
14	satellite (ODDS)	6435	36	31.639
15	satimage-2 (ODDS)	5803	36	1.223
16	speech (ODDS)	3686	400	1.654
17	thyroid (ODDS)	3772	6	2.465
18	vertebral (ODDS)	240	6	12.500
19	vowels (ODDS)	1456	12	3.434
20	wbc (ODDS)	378	30	5.555
21	wine (ODDS)	129	13	7.751
22	Annthyroid (DAMI)	7129	21	7.490
23	Arrhythmia (DAMI)	450	259	45.777
24	Cardiotocography (DAMI)	2114	21	22.043
25	HeartDisease (DAMI)	270	13	44.444
26	InternetAds (DAMI)	1966	1555	18.718
27	PageBlocks (DAMI)	5393	10	9.456
28	Pima (DAMI)	768	8	34.895
29	SpamBase (DAMI)	4207	57	39.909
30	Stamps (DAMI)	340	9	9.117
31	Wilt (DAMI)	4819	5	5.333
32	ALOI (DAMI)	49534	27	3.044
33	Glass (DAMI)	214	7	4.205
34	PenDigits (DAMI)	9868	16	0.202
35	Shuttle (DAMI)	1013	9	1.283
36	Waveform (DAMI)	3443	21	2.904
37	WBC (DAMI)	223	9	4.484
38	WDBC (DAMI)	367	30	2.724
39	WPBC (DAMI)	198	33	23.737

lent to expected performance when selecting a model from the candidate pool at random.

- iForest-r, with performance as the average of all (81) iForest models in the pool, equivalent to using iForest [180] (a state-of-the-art ensemble detector) with randomly chosen hyperpa-

rameters.[‖]

Method Configurations. For clustering-quality based measures, we split into two clusters as the top o_t (true number of outliers) and the rest, i.e. give those strategies the advantage of knowing o_t. This avoids the clustering step, which requires us to pick a clustering algorithm etc. and directly focuses on the measures themselves.

For EM and MV[**], we use the default values for α and t respectively (See Sec. 1.2.1.2) and set `n_generated`$= 100K$, which is the number of random samples to generate for estimating the null distributions.

For IREOS, we use the author-recommended settings https://github.com/homarques/ireos-extension.

For UDR, MC, and MC$_S$, we experiment with three different pairwise similarity measures: Spearman's ρ, Kendall's τ, and NDCG [178]. For MC$_S$, $P = \sqrt{N} \approx 18$.

For HITS and Ens, we set edge weights between model M_i and sample j in a dataset as $1/r_{ij}$, where r_{ij} is the position of j in the ranked list by M_i. Raw outlier scores are not used as they are not comparable across models. For comparison between selection versus consensus/ensembling, we also report the performance of the consensus outcome, called HITS-auth and Ens-pseudo; as ranked (resp.) by authority scores and by the pseudo ground truth at convergence.

Performance metrics. We evaluate performance w.r.t. three metrics. Two are based on the ranking quality: Average Precision (**AP**): the area under the precision-recall curve and **ROC** AUC: the area under the recall-false positive rate curve. The third metric measures the quality at the top: **Prec@*k***, precision at top k where we set $k = o_t$ (i.e. true number of outliers) for each $D_t \in \mathcal{D}$. In the Appendix of the original paper [190], we show that performances vary considerably across models for most datasets, justifying the importance of model selection.

[‖]Family-wise performances across datasets in the original paper [190] Appendix show that iForest is the most competitive among the 8 families of detectors in this study, and thus the strongest baseline.

[**]Code available at https://github.com/ngoix/EMMV_benchmarks

For brevity, all results in this section are w.r.t. AP. Corresponding results for other metrics are similar, all of which are provided in the Appendix of [190].

1.4.2 Results

1.4.2.1 Cluster quality based methods

We start by studying the 10 cluster quality based methods to identify those that stand out. We report the p-values by the one-sided paired Wilcoxon signed rank test in Table 1.3. We test the hypothesis: row-method is better than col-method (against the null hypothesis stating no difference). For reverse order, p-value $=$ 1 minus the reported value. STD is significantly worse than all other methods. Three strategies that stand out are RS, CH, and XB, which are identical; in the sense that despite differences in their values and overall ranking, they select exactly the same model on each dataset. Importantly, while both STD and S are significantly worse than RANDOM at $p = 0.05$, none of the others is significantly different from RANDOM (!) All methods (including XB, RS, and CH) are significantly worse than IFOREST-R.

Table 1.3: Comparison of cluster quality based methods and baselines by one-sided paired Wilcoxon signed rank test. p-values **bolded** (underlined) highlight the cases where the "row-method" is significantly **better** (worse) than the "column-method" at $p \leq 0.05$.

	STD	H	S	I	DB	SD	D	RND	IF
XB,RS,CH	**0.004**	0.240	**0.038**	0.212	0.370	0.127	0.357	0.500	0.981
STD		0.997	0.961	0.997	0.982	0.967	0.999	1.000	1.000
H			0.373	0.500	0.725	0.379	0.675	0.849	0.996
S				0.627	0.949	0.557	0.881	0.953	0.999
I					0.730	0.384	0.742	0.882	0.997
DB						0.307	0.647	0.522	0.982
SD							0.823	0.910	0.995
D								0.572	0.990
RND									1.000

These findings suggest that cluster quality based internal evaluation methods would not be useful for UOMS.

1.4.2.2 Other stand-alone methods

As discussed in Sec. 1.2.1.2, EM and MV quantify (roughly) the clusteredness of the inlier scores. Therefore, they are conceptually similar to the clustering quality based methods. Our findings confirm this intuition. As shown in Table 1.4, there is no significant difference between EM/MV and xb/rs/ch or Random. Both of them are also significantly worse than iForest-r. Thus, they do not prove useful for UOMS. Findings are similar for IREOS; despite using more information (input samples besides scores, see Eq. (1.1)) and computational cost, it is only comparable to Random.

Table 1.4: Comparison of stand-alone methods and baselines.

	EM	MV	IREOS	Rnd	iF
xb,rs,ch	0.533	0.500	0.862	0.500	0.981
EM		0.079	0.642	0.539	0.979
MV			0.716	0.687	0.994
IREOS				0.303	0.908

We provide an additional viewpoint by identifying the *q-th best* model per dataset where there exists no significant difference between the performance of the q-th best model and that selected by a given UOMS strategy across datasets. We report the *smallest* q for which one-sided Wilcoxon signed rank test yields $p>0.05$ in Table 1.5. A method with smaller q is better; the interpretation being that it could select, from a pool of 297, the model that is as good as the q-th best model per dataset. Stand-alone methods do not fare well against iForest-r which is comparable to the 84-th best model.

1.4.2.3 Consensus-based methods

We first study one-shot methods UDR, MC, and MC_S based on different similarity measures. As shown in Table 1.5, all versions provide similar results, which are significantly better than Random, and not different from iForest-r. We note that the faster, sampling-based MC_S achieves similar

Table 1.5: Summary of results: p-values by one-sided paired Wilcoxon signed rank test comparing UOMS methods to the baselines, smallest q-th best model with no significant difference, and mean/standard deviation AP across datasets.

	Method	Random	iForest-r	q_{AP}	mean AP	std AP
S-alone	xb,rs,ch	0.500	0.981	127	0.354	0.298
	EM	0.539	0.979	115	0.322	0.265
	IREOS	0.303	0.908	99	0.335	0.261
Consensus-based	UDR-ρ	**0.012**	0.905	104	0.383	0.283
	UDR-τ	**0.019**	0.952	109	0.379	0.282
	UDR-*NDCG*	**0.004**	0.825	93	0.384	0.270
	MC-ρ	**0.000**	0.217	89	0.395	0.289
	MC-τ	**0.002**	0.062	81	0.396	0.297
	MC-*NDCG*	**0.000**	0.182	82	0.404	0.291
	MC$_S$-ρ	**0.007**	0.706	108	0.385	0.289
	MC$_S$-τ	**0.001**	0.599	90	0.397	0.305
	MC$_S$-*NDCG*	**0.001**	0.205	83	0.391	0.285
	HITS	**0.000**	0.494	95	0.397	0.299
	Ens	**0.002**	0.730	81	0.371	0.282
Agg.	HITS-auth	**0.000**	0.577	94	0.401	0.286
	Ens-pseudo	**0.001**	0.422	79	0.373	0.282
Base.	Random	–	1.000	144	0.342	0.234
	iForest-r	–	–	84	0.399	0.300

performance to MC and can be used as a practical alternative.

Iterative methods HITS and Ens produce similar results to these simple one-shot methods, despite aiming to refine estimates of model trustworthiness over iterations. Again, as shown in Table 1.5, they significantly outperform Random and are comparable to iForest-r. The same holds true for their respective consensus scores, HITS-auth and Ens-pseudo, where model aggregation provides no significant advantage over selecting the best (single) model.

Table 1.6 shows a pairwise comparison of the consensus-based methods by one-sided Wilcoxon signed rank test, confirming mostly no significant difference between them.

Table 1.6: Comparison of consensus-based methods (UDR, MC, MC_S are based on *NDCG*).

	MC	MC_S	HITS	Ens
UDR	0.810	0.364	0.739	0.400
MC		0.551	**0.039**	0.116
MC_S			0.296	0.369
HITS				0.753

1.4.2.4 Running time analysis

In Fig. 1.1 we present for each method the running times on all datasets.[††] IREOS and EM/MV are both computationally demanding, while ineffective. In fact, IREOS takes more than 16 days (!) on the largest dataset (ALOI), due to kernel SVM training *for each* sample. MC is the next most expensive method, which is quadratic in the number of models, although still takes less than 1 hr on ALOI. In short, MC_S, Ens, and especially HITS prove to be both competitive as well as fast UOMS methods, completing within 10 minutes on our testbed.

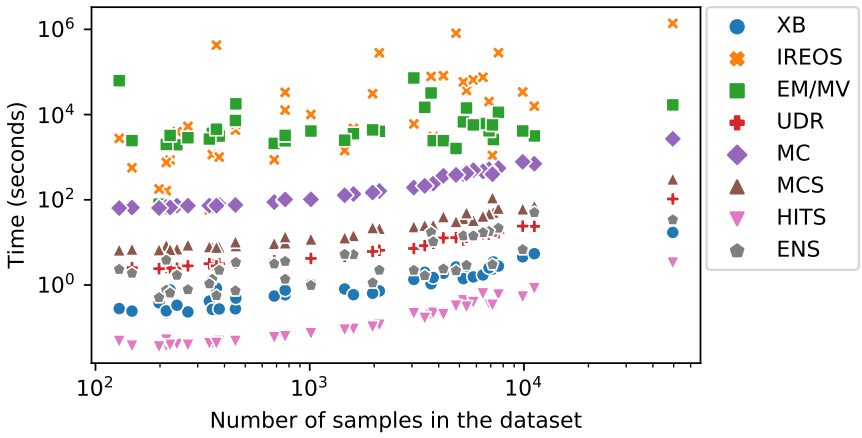


Figure 1.1: Run time comparison of UOMS methods.

[††]On an Intel Xeon E7 4830 v3 @ 2.1Ghz with 1TB RAM.

1.4.3 Discussion of the Results

We conclude with the two key takeaways from our study:

1. *None of the existing (stand-alone) UOMS methods is significantly different from random model selection* (!), and with the exception of IREOS. All are significantly worse than iForest (with random hyperparameter configuration). The slight advantage of IREOS can be attributed to it utilizing input features in addition to model outlier scores, unlike the other strategies that solely use the output scores. However, this advantage comes at the expense of significant running time.

2. *All consensus-based methods that we repurposed for UOMS are significantly better than random selection, but not different from the fast, state-of-the-art iForest detector with default HPs*.

Fig. 1.2 illustrates these takeaways where we show, via boxplots, the distribution of the performance difference between the model selected by each UOMS method and iForest-r across datasets. Consensus-based methods select models at best as good as iForest-r, where the AP difference concentrates around zero, whereas others are inferior.

These results suggest that **none of the UOMS methods we studied would be useful in practice**; because one would not first train a large *pool* of models – which would incur considerable computation – and then run a post hoc UOMS method to select a model, only to achieve comparable performance to a *single* iForest model (with default configuration) – which, in contrast, is extremely fast to train as it builds randomized trees on subsamples of data.

However, this is not to conclude that iForest is the best that one can hope to do. As given in Table 1.5, iForest-r is only as good as the 84-th best model per dataset. While it is the most competitive detector on average, other families outperform iForest on 28 out of 39 datasets in our study w.r.t. AP (See the original paper [190]). In Fig. 1.2 we also show the performance difference of the

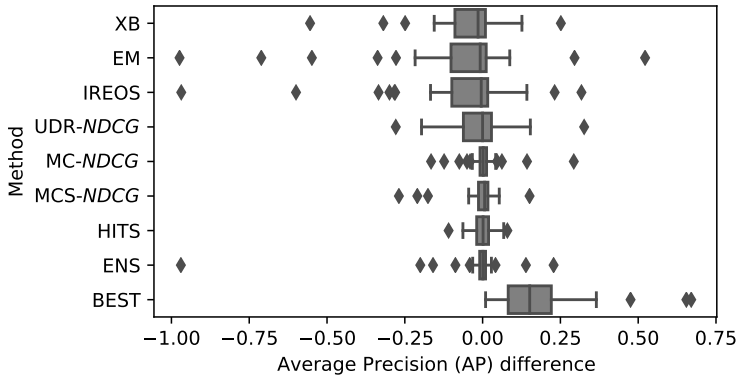


Figure 1.2: Distribution of performance difference across datasets: AP of selected model (by each UOMS method studied) minus that of iForest-r. Stand-alone methods and UDR are subpar, whereas other consensus-based method differences concentrate around zero (indicating no notable difference from iForest-r). Also shown for comparison is Best model on each dataset, showcasing ample room for improvement over iForest-r.

1-st Best model per dataset from iForest-r. (Also see the Appendix of the original paper [190]) One can clearly recognize that there is considerable room for progress in the area of UOMS.

1.5 Discussions

In this review, we considered the unsupervised outlier model selection (UOMS) problem: Given an unlabeled outlier detection task, which detection algorithm and associated hyperparameter (HP) settings should one use? This is a question of utmost importance not only for practitioners to do well on their new task, but also for the research community for being able to fairly compare new detection methods and keep an accurate track record of progress in the field. On the other hand, the problem is notoriously hard in the absence of any labeled data, any well-accepted objective or loss function, and potentially very large model space especially for deep outlier detectors with many HPs.

We focused on the body of methods that proposed internal (i.e., unsupervised) model evaluation strategies that leverage implicit signals from the input features and /or the output outlier scores

alone. On a large testbed comprising 297 models and 39 real-world datasets, we evaluated 7 different families of such internal evaluation strategies against simple baselines. Strikingly, we found that while consensus-based strategies are more promising against stand-alone ones which are not significantly better than random, none of them provides significant improvement over the state-of-the-art iForest detector with default HPs.

Our findings call for further research in this important area. As our work recently showed [71], deep detectors are considerably poor across varying HPs *on average* (i.e. when HPs are chosen randomly in the absence of any other guidance). As such, UOMS appears to stand as the biggest obstacle in front of deep models to fulfill their potential for outlier detection. A promising future direction is to develop stronger and faster internal strategies that can be leveraged within a meta-learning framework as in [348, 349, 342]. Our empirical evaluation revealed consensus-based internal strategies to be relatively more promising, which provides fertile ground for adaptation of prominent ideas from related areas such as truth discovery and crowdsourcing. We demonstrate the usage of consensus-based measures under the meta-learning framework for UOMS in Chapter 3 and for hyperparameter optimization in Chapter 4.

2

MetaOD: Meta-learning-based UOMS

This chapter is primarily based on:

Zhao, Yue, Ryan Rossi, and Leman Akoglu. "Automatic Unsupervised Outlier Model Selection." *Advances in Neural Information Processing Systems (NeurIPS)*, 2021

2.1 Highlight

In this chapter, we tackle the *unsupervised outlier model selection* (UOMS) problem (see the problem definition in §0.2.3), and propose a principled, data-driven approach called MetaOD. MetaOD is based on meta-learning, and stands on the prior performances of a large collection of existing detection models on an extensive corpus of historical outlier detection benchmark datasets. In a nutshell, the idea is to estimate a candidate model's performance on the new task (with no labels) based on its prior performance on *similar* historical tasks. We remark that MetaOD is strictly a model selection technique – that picks one model (a detector and its associated hyperparameter(s)) from a pool of (existing) candidate models – and *not* yet-another outlier detection algorithm itself.

In leveraging meta-learning, we establish a connection between the UOMS problem and the cold-start problem in collaborative filtering (CF), where the new task in UOMS is akin to a new user in CF (with no available evaluations, hence cold-start) and the model space is analogous to the item-set. Differently, OD necessitates the identification of a single best model (i.e., top-1 rank), whereas CF typically operates in a top-k setting. In CF, future recommendations can be improved based on user feedback which is not applicable to OD. Moreover, MetaOD requires the effective learning of task similarities based on characteristic dataset features (namely, meta-features) that capture the outlying properties within a dataset, whereas user features (location, age, etc.) in CF may be readily available.

In summary, the key contributions of this chapter include the following:

- **First Systematic Approach to UOMS**: We propose MetaOD, (to our knowledge) the first meta-learning effort on *unsupervised model selection* for OD tasks. Notably, given a new dataset (i.e., at test time), it does not rely on any ground-truth labels for model evaluation or any loss or heuristic criterion for model comparison. MetaOD stands on meta-learning in principle, and historical collections of outlier models and benchmark datasets in practice.

- **Problem Formulation:** We establish a *correspondence between UOMS and CF under cold-start*, where the new task "better likes" a model that performs better on similar historical tasks.

- **Specialized Meta-features:** We design novel meta-features to capture the outlying characteristics in a dataset toward effectively quantifying task similarity specifically among OD tasks.

- **Effectiveness and Efficiency**: Through extensive experiments on two benchmark testbeds that we have constructed, we show that *selecting* a model by MetaOD for each given task significantly outperforms always using a popular model like iForest, as well as other possible meta-learning approaches that we tailored for UOMS. Moreover, MetaOD incurs negligible run-time overhead (<1 second) at test time.

2.2 MetaOD Framework

2.2.1 Proposed MetaOD

In this chapter, we consider the UOMS problem and propose a meta-learning based solution, leveraging past experience on historical detection tasks. As such, our MetaOD relies on:

- a collection of historical outlier detection datasets $\boldsymbol{\mathcal{D}}_{\text{train}} = \{\mathbf{D}_1, \ldots, \mathbf{D}_n\}$, namely, a meta-train database with ground truth labels, i.e., $\{\mathbf{D}_i = (\mathbf{X}_i, \mathbf{y}_i)\}_{i=1}^{n}$, and

- the historical performances of the pool of candidate models, $\boldsymbol{\mathcal{M}}$, on the meta-train datasets. We denote by $\mathbf{P} \in \mathbb{R}^{n \times m}$ the performance matrix, where $\mathbf{P}_{ij}$ corresponds to the j-th model M_j's performance* on the i-th meta-train dataset $\mathbf{D}_i$.

*Area under the precision-recall curve (Average Precision or AP); can be substituted with any other measure.

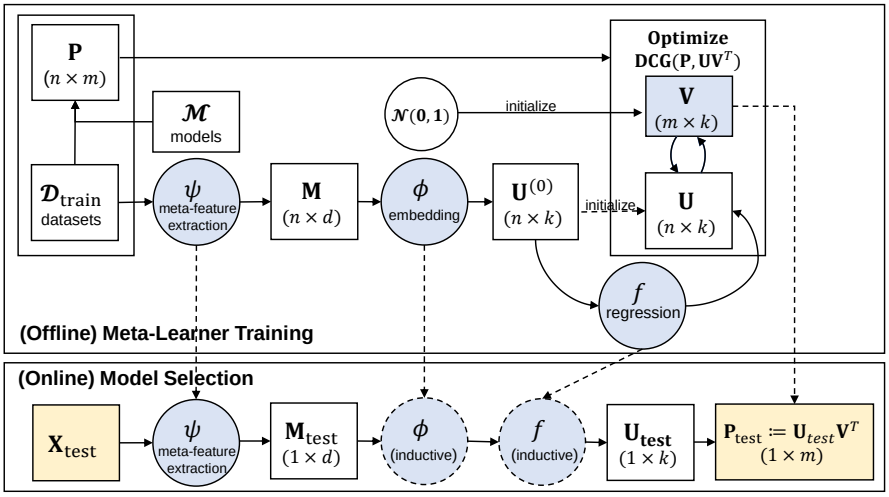


Figure 2.1: MetaOD overview; components that transfer from the offline (meta-learning) to the online (model selection) phase shown in blue; namely, meta-feature extractors (ψ), embedding model (φ), regressor f, dataset matrix $\mathbf{U}$, and model matrix $\mathbf{V}$. For the online phase, the input dataset $\mathbf{X}_{\text{test}}$ and the predicted model performance $\mathbf{P}_{\text{test}}$ are denoted in yellow.

Note that model performance can be evaluated on the historical meta-train datasets as they contain ground truth labels, which however is not the case for any newcoming task at test time.

Our MetaOD consists of two phases: **offline (meta-)training** of the meta-learner on $\mathcal{D}_{\text{train}}$, and **online prediction** that enables unsupervised model selection at test time for $\mathbf{D}_{\text{test}}$. Arguably, the running time of the offline phase is not critical. In contrast, model selection for a newcoming task should incur small run-time overhead, as it precedes the actual building of the selected OD model. Fig. 2.1 summarizes the process and the major components of MetaOD, where we highlight the components transferred from offline (meta-learning) to online stage (model selection) in blue. We also provide the detailed steps of MetaOD in pseudo-code, for both meta-training (offline) and model selection (online). Algorithm 2 provides detailed steps of MetaOD, for both offline (meta-learning) and online (model selection) stages.

Algorithm 2 METAOD: Offline and Online Phases

Input: (Offline) meta-train database $\mathcal{D}_{\text{train}}$, model set $\mathcal{M}$, latent dimension k; (Online) new OD dataset $\mathbf{X}_{\text{test}}$

Output: (Offline) Meta-learner for OD model selection; (Online) Selected model for $\mathbf{X}_{\text{test}}$

▶ (Offline) OD Meta-learner Training **(§2.2.1.1)**

Train & evaluate $\mathcal{M}$ on $\mathcal{D}_{\text{train}}$ to get performance matrix $\mathbf{P}$

Extract meta-features **(§2.2.2)**, $\mathbf{M} := \psi(\{\mathbf{X}_1, \ldots, \mathbf{X}_n\})$

Init. $\mathbf{U}^{(0)}$ by embedding meta-features, $\mathbf{U}^{(0)} := \varphi(\mathbf{M}; k)$

Init. $\mathbf{V}^{(0)}$ by standard normal dist.n $\mathbf{V}^{(0)} \sim \mathcal{N}(0, 1)$

while not converged **do** ▶ alternate. opt. by SGD, **(§2.2.3)**

    Shuffle dataset order in $\mathcal{D}_{\text{train}}$

    for $i = 1, \ldots, n$ **do**

        Update $\mathbf{U}_i$ by Eq. (2.7)

        for $j = 1, \ldots, m$ **do**

            Update $\mathbf{V}_j$ by Eq. (2.8)

        end for

    end for

end while

Train f regressing $\varphi(\mathbf{M}; k)$ onto $\mathbf{U}$ (at convergence)

Save extractors ψ, embed. φ, regressor f, $\mathbf{V}$ (at conv.)

▶ (Online) OD Model Selection **(§2.2.1.2)**

Extract meta-features, $\mathbf{M}_{\text{test}} := \psi(\mathbf{X}_{\text{test}})$

Get latent vector after embedding, $\mathbf{U}_{\text{test}} := f(\varphi(\mathbf{M}_{\text{test}}))$

Predict model set performance, $\mathbf{P}_{\text{test}} := \mathbf{U}_{\text{test}}\mathbf{V}^T$

Return $\arg\max_j \mathbf{P}_{\text{test}}(j)$ as the selected model for $\mathbf{X}_{\text{test}}$

2.2.1.1 (Meta-)Training (Offline)

In principle, meta-learning carries over prior experience on a set of historical tasks to "do better" on a new task. Such improvement can be unlocked only if the new task *resembles* and thus can build on *at least some* of the historical tasks (such as learning ice-skating given prior experience with roller-blading), rather than representing completely unrelated phenomena. This entails defining an effective way to capture task similarity between an input task and the historical tasks at hand.

In machine learning, similarity between meta-train and test datasets are quantified through characteristic features of a dataset, also known as *meta-features*. Those typically capture statistical properties of the data distributions. (See survey [293] for various types of meta-features.)

To capture **prior experience**, MetaOD first constructs the performance matrix $\mathbf{P}$ by running/building and evaluating all the m models in our defined model space $\boldsymbol{\mathcal{M}}$ on all the n meta-train datasets $\boldsymbol{\mathcal{D}}_{\text{train}}$.[†] To capture **task similarity**, it then extracts a set of d meta-features from each meta-train dataset, denoted by $\mathbf{M} = \psi(\{\mathbf{X}_1, \ldots, \mathbf{X}_n\}) \in \mathbb{R}^{n \times d}$ where $\psi(\cdot)$ depicts the feature extraction module. We defer the details on the meta-feature specifics to §2.2.2.

At this stage, it is easy to recognize the connection between the UOMS and collaborative filtering (CF) under cold start problems. Simply put, meta-train datasets are akin to existing users in CF that have prior evaluations on a set of models that are akin to the item-set in CF. The test task is akin to a newcoming user with no prior evaluations (and in our case, no possible future evaluation either), which however exhibits some pre-defined features.

Capitalizing on this connection, we take a matrix factorization based approach where $\mathbf{P}$ is approximated by the dot product of what-we-call dataset matrix $\mathbf{U} \in \mathbb{R}^{n \times k}$ and model matrix $\mathbf{V} \in \mathbb{R}^{m \times k}$. The intent is to capture the inherent dataset-to-model affinity via the dot product similarity in the k-dimensional latent space, such that $\mathbf{P}_{ij} \approx \mathbf{U}_i {\mathbf{V}_j}^T$ where matrix subscript denotes

[†]Note that this step takes considerable compute-time, which however amortizes to "do better" for future tasks. To this effect, we open-source our *trained* meta-learner to be readily deployed.

the row.

What loss criterion is suitable for factorization? In CF the typical goal is top-k item recommendation. In MetaOD, we aim to select the model with the best performance on a task which demands top-1 optimization. Therefore, we discard least squares and instead optimize the rank-based (row- or dataset-wise) discounted cumulative gain (DCG) [124],

$$\max_{\mathbf{U},\mathbf{V}} \sum_{i=1}^{n} \mathrm{DCG}_i(\mathbf{P}_i, \mathbf{U}_i\mathbf{V}^T) \ . \tag{2.1}$$

The factorization is solved via alternating optimization, where initialization plays an important role for such non-convex problems. We find that initializing $\mathbf{U}$, denoted $\mathbf{U}^{(0)}$, based on meta-features facilitates stable training, potentially by hinting at inherent similarities among datasets as compared to random initialization. Specifically, an embedding function $\varphi(\cdot)$ is used to set $\mathbf{U}^{(0)} := \varphi(\mathbf{M})$ for $\varphi : \mathbb{R}^d \mapsto \mathbb{R}^k, k < d$. Details on objective criteria and optimization are deferred to §2.2.3.

By construction, matrix factorization is transductive. On the other hand, we would need $\mathbf{U}_{\text{test}}$ to be able to estimate performances of the model set $\boldsymbol{\mathcal{M}}$ on a new dataset $\mathbf{X}_{\text{test}}$. To this end, one can learn an (inductive) multi-output regression model that maps the meta-features onto the latent features. We simplify by learning a regression function $f : \mathbb{R}^k \mapsto \mathbb{R}^k$ that maps the (lower dimensional) embedding features $\varphi(\mathbf{M})$ (which are also used to initialize $\mathbf{U}$) onto the final optimized $\mathbf{U}$. Note that this requires an inductive embedding function $\varphi(\cdot)$ to be applicable to newcoming datasets. In implementation, we use PCA for $\varphi(\cdot)$ and a random forest regressor for $f(\cdot)$ although MetaOD is flexible to accommodate any others provided they are inductive.

Remark: MetaOD improves over the existing methods that use CF in machine learning model selection (see §0.2.2) in two aspects. First, MetaOD builds specialized landmarker features tailored for capturing outlying characteristics of a dataset, while the existing ML model selection mainly uses generic statistical features (see §2.2.2). Second, MetaOD uses a customized (back-

propagatable/smooth) rank-based loss in CF for more effective top-1 optimization (see §2.2.3), while existing approaches mainly leverage mean squared loss (MSE).

2.2.1.2 Prediction for Unsupervised Model Selection (Online)

Meta-training stage yields the estimated functions $\psi(\cdot)$, $\varphi(\cdot)$, and $f(\cdot)$ as well as the model matrix $\mathbf{V} \in \mathbb{R}^{m \times k}$, which we save for test time (See Fig. 2.1). Given a new dataset $\mathbf{X}_{\text{test}}$ for OD, MetaOD first computes the corresponding meta-features as $\mathbf{M}_{\text{test}} := \psi(\mathbf{X}_{\text{test}}) \in \mathbb{R}^d$. Those are then embedded via $\varphi(\mathbf{M}_{\text{test}}) \in \mathbb{R}^k$, which are regressed to obtain the latent features, i.e., $\mathbf{U}_{\text{test}} := f(\varphi(\mathbf{M}_{\text{test}})) \in \mathbb{R}^k$. Model set performances are predicted as $\mathbf{P}_{\text{test}} := \mathbf{U}_{\text{test}}\mathbf{V}^T \in \mathbb{R}^m$. Finally, the model with the largest predicted performance is outputted as the selected model, that is,

$$\arg\max_j \ \langle\, f(\varphi(\psi(\mathbf{X}_{\text{test}}))), \mathbf{V}_j \,\rangle \ . \tag{2.2}$$

Remark: Notice that model selection by Eq. 2.2 for a newcoming dataset is solely based on its meta-features and other pre-trained components from meta-learning. It does not rely on ground-truth labels or any OD model evaluations, therefore, MetaOD provides *unsupervised* outlier model selection. Further, it does not require choosing or tuning any values at test time, and hence is fully automatic. In terms of computation, test-time embedding by φ (PCA) and regression by f (regression trees) take near-constant time given the small number of meta-features, embedding dimensions, and trees of fixed depth. Moreover, we use meta-features with computational complexity linear in the dataset size as we describe next.

2.2.2 Meta-Features for Outlier Detection

A key part of MetaOD is the extraction of meta-features that capture the important characteristics of an arbitrary dataset. Existing outlier detection models have different methodological designs (e.g.,

density, distance, angle, etc. based) and different assumptions around the topology of outliers (e.g., global, local, clustered). As a result, we expect different models to perform differently depending on the input dataset and the nature of outliers it exhibits—hence no "winner". In our meta-learning approach, the goal is to identify the datasets in the meta-train database that exhibit *similar* characteristics to a given test dataset, and focus on models that do well on those similar datasets. This is akin to recommending to a new user those items liked by similar users.

To this end, we extract meta-features that can be organized into two categories: (1) statistical features, and (2) landmarker features. Broadly speaking, the former captures statistical properties of the underlying data distributions; e.g., min, max, variance, skewness, covariance, etc. of the features and feature combinations. (See Appendix of [348] for the complete list.) These kinds of meta-features have been commonly used in the AutoML literature [39].

The optimal set of meta-features has been shown to be application-dependent [293]. Therefore, perhaps more important are the landmarker features, which are *problem-specific*, and aim to capture the *outlying* characteristics of a dataset. The idea is to apply a few of the fast, easy-to-construct OD models on a dataset and extract features from (*i*) the structure of the estimated OD model, and (*ii*) its output outlier scores. For the OD-specific landmarkers, we use four OD algorithms: iForest [180], HBOS [92], LODA [236], and PCA [272] (reconstruction error as outlier score). We choose the four OD algorithms due to their efficiency and diversity (as a group). First, they are all fast algorithms and able to handle large, high-dimensional datasets [7]. This makes the meta-feature generation efficient and practical in the real world. Second, these four OD algorithms as a group show decent diversity (i.e., internal detection mechanism) to capture rich outlying characteristics. Consider iForest as an example. It creates a set of what-is-called extremely randomized trees that define the model structure, from which we extract structural features such as average horizontal and vertical tree imbalance. As another example, LODA builds on random-projection histograms from which we extract features such as entropy. In addition, based on the list of outlier scores from these

models, we compute features such as dispersion, max consecutive gap in the sorted order, etc. See the details of the landmarker features in Appendix of [348].

2.2.3 Meta-Learning Objective and Training

2.2.3.1 Rank-based Criterion

A typical loss criterion for matrix factorization is the mean squared error (MSE), a.k.a. the Frobenius norm of the error matrix $\mathbf{P} - \mathbf{U}\mathbf{V}^T$. While having nice properties from an optimization perspective, MSE does not (at least directly) concern with the ranking quality. In contrast, our goal is to rank the models for *each* dataset row-wise, as model selection concerns with picking the best possible model to employ. Therefore, we use a rank-based criterion called DCG from the information retrieval literature [124]. For a given ranking, DCG is given as

$$\text{DCG} = \sum_r \frac{b^{rel\_r} - 1}{\log_2(r+1)} \tag{2.3}$$

where *rel_r* depicts the true relevance of the item ranked at the r-th position and b is a scalar (typically set to 2). In our setting, we use the performance of a model to reflect its true relevance to a dataset. As such, DCG for dataset i is re-written as

$$\text{DCG}_i = \sum_{j=1}^{m} \frac{b^{\mathbf{P}_{ij}} - 1}{\log_2(1 + \sum_{k=1}^{m} 1[\widehat{\mathbf{P}}_{ij} \leq \widehat{\mathbf{P}}_{ik}])} \tag{2.4}$$

where $\widehat{\mathbf{P}}_{ij} = \langle \mathbf{U}_i, \mathbf{V}_j \rangle$ is the predicted performance that dictates the ranking order. Intuitively, ranking high-performing models at the top leads to higher DCG, and a larger b increases the emphasis on the quality of models at the higher rank positions.

A challenge with DCG is that it is not differentiable, unlike MSE, as it involves ranking/sorting. Specifically, the sum term in the denominator of Eq. (2.4) uses the (nonsmooth) indicator function

to obtain the position of model j as ranked by the estimated performances. We circumvent this challenge by replacing the indicator function by the (smooth) sigmoid approximation [86] as follows.

$$\mathrm{DCG}_i \approx \mathrm{sDCG}_i = \sum_{j=1}^{m} \frac{b^{\mathbf{P}_{ij}} - 1}{\log_2(1 + \sum_{k=1}^{m} \sigma(\widehat{\mathbf{P}}_{ik} - \widehat{\mathbf{P}}_{ij}))} \tag{2.5}$$

2.2.3.2 Initialization & Alternating Optimization

Overall we optimize the smoothed criterion, sDCG, over all meta-train datasets $\boldsymbol{\mathcal{D}}_{\text{train}} = \{\mathbf{D}_i\}_{i=1}^{n}$ as

$$\min_{\mathbf{U},\mathbf{V}} \; L = -\sum_{i=1}^{n} \mathrm{sDCG}_i(\mathbf{P}_i, \mathbf{U}_i\mathbf{V}^T) \;, \tag{2.6}$$

by alternatingly solving for $\mathbf{U}$ as we fix $\mathbf{V}$ (and vice versa) by gradient descent. We initialize $\mathbf{U}$ by leveraging the meta-features, which are embedded to a space with the same size as $\mathbf{U}$. By capturing the latent similarities among the datasets, such an initialization not only accelerates convergence [353] but also facilitates convergence to a better local optimum. $\mathbf{V}$ is initialized from a unit Normal.

As we aim to maximize the total *dataset-wise* DCG, we make a pass over meta-train datasets one by one at each epoch. For brevity, we give the gradients for $\mathbf{U}_i$ and $\mathbf{V}_j$ in Eq.s (2.7) and (2.8), respectively.

$$\frac{\partial L}{\partial \mathbf{U}_i} = \ln(2) \sum_{j=1}^{m} \left[\frac{b^{\mathbf{P}_{ij}} - 1}{\beta_j^i \ln^2(\beta_j^i)} \sum_{k \neq j} \sigma(w_{jk}^i)(1 - \sigma(w_{jk}^i))(\mathbf{V}_k - \mathbf{V}_j) \right] \tag{2.7}$$

$$\frac{\partial L}{\partial \mathbf{V}_j} = -\ln(2) \sum_{j=1}^{m} \left[\frac{b^{\mathbf{P}_{ij}} - 1}{\beta_j^i \ln^2(\beta_j^i)} \sum_{k \neq j} \sigma(w_{jk}^i)(1 - \sigma(w_{jk}^i))\mathbf{U}_i \right] \tag{2.8}$$

where $w_{jk}^i = \langle \mathbf{U}_i, (\mathbf{V}_k - \mathbf{V}_j) \rangle$ and $\beta_j^i = \frac{3}{2} + \sum_{k \neq j} \sigma(w_{jk}^i)$. We provide the derivation below.

2.2.3.3 Gradient Derivation

In this section, we provide the details for the gradient derivation of MetaOD. It is organized as follows. We first provide a quick overview of gradient derivation in classical recommender systems, and then show the derivation of the rank-based criterion used in MetaOD.

Background. Given a rating matrix $\mathbf{P} \in \mathbb{R}^{n \times m}$ with n users rating on m items, $\mathbf{P}_{ij}$ denotes ith user's rating on the jth item in classical recommender system setting. For learning the latent factors in k dimensions, we try to factorize $\mathbf{P}$ into user matrix $\mathbf{U} \in \mathbb{R}^{n \times k}$ and the item matrix $\mathbf{V} \in \mathbb{R}^{d \times k}$ to make $\mathbf{P} \approx \mathbf{U}\mathbf{V}^T$.

In the classical matrix factorization setting, some entries of the performance matrix $\mathbf{P}$ is missing. Consequently, one may use stochastic gradient descent to minimize the mean squared error (MSE) between $\mathbf{P}$ and $\mathbf{U}\mathbf{V}^T$ through all non-empty entries. For each rating $\mathbf{P}_{ij}$, the loss L is defined as:

$$L_{i,j} = L(\mathbf{U}_i, \mathbf{V}_j^T) = (\mathbf{P}_{ij} - \mathbf{U}_i \mathbf{V}_j^T)^2 \tag{2.9}$$

The total loss over all non-empty entries is:

$$L = \sum_{i,j} (\mathbf{P}_{ij} - \mathbf{U}_i \mathbf{V}_j^T)^2 \tag{2.10}$$

The optimization process iterates over all non-empty entries of the performance matrix $\mathbf{P}$, and updates $\mathbf{U}_i$ and $\mathbf{V}_j$ using the learning rate η as:

$$\mathbf{U}_i \leftarrow \mathbf{U}_i - \eta \frac{\partial L}{\partial \mathbf{U}_i} \tag{2.11}$$

$$\mathbf{V}_j \leftarrow \mathbf{V}_j - \eta \frac{\partial L}{\partial \mathbf{V}_j} \tag{2.12}$$

Gradient Derivation for DCG-based Criterion. As we aim to maximize the total *dataset-wise* DCG, we make a pass over meta-train datasets one by one at each epoch as shown in Algorithm 2.

We update $\mathbf{U}_i$ and $\mathbf{V}_j$ by gradient descent as shown below. It is noted that the predicted performance of jth model on ith dataset is defined as the dot product of the corresponding dataset and model vector: $\widehat{\mathbf{P}}_{ij} = \mathbf{U}_i \mathbf{V}_j^T$. So Eq. (2.5) can be rearranged as:

$$
\begin{aligned}
-\text{sDCG}_i &= \sum_{j=1}^{m} \frac{b^{\mathbf{P}_{ij}} - 1}{\log_2(1 + \sum_{k=1}^{m} \sigma(\widehat{\mathbf{P}}_{ik} - \widehat{\mathbf{P}}_{ij}))} \\
&= \ln(2) \sum_{j=1}^{m} \frac{b^{\mathbf{P}_{ij}} - 1}{\ln(1 + \sum_{k=1}^{m} \sigma(\mathbf{U}_i \mathbf{V}_k^T - \mathbf{U}_i \mathbf{V}_j^T))} \\
&= \ln(2) \sum_{j=1}^{m} \frac{b^{\mathbf{P}_{ij}} - 1}{\ln(1 + \sigma(0) + \sum_{k \neq j} \sigma(\mathbf{U}_i \mathbf{V}_k^T - \mathbf{U}_i \mathbf{V}_j^T))} \\
&= \ln(2) \sum_{j=1}^{m} \frac{b^{\mathbf{P}_{ij}} - 1}{\ln(\frac{3}{2} + \sum_{k \neq j} \sigma(\mathbf{U}_i \mathbf{V}_k^T - \mathbf{U}_i \mathbf{V}_j^T))} \\
&= \ln(2) \sum_{j=1}^{m} \frac{b^{\mathbf{P}_{ij}} - 1}{\ln(\frac{3}{2} + \sum_{k \neq j} \sigma(\mathbf{U}_i \mathbf{V}_k^T - \mathbf{U}_i \mathbf{V}_j^T))} \qquad (2.13)
\end{aligned}
$$

(2.14)

We compute the gradient of $\mathbf{U}_i$ and $\mathbf{V}_j^T$ as the partial derivative of $-\text{sDCG}_i$ as shown in Eq. (2.13). To ease the notation, we define

$$
w_{jk}^i = \mathbf{U}_i \mathbf{V}_k^T - \mathbf{U}_i \mathbf{V}_j^T = \langle \mathbf{U}_i, (\mathbf{V}_k - \mathbf{V}_j) \rangle \qquad (2.15)
$$

$$
\beta_j^i = \frac{3}{2} + \sum_{k \neq j} \sigma(\mathbf{U}_i \mathbf{V}_k^T - \mathbf{U}_i \mathbf{V}_j^T) = \frac{3}{2} + \sum_{k \neq j} \sigma(w_{jk}^i) \qquad (2.16)
$$

By plugging Eq. (2.15) and (2.16) back into Eq. (2.13), it is simplified into

$$-\text{sDCG}_i = \ln(2)\sum_{j=1}^{m}\frac{b^{\mathbf{P}_{ij}}-1}{\ln(\frac{3}{2}+\sum_{k\neq j}\sigma(\mathbf{U}_i\mathbf{V}_k^T-\mathbf{U}_i\mathbf{V}_j^T))} = \ln(2)\sum_{j=1}^{m}\frac{b^{\mathbf{P}_{ij}}-1}{\ln(\beta_j^i)} \tag{2.17}$$

$$\tag{2.18}$$

We then obtain the gradients of $\mathbf{U}_i$ and $\mathbf{V}_j^T$ as follows:

$$\frac{\partial L}{\partial \mathbf{U}_i} = \frac{\partial(-\text{sDCG}_i)}{\partial \mathbf{U}_i} = \ln(2)\frac{\partial\left(\sum_{j=1}^{m}\frac{b^{\mathbf{P}_{ij}}-1}{\ln(\beta_j^i)}\right)}{\partial \mathbf{U}_i} \tag{2.19}$$

$$= \ln(2)\sum_{j=1}^{m}\left[\frac{b^{\mathbf{P}_{ij}}-1}{\beta_j^i\ln^2(\beta_j^i)}\sum_{k\neq j}\sigma(w_{jk}^i)(1-\sigma(w_{jk}^i))(\mathbf{V}_k-\mathbf{V}_j)\right] \tag{2.20}$$

$$\frac{\partial L}{\partial \mathbf{V}_j} = \frac{\partial(-\text{sDCG}_i)}{\partial \mathbf{V}_j} = \ln(2)\frac{\partial\left(\sum_{j=1}^{m}\frac{b^{\mathbf{P}_{ij}}-1}{\ln(\beta_j^i)}\right)}{\partial \mathbf{U}_i} \tag{2.21}$$

$$= -\ln(2)\sum_{j=1}^{m}\left[\frac{b^{\mathbf{P}_{ij}}-1}{\beta_j^i\ln^2(\beta_j^i)}\sum_{k\neq j}\sigma(w_{jk}^i)(1-\sigma(w_{jk}^i))\mathbf{U}_i\right] \tag{2.22}$$

2.3 Experiments

2.3.1 Experiment Setting

Model Set and Evaluation. We pair 8 SOTA OD algorithms and their corresponding hyperparameters to compose a model set $\mathcal{M}$ with 302 unique models. Model set $\mathcal{M}$ is composed by pairing outlier detection algorithms to distinct hyperparameter choices. Table 2.1 provides a comprehensive description of models, including 302 unique models composed by 8 popular outlier detection

(OD) algorithms. All models and parameters are based on the Python Outlier Detection Toolbox (PyOD)–see Chapter 6 for details.

Table 2.1: Outlier Detection Models; see hyperparameter definitions from PyOD [347]

Detection algorithm	Hyperparameter 1	Hyperparameter 2	Total
LOF [46]	n_neighbors: [1, 5, 10, 15, 20, 25, 50, 60, 70, 80, 90, 100]	distance: ['manhattan', 'euclidean', 'minkowski']	36
kNN [243]	n_neighbors: [1, 5, 10, 15, 20, 25, 50, 60, 70, 80, 90, 100]	method: ['largest', 'mean', 'median']	36
OCSVM [259]	nu (train error tol): [0.1, 0.2, . . . , 0.9]	kernel: ['linear', 'poly', 'rbf', 'sigmoid']	36
COF [285]	n_neighbors: [3, 5, 10, 15, 20, 25, 50]	N/A	7
ABOD [143]	n_neighbors: [3, 5, 10, 15, 20, 25, 50, 60, 70, 80, 90, 100]	N/A	7
iForest [180]	n_estimators: [10, 20, 30, 40, 50, 75, 100, 150, 200]	max_features: [0.1, 0.2, . . . , 0.9]	81
HBOS [92]	n_histograms: [5, 10, 20, 30, 40, 50, 75, 100]	tolerance: [0.1, 0.2, 0.3, 0.4, 0.5]	40
LODA [236]	n_bins: [10, 20, 30, 40, 50, 75, 100, 150, 200]	n_random_cuts: [5, 10, 15, 20, 25, 30]	54
			302

We evaluate METAOD and the baselines on 2 testbeds introduced below, resp. with 100 and 62 datasets, via cross-validation where datasets are split into meta-train/test in each fold. For each testbed, we first generate the performance matrix $\mathbf{P}$, by evaluating the models from $\mathcal{M}$ against the benchmark datasets in the testbed. For randomized detectors (random-split trees/random projections/etc.), we run five independent trials and record the average performance. For consistency, all models are built using the PyOD library [347] on an Intel i7-9700 @3.00 GHz, 64GB RAM, 8-core workstation. We compare two methods statistically, using the pairwise Wilcoxon signed rank test on performances across datasets (significance level $p < 0.05$).

Testbed Setup. Meta-learning works well if a new task can leverage prior knowledge; e.g., mastering motorcycle can benefit from bike riding experience. As such, METAOD relies on the assumption that a newcoming test dataset shares similarity with some meta-train datasets. We create two testbeds with different train/test dataset similarity, to systematically study the effect of task similarity.

1. **Proof-of-Concept (POC) testbed** contains 100 datasets that form clusters of similar datasets, where 5 different detection tasks ("siblings") are created from each one of 20 "mothersets".

2. **Stress Testing (ST) testbed** consists of 62 independent datasets from 3 different public-domain OD dataset repositories, which exhibit relatively lower similarity to one another.

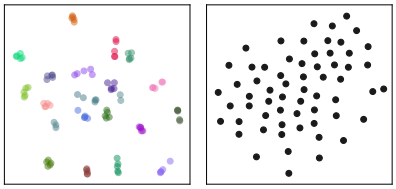


Figure 2.2: 2-D embedding of datasets in (left) POC and (right) ST. POC exhibits higher task similarity, wherein "siblings" (marked by same color) form clusters. ST contains independent datasets with no apparent clusters.

We refer to [348] for the complete list of datasets and details on testbed generation. Fig. 2.2 illustrates the differences between POC and ST testbeds, where the meta-features of their constituting datasets are embedded to 2-D by t-SNE [291]. By construction, POC consists of clusters and hence exhibits higher task/dataset similarity as compared to ST.

Baselines. Being the first work for UOMS, MetaOD does not have immediate competing baselines. Therefore we employ simple ideas and tailor some existing methods for comparison. We also create 2 variations of MetaOD (marked with †) for ablation analysis.

In [348], we give detailed descriptions of all 10 baselines. Briefly, they are organized as follows: (*i*) ***no model selection*** always employs the same popular model, namely **(1) LOF** [46] or **(2) iForest** [180], or the ensemble of all the models called **(3)** Mega Ensemble **(ME)**; (*ii*) ***simple meta-learners*** include **(4)** Global Best **(GB)** that selects the model with the largest avg. performance across meta-train datasets, **(5) ISAC** [132] and **(6)** ARGOSMART **(AS)** [217]; and (*iii*) ***optimization-based meta-learners*** include **(7)** Supervised Surrogates **(SS)** [325] and **(8) ALORS** [206].

Variants of MetaOD are **(9)** †MetaOD_**C** where performance and meta-feature matrices are concatenated as $\mathbf{C} = [\mathbf{P}, \mathbf{M}] \in \mathbb{R}^{n \times (m+d)}$, before factorization, $\mathbf{C} \approx \mathbf{U}\mathbf{V}^T$. Given a test dataset, zero-concatenated meta-features are projected and reconstructed as $[\widehat{\mathbf{P}}_{\text{test}}; \widehat{\mathbf{M}}_{\text{test}}] := [0 \ldots 0; \mathbf{M}_{\text{new}}]\mathbf{V}\mathbf{V}^T$; and **(10)** †MetaOD_**F** where $\mathbf{U}$ is fixed at $\varphi(\mathbf{M})$ after the embedding step and only $\mathbf{V}$ is optimized.

Additionally, we report Empirical Upper Bound **(EUB)** (only) for POC, as the performance of the best model on a dataset's 4 "siblings"; this (valuable) information is not available in practice– hence "upper bound". For ST with lower task similarity, we include Random Selection **(RS)** as a baseline.

2.3.2 POC Testbed Results

Testbed Setting. POC testbed is built to simulate the scenario where there are similar meta-train tasks to a given test task. We use the benchmark datasets[‡] by Emmott *et al.* [76], who created "childsets" from 20 independent "mothersets" by sampling. Consequently, the childsets generated from the same motherset using the same generation properties (e.g., the frequency of anomalies) can be deemed as "siblings" with large similarity. We build the POC testbed by using 5 siblings from each motherset, resulting in 100 datasets. We split them into 5 folds for cross-validation, each test fold containing 20 independent childsets without siblings.

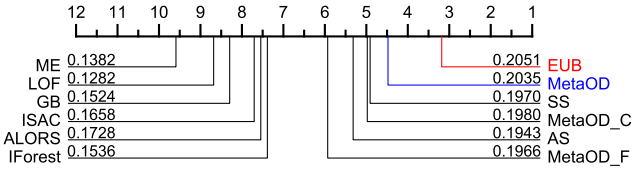


Figure 2.3: Comparison of avg. rank (lower is better) of methods w.r.t. performance across datasets in POC. Mean AP across datasets (higher is better) shown on lines. MetaOD is the top-performing meta-learner, and comparable to EUB.

Results. In Fig. 2.3, we observe that MetaOD **is superior to all baseline methods w.r.t. the average rank and mean average precision (MAP), and performs comparably to the Empirical Upper Bound (EUB)**. Table 1 (left) shows that MetaOD is the only meta-learner that is not significantly different from both EUB (MAP=0.2051) and the 4-*th* best model (0.2185). Moreover, MetaOD is significantly better than the baselines that do not employ any model selection (LOF

[‡] https://ir.library.oregonstate.edu/concern/datasets/47429f155

Ours	Baseline	p-value
MetaOD	**EUB**	0.0522
MetaOD	**4-th Best**	0.0929
MetaOD	LOF	0.0013
MetaOD	iForest	0.0090
MetaOD	ME	0.0004
MetaOD	GB	0.0051
MetaOD	ISAC	0.0019
MetaOD	**AS**	0.2959
MetaOD	**SS**	0.7938
MetaOD	ALORS	0.0025
MetaOD	**MetaOD_C**	0.6874
MetaOD	**MetaOD_F**	0.1165

Ours	Baseline	p-value
MetaOD	**58-th Best**	0.0517
MetaOD	RS	0.0001
MetaOD	LOF	0.0001
MetaOD	**iForest**	0.1129
MetaOD	ME	0.0001
MetaOD	GB	0.0030
MetaOD	ISAC	0.0006
MetaOD	AS	0.0009
MetaOD	SS	0.0190
MetaOD	ALORS	0.0001
MetaOD	MetaOD_C	0.0001
MetaOD	MetaOD_F	0.0001

Table 2.2: Pairwise statistical test results between METAOD and baselines by Wilcoxon signed rank test. Statistically better method shown in **bold** (both marked **bold** if no significance). In (left) POC, METAOD is the only meta-learner with no diff. from both EUB and the 4-*th* best model. In (right) ST, METAOD is the only meta-learner with no statistical diff. from the 58-*th* best model. It is statistically better than all except iForest.

(0.1282), iForest (0.1536), and ME (0.1382)), as well as all the other meta-learners including GB (0.1524), ISAC (0.1658) and ALORS (0.1728). For the full POC evaluation, see [348].

Averaging all models (ME) does not lead to good performance as one may expect. As shown in Fig. 2.3, ME is the worst baseline by average rank in the POC testbed. Using a single detector, e.g., iForest, is significantly better. This is mainly because some models perform poorly on any given dataset, and ensembling all the models indiscriminately draws overall performance down. Using selective ensembles [246] could be beneficial, however, ensembles of many models are expensive to build in practice. In contrast, METAOD is fast at test time and selects without building any models.

Meta-learners perform significantly better than methods without model selection. In particular, four meta-learners (METAOD, SS, METAOD_**C**, METAOD_**F**) significantly outperform single outlier detection methods (LOF and iForest) as well as the Mega Ensemble (ME) that averages all the models. METAOD respectively has 58.74%, 32.48%, and 47.25% higher MAP over LOF, iForest, and ME. These results signify the benefits of model selection.

Optimization-based meta learners generally perform better than simple meta learners. Top-3 meta learners by average rank (MetaOD, SS, and MetaOD_C) are all optimization-based and significantly outperform simple meta-learners like ISAC as shown in Fig. 2.3. Simple meta-learners weigh meta-features equally for task similarity, whereas others learn which meta-features matter (e.g., regression on meta-features), leading to better results. We find that MetaOD respectively achieves 33.53%, 22.74%, and 4.73% higher MAP than simple meta-learners including GB, ISAC, and AS.

2.3.3 ST Testbed Results

Testbed Setting. When meta-train datasets lack similarity to the test dataset, it is hard to capitalize on prior experience. In the extreme case, meta-learning may not perform better than no-model-selection baselines, e.g., a single detector. To investigate the impact of the train/test similarity on meta-learning performance, we build the ST testbed that consists of 62 public-domain datasets from 3 different repositories (See [348]) with relatively low similarity as shown in Fig. 2.2. For evaluation on ST, we use leave-one-out cross validation; each time using 61 datasets as meta-train.

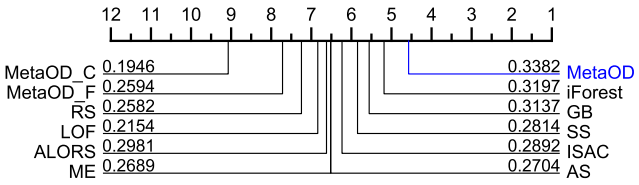


Figure 2.4: Comparison of avg. rank (lower is better) of methods w.r.t. performance across datasets in ST. Mean AP (higher is better) shown on lines. MetaOD outperforms all baselines.

Results. For the ST testbed, MetaOD **still outperforms all baseline methods w.r.t. average rank and MAP** as shown in Fig. 2.4. Table 1 (right) shows that MetaOD (0.3382) could select, from a pool of 302, the model that is as good as the 58-*th* best model (top 20%) per dataset (0.3513) in this challenging testbed. The comparable model changes from the 4-*th* best per dataset in POC to 58-*th* best in ST, which is expected due to the lower task similarity to leverage in ST. Notably, all other baselines are worse than the 80-*th* best model with statistical significance. Moreover,

MetaOD is significantly better than all baselines except iForest. Note that MetaOD also significantly outperforms RS, showing that it is able to exploit the meta-train database despite limited task similarity and not simply resorting to random picking. These results suggest that MetaOD **is a good choice under various extent of similarity among train/test datasets**. We refer to [348] for detailed ST results on individual ST datasets.

Training stability affects performance for optimization-based methods. Notably, several optimization-based meta-learners, such as ALORS and MetaOD_**C**, do not perform well for ST. We find that the training process of matrix factorization is not stable when latent similarities are weak. In MetaOD, we employ two strategies that help stabilize the training. First, we leverage meta-feature based (rather than random) initialization. Second, we use cyclical learning rates that help escape saddle points for better local optima [275]. Consequently, MetaOD (0.3382) significantly outperforms ALORS (0.2981) and MetaOD_**C** (0.1946) with 13.45% and 73.79% higher MAP.

Global methods outperform local methods under limited task similarity. In ST, datasets are less similar and simple meta-learners that leverage task similarity locally often perform poorly. For example, AS selects the model based on the 1-NN, and is likely to fail if the most similar meta-train task is still quite dissimilar to the current task. Notably, the global meta-learner GB outperforms AS and ISAC. Note the opposite ordering among these methods in POC as shown in Fig. 2.3. In short, **effectiveness of simple meta-learners tends to be sensitive to the train/test dataset similarity**, which makes them hard to use in general. In contrast, MetaOD performs well in both settings.

2.3.4 Runtime Analysis

Empowered by meta-training, MetaOD (meta-feature generation and model selection) takes less than 1 second on most test datasets, as shown in Fig. 2.5, where it incurs negligible overhead relative to building/training the selected outlier model (≈10% on avg.). Fig. 2.6 corroborates the statement by showing the comparison on the 10 largest datasets in POC.

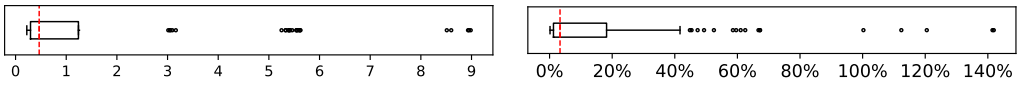


Figure 2.5: MetaOD running time at test time in sec.s (left), and percentage of time relative to building the selected model (right). Notice that it is fast, and incurs negligible computational overhead.

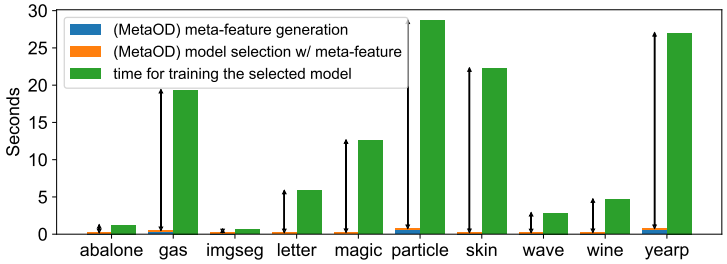


Figure 2.6: Time for MetaOD vs. training of the selected model (on 10 largest datasets in POC). MetaOD incurs only negligible overhead (diff. shown w/ black arrows).

Notably, meta-feature extraction may be trivially parallelized whereas the model selection is even faster, e.g., using SUOD [345], effectively taking constant time (See §2.2.1.2).

2.4 Discussions

We addressed the unsupervised outlier model selection (UOMS) problem *without relying on any labels, model evaluations or comparisons* in this chapter. Our proposed MetaOD is a meta-learner, and builds on an extensive pool of historical outlier detection datasets and models. Given a new task, it selects a model based on the past performances of models on similar historical tasks. To effectively capture task similarity, we designed novel problem-specific meta-features. Importantly, MetaOD is (*i*) **fully automatic**, requiring no supervision at test time, and (*ii*) **lightweight**, incurring relatively small selection time overhead prior to outlier model building. Extensive experiments on two large testbeds showed that MetaOD significantly improves detection performance over model to use outperforms always using some of the most popular outlier models as well as several other meta-learners tailored for UOMS.

Future work can address UOMS in the continuous hyperparameter space, leverage self-aware learning [169] and conformal prediction [264] to estimate the confidence in selection, and explore the potential bias and fairness issues in OD model selection [63, 267].

3

ELECT: Automatic Outlier Model Selection in Iterations

This chapter is primarily based on:

Yue Zhao*, Sean Zhang, Leman Akoglu. "ELECT: Toward Unsupervised Outlier Model Selection". *IEEE International Conference on Data Mining (ICDM)*, 2022.

3.1 Highlight

Moving beyond designing yet-another detection algorithm, it is exactly our goal in this chapter to systematically address the unsupervised outlier model selection (UOMS) problem (see the definition in §0.2.3), which involves selecting an OD algorithm as well as its hyperparameter(s) for a given new OD dataset. While essential, the problem is notoriously hard for: (i) model evaluation (say, on hold-out data) is infeasible due to the lack of labels, and (ii) model comparison is inapplicable as there is no universal loss/objective criterion applicable to all OD algorithms. Notably, even if they were available, using labels in OD model selection could be challenged as the available labels may be too limited to be comprehensive/high-coverage, and in turn the selected model based on known labels may not be suited to unknown/emerging types of outliers in deployed systems.

Prior Work. In Chapter 2, we propose MetaOD. where similarity among the input task and historical OD tasks is leveraged. In MetaOD, task similarities are measured based on meta-features, i.e., summary statistics of a dataset. Our present work in this chapter is inspired by MetaOD, yet takes a distinct approach for quantifying task similarity. See discussion of related work in §0.2.2.

Present Work. The primary motivation behind this chapter is to use *performance-driven similarity* for quantifying the resemblance among the input task and historical OD tasks in meta-learning. Building upon this, we propose ELECT, an iterative meta-learning approach for unsupervised outlier model selection. In principle, meta-learning carries over prior "experience" from a database of historical tasks to facilitate the learning on a new task, provided that the new task resembles at least some of the historical ones. As we aim to carry over the information of prior performance of OD models on historical tasks to effectively select a high-performance model for an input task, we argue that the most suitable measure of task similarity is *the similarity of model performances on two tasks*. Of course, model performances are unknown and cannot be directly evaluated on the new, unsupervised task. This is where we employ meta-learning by training a supervised

predictor on the historical tasks that maps internal information of trained models (without using any labels) to their actual performance. We carefully and adaptively search for similar historical tasks and select a model that achieves high performance on those, without training too many models on the input task, such that UOMS incurs negligible computational overhead. We find it important to remark that UOMS precedes OD (say, by the selected model), and that ELECT is strictly a model selection technique other than a new OD algorithm. Our contributions:

- **New UOMS Method.** We introduce ELECT, a novel approach to unsupervised OD model selection based on meta-learning. It capitalizes on historical OD tasks with labels to select an effective, high-performance model for a new task without any labels.
- **Performance-driven Task Similarity.** The key mechanism behind meta-learning is to effectively identify and transfer knowledge from *similar* historical tasks to the new task. Unlike prior work [348] that relies on handcrafted meta-features to quantify task similarity, ELECT takes a direct and goal-driven approach, and deems two tasks similar if OD models *perform* similarly on both tasks.
- **Unsupervised Adaptive Model Search.** As ground-truth labels are unavailable for a new task, ELECT learns (during meta-training) to quantify performance based on *internal* model performance measures. Moreover, it carefully decides which model to train on the new task iteratively, keeping as small as possible the total number of models trained before selection, thereby reducing computation time. Moreover, it is an *anytime* algorithm that can output users a selected model at any time they choose to stop it.
- **Effectiveness.** Through extensive experiments on two testbeds, we show that *selecting* a model by ELECT is significantly better than employing popular models like iForest as well as all meta-feature baselines, including the SOTA MetaOD ($p = 0.0016$), in the controlled testbed.

3.2 Overview of ELECT

As defined in Chapter 0 §0.2.3 Problem 1, we consider the model selection problem for unsupervised outlier detection. UOMS selects a model (i.e., {`detector`, `configuration`}) for a new unsupervised task $\mathcal{D}_{\text{test}} = (\mathbf{X}_{\text{test}}, \emptyset)$ from a predefined model pool $\boldsymbol{\mathcal{M}} = \{M_1, \ldots, M_m\}$.

3.2.1 Background: Hyperparameter Optimization (HPO) and Meta-Learning

HPO has gained significant attention within AutoML owing to the advent of complex models with large HP spaces that are costly to train and thereby to tune [81]. Besides model-free techniques such as grid or random search [168], Bayesian optimization and the adaptation of Sequential Model-Based Optimization (SMBO) [131] is one of the main lines of work in HPO. The idea is to iterate between (i) fitting a *surrogate* performance function onto past HP evaluations, and (ii) using it to choose the next HP based on an *acquisition* function. Well-established SMBO approaches include SMAC [120] and Auto-WEKA [286]. We remark that SMBO cannot directly be used for UOMS, as we cannot evaluate model performance reliably to train an effective surrogate. Instead, ELECT employs meta-learning to estimate the mean and variance of a candidate model's performance (to be used for *acquisition*) based on similar historical tasks.

Meta-learning has been used for HPO in various forms [313]; e.g. to warm-start SMBO [83], prune the HP search space [311], and transfer surrogate models [332]. Active testing [166] has used meta-learning for model selection, which differs from our work in two key aspects.

3.2.2 Proposed ELECT: Overview

At the heart of our proposed approach to UOMS lies *meta-learning*, where the underlying principle is to transfer useful information from historical tasks to a new task. To this end, ELECT takes as

input a set of historical OD datasets $\boldsymbol{\mathcal{D}}_{\text{train}} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$, namely, a meta-train database with ground-truth labels where $\{\mathcal{D}_i = (\mathbf{X}_i, \mathbf{y}_i)\}_{i=1}^{n}$ to compute:

- the historical output scores of each candidate model $M_j \in \boldsymbol{\mathcal{M}}$ on each meta-train dataset $\mathcal{D}_i \in \boldsymbol{\mathcal{D}}_{\text{train}}$, where $\mathcal{O}_{i,j} := M_j(\mathcal{D}_i)$ refers to the j-th model's output outlier scores for the points in the i-th meta-train dataset $\mathcal{D}_i$; and
- the historical performances matrix $\mathbf{P} \in \mathbb{R}^{n \times m}$ of $\boldsymbol{\mathcal{M}}$ on $\boldsymbol{\mathcal{D}}_{\text{train}}$, where $\mathbf{P}_{i,j}$ depicts model M_j's performance* on meta-train dataset $\mathcal{D}_i$.

ELECT consists of two phases. In a nutshell, during the **(meta-)training phase (offline)**, it learns information necessary to quantify similarity between two tasks based on $\boldsymbol{\mathcal{D}}_{\text{train}}$. It uses this information during the **(outlier) model selection phase (online)** to identify similar meta-train tasks to a new input task $\mathcal{D}_{\text{test}}$ and chooses a model without using any labels.

Next, we present a high-level description of these phases. Fig. 3.1 illustrates the key steps in each phase.

3.2.2.1 (Offline) Meta-training Overview

Given historical tasks $\boldsymbol{\mathcal{D}}_{\text{train}}$ and the new task $\mathcal{D}_{\text{test}}$ for model selection, the main idea is to first identify a subset of tasks $\mathcal{N} \subset \boldsymbol{\mathcal{D}}_{\text{train}}$ that are *similar* to $\mathcal{D}_{\text{test}}$, and then choose the model that performs the best on average on those "neighbor" tasks. Thus, a key ingredient of ELECT is an effective task similarity measure. Distinctly, it utilizes **performance-driven task similarity**: two tasks are similar if the performance rank-ordering of all the models on each task is similar.

Of course, the similarity of $\mathcal{D}_{\text{test}}$ to meta-train datasets cannot be computed based on the ground-truth model performances on $\mathcal{D}_{\text{test}}$ given that labels are unavailable for the input task (obviously,

*Area under the precision-recall curve (AUCPR, a.k.a. Average Precision or AP); can be substituted with any other accuracy measure of interest.

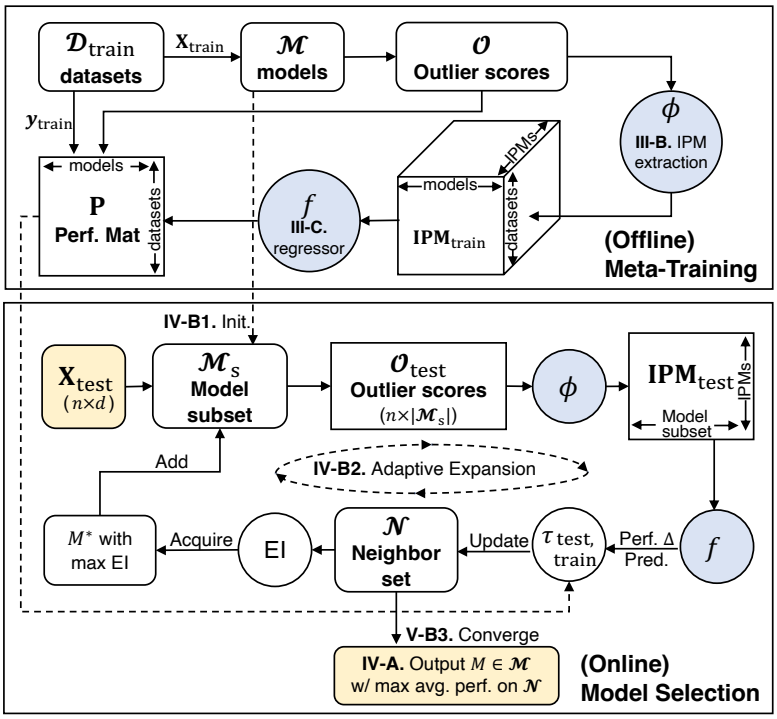


Figure 3.1: Main steps during the Offline and Online phases of ELECT.

that would void the model selection problem altogether). A core idea of ELECT is to learn to estimate true performance from **internal performance measures** (IPMs) during meta-training. IPMs are unsupervised signals that are solely based on the input points and/or a given model's output (e.g. outlier scores) that can be used to compare two models [91, 196]. However, they are noisy/weak indicators of performance [190]. ELECT makes the best use of these weak signals by *learning* to regress the IPMs of two models onto their true performance difference.

3.2.2.2 (Online) Model Selection Overview

In the (online) model selection phase, given $\mathcal{D}_{\text{test}}$, we can build/run each candidate model on $\mathbf{X}_{\text{test}}$, extract the corresponding IPMs, and use the regressor (from meta-training) to estimate pairwise model performance differences for quantifying $\mathcal{D}_{\text{test}}$'s similarity to each meta-train dataset based on

the all-pairs rank similarity of the models. However, building and internally evaluating *all* of the candidate models would be computationally expensive.

To avoid training all the models at test time, a core idea behind ELECT is to quantify similarity between $\mathcal{D}_{\text{test}}$ and meta-train tasks based on only a small *subset* of the models, which is carefully and adaptively picked and expanded. At each iteration, this strategy picks the next model to train on $\mathcal{D}_{\text{test}}$ with the best explore-exploit trade-off, where exploration (picking a model with high uncertainty in performance estimate) helps refine similarity and thereby update the set of neighbor tasks, and exploitation (picking a model with high-performance estimate) ensures that the neighbor tasks share similarity w.r.t. the well-performing models. Ultimately, the goal is to effectively identify similar historical tasks based on as few models run on $\mathcal{D}_{\text{test}}$ as possible. Upon convergence, the model with the largest average performance on the neighbor tasks (from the latest iteration) is reported as the selected model for $\mathcal{D}_{\text{test}}$.

We present the technical details of ELECT's (offline) meta-training and (online) model selection phases in §3.3 and §3.4. Additional implementation details are given in §3.5.1 & 3.5.2.

3.3 ELECT: Meta-training (Offline)

The main component of meta-training is to learn, using historical (meta-train) tasks, how to quantify performance-based task similarity from imperfect indicators of performance, namely internal performance measures (IPMs). In the following, we introduce our task similarity measure (§3.3.1), specific IPMs used by ELECT (§3.3.2), and how to do the mapping between the two (§3.3.3).

3.3.1 Performance-driven Task Similarity

Meta-learning carries the prior experience on historical tasks over to a new task, provided that the latter resembles some of the historical tasks. Thus, quantifying task similarity is crucial to the effec-

tiveness of a meta-learning based approach [292].

Our work is most similar to the recent work by Zhao *et al.* [348] that developed the first unsupervised approach to outlier model selection. Their proposed METAOD quantifies task similarity based on *meta-features*, which reflect general statistical properties of a dataset where the majority of the points are inliers. As such, two datasets with similar inlier distribution but different types of outliers may look similar w.r.t. meta-features, while different outlier models may be more effective in detecting different outlier types that they exhibit. In such scenarios, meta-feature similarity would be a poor indicator of model performance similarity.

In contrast to METAOD, we propose a *performance-driven* similarity measure, where two tasks are deemed more similar, the more the same set of models perform well/poorly on both tasks. Arguably, ours is a *direct* means to the end goal—only when task similarity is defined in this performance-based fashion would it be natural (and even gold standard) to choose a model for a new task that performs well on its neighbor tasks.

Our performance-driven similarity is a **rank-based** measure, called the weighted Kendall's tau rank correlation coefficient [270] (τ for short), which quantifies the similarity between the *ordering of the candidate models by performance*. Formally, let $\mathbf{P}_i \in \mathbb{R}^m$ depict the i-th row of $\mathbf{P}$ containing the detection performances of models $\boldsymbol{\mathcal{M}} = \{M_1, \ldots, M_m\}$ on dataset $\mathcal{D}_i$, and let $\Delta^{(i)}_{j,j'} = \mathbf{P}_{i,j} - \mathbf{P}_{i,j'}$ denote the difference between the performances of M_j and $M_{j'}$ on $\mathcal{D}_i$. Then, τ between two tasks $\mathcal{D}_i$ and $\mathcal{D}_{i'}$ is defined as follows.

$$\tau^{(i,i')} = \tau(\mathbf{P}_i, \mathbf{P}_{i'}) = \frac{\sum_{j=1}^{m-1}\sum_{j'=j+1}^{m} w^{(i,i')}_{j,j'}}{\sum_{j=1}^{m-1}\sum_{j'=j+1}^{m} |w^{(i,i')}_{j,j'}|}\,, \quad \text{where}$$

$$w^{(i,i')}_{j,j'} = \begin{cases} 1 & \text{if } \Delta^{(i)}_{j,j'} = \Delta^{(i')}_{j,j'} = 0 \\ \Delta^{(i)}_{j,j'} \Big/ \Delta^{(i')}_{j,j'} & \text{else if } |\Delta^{(i)}_{j,j'}| \leq |\Delta^{(i')}_{j,j'}| \\ \Delta^{(i')}_{j,j'} \Big/ \Delta^{(i)}_{j,j'} & \text{else; i.e. } |\Delta^{(i)}_{j,j'}| > |\Delta^{(i')}_{j,j'}| \end{cases} \tag{3.1}$$

Intuitively, τ quantifies the concordance/discordance between pairwise model performances, where τ becomes smaller both when the rank-orders are discordant (i.e., $w_{j,j'}$ is negative) and when they are concordant but the Δ-differences are disproportionate (i.e., $w_{j,j'}$ is near-zero). Put differently, two tasks are more similar when the models are ranked similarly, and also the perf. of each model are similar in value on both.

3.3.2 Internal Performance Measures (IPMs)

To compute the similarity of a new task to historical meta-train tasks, we would need pairwise model performance differences on the new task. These model performances, of course, cannot be computed due to the lack of ground-truth labels. In fact, having these at hand would obviate the model selection problem altogether. Then, how can we really compute performance-driven task similarity?

The key idea is to learn a predictive model of ground-truth performance (which is available for meta-train tasks) from unsupervised indicators/features. The challenge is to identify such features that correlate with model performance. Notably, there exists a small literature on internal evaluation of outlier detection models [91, 196], and recently also unsupervised model selection strategies for deep representation learning based on other internal measures [75, 177]. These are called *internal* performance measures (IPMs) as they solely rely on the input samples, the outlier scores and/or the consensus between the candidate models. For example, consensus-based IPMs in principle associate closeness to an overall consensus of outlier scores/ranking with being a better model. We refer to Chapter 1 for more details on specific IPMs where we called them *internal evaluation strategies*.

An IPM is exactly designed to compare models without any ground truth labels. Then, the question is why do not we simply and directly use an IPM for model selection? The reasons are two-fold. First, effectiveness-wise, IPMs are weak/noisy signals of true performance. As a recent study showed, they enable only slightly better model selection than random [190]. Distinctly, ELECT plugs the IPMs into a meta-learning framework, taking advantage of machine learning's ability to map weak

signals onto desired ones. Related, it would not be clear *which* IPM we should use for model selection (a "chicken-egg" scenario), provided several options. Notably, ELECT leverages all/any available IPMs as internal features in meta-training. Second, computation-wise, we would need to train each and every candidate model on $\mathcal{D}_{\text{test}}$ to obtain its IPM and compare it to others, leading to inhibitively high cost. In contrast, ELECT employs meta-learning to identify similar historical tasks based on a small *subset* of trained models on $\mathcal{D}_{\text{test}}$ while still being able to select among *all* of the candidate models via their (ground-truth) performance on these neighbor tasks.

3.3.3 From IPMs to (True) Model Performance

At the core of ELECT's meta-training is *learning* to map IPMs onto ground-truth performance by the supervision from the meta-train database. In particular, ELECT learns a regressor that maps the IPMs from two models onto their performance difference.

More formally, let $\varphi(\cdot)$ denote the process of extracting various IPMs of model M_j when trained on $\mathcal{D}_i$, and $\mathbf{m}_{i,j}$ denote the corresponding vector of IPMs. ELECT uses three IPMs; namely ModelCentrality (MC), HITS, and SELECT, as described in [190]. The regression function, named *pairwise performance predictor*, maps the IPMs of any pair of models M_j and $M_{j'}$ onto their performance difference on a dataset, i.e. $f(\mathbf{m}_{i,j}, \mathbf{m}_{i,j'}) \mapsto \Delta^{(i)}_{j,j'}$. In implementation we use LightGBM [135], while it is flexible in choosing any other. We design $f(\cdot)$ for pairwise prediction such that its output can be directly plugged into Eq. (3.1) for computing task similarity. We find it important to remark that provided with $\varphi(\cdot)$ and the trained $f(\cdot)$ at test time, measuring performance-driven task similarity via τ becomes possible without using any ground-truth labels.

For clarity of presentation, we defer a few implementation details to §3.5.1, where we describe how to incrementally compute the IPMs as additional models are trained on the new input task at test time, as well as how to effectively train the regressor and use its predictions at test time.

3.4 ELECT: Model Selection (Online)

After the meta-training phase, MetaOD is ready to admit a new task for model selection. Simply, it selects the highest performance model on meta-train tasks (or meta-tasks) that are very similar to the new task (§3.4.1). It identifies these similar meta-tasks iteratively by refining the similarity estimates adaptively (§3.4.2).

3.4.1 Model Selection via Similar Meta-tasks

Given a new task $\mathcal{D}_{\text{test}}$, we aim to identify its similar meta-train tasks (referred as the neighbor set $\mathcal{N} \subset \boldsymbol{\mathcal{D}}_{\text{train}}$). By the principle of meta-learning, the model(s) that outperform on the neighbor set is likely to outperform on the new task as well. Consequently, we could output the model with the largest average performance on the neighbor set as the selected model for the new task, that is,

$$\underset{M_j \in \boldsymbol{\mathcal{M}}}{\arg\max} \; \frac{1}{|\mathcal{N}|} \sum_{\mathcal{D}_i \in \mathcal{N}} \mathbf{P}_{i,j} \; . \tag{3.2}$$

If the model performances $\mathbf{P}_{\text{test}}$ were available for $\mathcal{D}_{\text{test}}$, we could iterate over the meta-train database to measure task similarity via Kendall's tau in Eq. (3.1), and pick $\mathcal{N}$ to be the top t most similar meta-train tasks, as shown in Eq. (3.3). Here t denotes the size of the neighbor set (i.e. $|\mathcal{N}| = t$), which can be chosen by cross-validation on the meta-train database (see Appx. §3.5.2.1 for details).

$$\mathcal{N} := \underset{i=1\ldots n}{\text{top-}t} \; \tau^{(\text{test},i)} = \underset{i=1\ldots n}{\text{top-}t} \; \tau(\mathbf{P}_{\text{test}}, \mathbf{P}_i) \tag{3.3}$$

However, we cannot directly identify $\mathcal{N}$ due to the lack of ground-truth labels and thus evaluations $\mathbf{P}_{\text{test}}$ on the new task. Note the calculation of Kendall's tau in Eq. (3.1) only depends on pairwise performance gaps (i.e. Δ-differences) to measure concordance/discordance, where the trained regressor $f(\cdot)$ for pairwise performance gap prediction comes into play. By plugging the **predicted pairwise gaps** (i.e. $\widehat{\Delta}$-differences) of the new task into Eq. (3.1), we could therefore estimate its

neighbor set $\mathcal{N}$ even when $\mathbf{P}_{\text{test}}$ is inaccessible.

Specifically, for the new task $\mathcal{D}_{\text{test}}$, we first get its outlier scores $\mathcal{O}_{\text{test}} = \boldsymbol{\mathcal{M}}(\mathcal{D}_{\text{test}})$ and build the IPMs $\mathbf{m}_{\text{test}} = \varphi(\mathcal{O}_{\text{test}})$ across candidate models.Note that We slightly abuse notation here and use $\mathbf{m}_{\text{test}}$ to depict the IPM vectors for all models, which is in fact a matrix. We could then predict the performance gap of any pair of models for $\mathcal{D}_{\text{test}}$ using the regressor $f(\cdot)$ by

$$\widehat{\Delta}_{j,j'}^{(\text{test})} := f(\mathbf{m}_{\text{test},j}, \mathbf{m}_{\text{test},j'}) \approx \Delta_{j,j'}^{(\text{test})} , \tag{3.4}$$

where $j = 1 \ldots m$ and $j < j'$. The estimated Kendall-tau similarity ($\widehat{\tau}$) between the new task and meta-train tasks can be calculated using the *predicted* pairwise performance gaps on the new task $\widehat{\Delta}^{\text{test}}$ in Eq. (3.4) and the *actual* performance gaps on meta-train tasks Δ^{train}, where e.g., we denote by $\widehat{\tau}^{\,(\text{test},i)}$ the estimated Kendall-tau similarity between $\mathcal{D}_{\text{test}}$ and the i-th meta-train dataset.

Then, by plugging the estimated task similarities into Eq. (3.3), we obtain the neighbor set $\mathcal{N}$ of the new task as the top-t meta-train datasets with the highest estimated Kendall-tau similarity *without relying on ground-truth labels*, i.e.,

$$\mathcal{N} := \underset{i=1\ldots n}{\text{top-}t}\ \widehat{\tau}^{\,(\text{test},i)} \approx \underset{i=1\ldots n}{\text{top-}t}\ \tau^{(\text{test},i)} . \tag{3.5}$$

3.4.2 Unsupervised Adaptive Search

3.4.2.1 Motivation and Initialization

Quantifying the task similarity by Kendall-tau using the full model set $\boldsymbol{\mathcal{M}} = \{M_1, \ldots, M_m\}$ (based on all $\binom{m}{2}$ pairs of performance gaps) incurs high computational cost, as it involves model fitting to get outlier scores and extracting corresponding IPMs for *all* candidate models. We therefore propose to only measure task similarity based on a *subset* of the models, denoted $\boldsymbol{\mathcal{M}}_s \subset \boldsymbol{\mathcal{M}}$ for model subset. Initially, $\boldsymbol{\mathcal{M}}_s$ can be set to a small random subset of $\boldsymbol{\mathcal{M}}$, while a more careful

initialization strategy may facilitate better similarity measurement. In ELECT, we design a coverage-maximization strategy for initializing $\boldsymbol{\mathcal{M}}_s$, details of which are described in Appx. §3.5.2.2. The ablation in §3.6.4.1 shows it is significantly better than random initialization.

3.4.2.2 Iteration: Adaptively Expanding Model Subset

The initial $\boldsymbol{\mathcal{M}}_s$ may not be sufficient to capture a complete picture of task similarity. Therefore, ELECT expands the model subset iteratively to refine the task similarity estimates and thereby obtain increasingly better estimates of the most similar datasets to $\mathcal{D}_{\text{test}}$.

Assume for now that an objective criterion exists for choosing the next model to be included in $\boldsymbol{\mathcal{M}}_s$, then, the adaptive search proceeds as follows. In each iteration, we update the neighbor set $\mathcal{N}$ by the Kendall-tau similarities in Eq. (3.1) computed based on the model subset $\boldsymbol{\mathcal{M}}_s$. Then, we quantify the value of each candidate model that is not already in $\boldsymbol{\mathcal{M}}_s$ against the objective criterion and expand the model subset with the model M^* having the maximum value, i.e. $\boldsymbol{\mathcal{M}}_s := \boldsymbol{\mathcal{M}}_s \cup M^*$. In this way, we only need to fit on $\mathcal{D}_{\text{test}}$ (and get the corresponding IPMs) for the newly added model M^* per iteration. To reduce the overall computational cost, the goal is to accurately identify highly similar neighbors $\mathcal{N}$ based on as *few* models trained on $\mathcal{D}_{\text{test}}$ as possible. It is important to note, however, that even if we train only a small subset of the models on the new task, upon identifying $\mathcal{N}$, we select a model from among *all* candidate models using Eq. (3.2).

What objective criterion is suitable for iteratively choosing the next model to be included in the model subset $\boldsymbol{\mathcal{M}}_s$? We argue that the added model should meet two criteria, *uncertainty* and *quality*:

Criterion 1 (Uncertainty) *The performance (rank) of the added model should vary across the current neighbor set $\mathcal{N}$.*

Criterion 2 (Quality) *The added model should outperform on the current neighbor set $\mathcal{N}$.*

Notably, without *uncertainty*, $\mathcal{M}_s$ would end up choosing a group of similar models without enough representation of the full model space, which inhibits finding truly similar meta-tasks. A model with high performance variance over $\mathcal{N}$ indicates the datasets within the neighbor set exhibit disagreement (i.e. neighbors are not as similar among themselves), and including the model unlocks the opportunity to find truly similar meta-tasks. On the other hand, the *quality* criterion emphasizes that the added model should be a well-performing model on the neighbor set, as model selection mainly concerns "top models". As such, we aim to identify the neighbor set based on task similarity regarding the well-performing models, whereas performance similarity based on underperforming models does not contribute much to the main goal of (top) model selection.

Now it is easy to see the **trade-off** between *uncertainty* and *quality*—the former emphasizes the model's performance variation among the neighbor set while the latter expects high performance over all. This is akin to the explore-exploit trade-off; *uncertainty* drives exploration (for better neighbors) while *quality* drives exploitation (by promptly pinning a top model).

How can we quantify the *uncertainty* and *quality* of a candidate model (say, M_j)? Naturally, they can be respectively measured as the *variance* of the model's performance, $\sigma_j^2 = \sigma^2(M_j|\mathcal{N})$, and its *average* performance, $\mu_j = \mu(M_j|\mathcal{N})$, given the neighbor set. Thus, the simplest objective criterion that considers both *uncertainty* and *quality* criteria can be defined as the sum of the two:

$$\underset{M_j \in \mathcal{M} \setminus \mathcal{M}_s}{\arg\max} \underbrace{\sigma^2(M_j|\mathcal{N}))}_{\text{Uncertainty}} + \underbrace{\mu(M_j|\mathcal{N}))}_{\text{Quality}} \tag{3.6}$$

Can we define a better objective that automatically balances the trade-off between the two criteria? At this stage, we can recognize a connection to Sequential Model-based Bayesian Optimization (SMBO) [131]. As discussed in §3.2.1, SMBO is a state-of-the-art paradigm for solving sequential problems like hyperparameter optimization in supervised settings [276]. As an iterative method, it relies on what-is-called an *acquisition function* $a(\cdot)$ that quantifies the utility of a candidate hyper-

parameter configuration (HPC) for the next evaluation [131]. In fact, as with the uncertainty/exploration and quality/exploitation trade-off, $a(\cdot)$ typically aims to balance between picking an HPC from the unexplored regions of the hyperparameter space and one with high estimated accuracy.

To capitalize on this connection, ELECT leverages the prominent acquisition function in SMBO called Expected (positive) Improvement (EI) [131] as our objective criterion to automatically balance the uncertainty-quality trade-off. In our setting, EI measures the expected improvement of including a candidate model into the model subset. The high EI value of a candidate model means that it has large performance variation and also high performance over the neighbor set $\mathcal{N}$. Moreover, one of the nice properties of EI is it has a closed-form expression under the Gaussian assumption, where the EI of a candidate model is defined as:

$$EI(M_j|\mathcal{N}) := \sigma_j \cdot [u_j \cdot \Phi(u_j) + \phi(u_j)], \quad \text{where} \tag{3.7}$$
$$u_j = \begin{cases} \frac{\mu_j - \mu_s^*}{\sigma_j} & \text{if } \sigma_j > 0 \ ; \quad \text{and} \quad \begin{cases} 0 & \text{if } \sigma_j = 0 \end{cases} . \end{cases}$$

In the above, Φ and ϕ respectively denote the cumulative distribution and the probability density functions of standard Normal distribution, μ_j and σ_j are the mean and the standard deviation of model M_j across the neighbor set $\mathcal{N}$, and μ_s^* is the maximum value of the average model performance of $\boldsymbol{\mathcal{M}}_s$ over $\mathcal{N}$, i.e.,

$$\mu_s^* = \max_{M_b \in \boldsymbol{\mathcal{M}}_S} \frac{1}{|\mathcal{N}|} \sum_{\mathcal{D}_i \in \mathcal{N}} \mathbf{P}_{i,b} \ .$$

With the EI-based objective in Eq. (3.7), we include the highest EI candidate model to the model subset per iteration:

$$\boldsymbol{\mathcal{M}}_s := \boldsymbol{\mathcal{M}}_s \cup \underset{M_j \in \boldsymbol{\mathcal{M}} \setminus \boldsymbol{\mathcal{M}}_s}{\arg\max} \ EI(M_j|\mathcal{N}) \ . \tag{3.8}$$

To sum up, ELECT alternates between (*i*) updating the neighbor set $\mathcal{N}$ using Eq. (3.5) based on the (expanded) model subset $\boldsymbol{\mathcal{M}}_s$, and (*ii*) expanding $\boldsymbol{\mathcal{M}}_s$ using Eq. (3.8) based on the (updated)

$\mathcal{N}$, until converged or termination criteria are met.

3.4.2.3 Convergence and Termination Criteria

ELECT can operate under two different practical settings: (1) *hands-off*: there is no time budget and (2) *hands-on*: the user has time constraints and the algorithm is to output a selected model whenever prompted.

Hands-off: When there is *no* time budget, ELECT stops when the neighbor set $\mathcal{N}$ stays unchanged in p consecutive iterations, indicating that the identified similar meta-tasks have stabilized. We refer to parameter p as "patience", as a larger p requires more iterations to converge. In the extreme case (when p is set to a very large value), the algorithm would stop when all the models are added to the model subset, i.e. $\boldsymbol{\mathcal{M}}_s = \boldsymbol{\mathcal{M}}$. In this case, ELECT measures task similarity based on all models, falling back to the original setting (§3.4.1) *without* the adaptive expansion. p can be decided by cross-validation; see details in §3.5.2.1.

Hands-on: When there is a time budget b (iterations) to accommodate, ELECT stops the adaptive search when whichever one of two conditions occurs earlier: time budget is up, or patience criterion above is met. Note that the time budget need not be known to ELECT apriori, for it is an "*any-time* algorithm": at any time the user prompts it during its course, it can always output a selected model as the one with the highest average performance on the neighbor set at the current iteration, based on Eq. (3.2).

3.4.3 Computational Complexity

Suppose that a task has r samples and d features on average, and the score computation of an OD model takes $C_{\text{train}}(r, d)$.

Lemma 1 (Meta-training) *The computational complexity of* ELECT*'s meta-training phase (offline) is* $O(nmr + nm^2)$.

Meta-training involves (*i*) IPM generation (§3.3.2) for n meta-train tasks on all m models; MC, SELECT, and HITS have $O(nr)$, $O(nmr)$, $O(nmr)$, respectively; (*ii*) training of pairwise performance predictor (i.e., lightGBM, §3.3.3) uses the input data composed by n tasks, each with $\binom{m}{2}$ model pairs by enumeration, leading to $O(nm^2)$ complexity; and (*iii*) building the anchor set with forward selection (§3.5.1.1) takes $O(nmr)$. Overall runtime is $O(nmr + nm^2)$.

Lemma 2 (Model selection) *The computational complexity of* ELECT*'s model selection phase (online) is* $O[b(bn + mt + C_{train}(r, d))]$*, for budget b and neighbor count t.*

Model selection first initializes the model subset $\mathcal{M}_s$ with the coverage-driven strategy (§3.4.2.1), yielding $O(nm)$. For each initial model in $\mathcal{M}_s$, outlier score and IPM generation take $O(C_{\text{train}}(r, d) + r)$. In each of b iterations of adaptive search (§3.4.2.2), ELECT (*i*) predicts the model performance with $O(bn)$ complexity (*ii*) identifies the next model to be included in $\mathcal{M}_s$ with EI, taking $O(mt)$ and (*iii*) gets the next model's outlier scores and IPMs with $O(C_{\text{train}}(r, d) + r)$ runtime. The total selection runtime is $O[b(bn + mt + C_{\text{train}(r,d)})]$.

Note that the quadratic m^2 term in offline training is for measuring *pairwise* performance-driven task similarity, where m is small (e.g., 297 in this study). In the online phase, the complexity is linear in both the number of meta-train datasets (n) and that of candidate models (m).

3.5 Additional Design and Implementation Details

3.5.1 (Offline) Meta-training Details for §3.3

3.5.1.1 Building Internal Performance Measures (IPMs)

As described in §3.3.2 and 3.3.3, IPMs are used as the input features of the performance predictor $f(\cdot)$. In ELECT, we use three consensus-based IPMs (i.e., MC, SELECT, and HITS) as they are reported to carry useful signals in model selection [190]. Namely, consensus-based IPMs consider the resemblance to the overall consensus of outlier scores as a sign of a better model; their computation requires a group of models.

In [190], Ma *et al.* use all models in $\mathcal{M}$ for building IPMs, leading to high cost in generating outlier scores and then IPMs. To reduce the cost, we identify a small subset of representative models $\mathcal{M}_A \in \mathcal{M}$ called the *anchor* set (i.e., $|\mathcal{M}_A| \ll |\mathcal{M}|$), for calculating IPMs. That is, we generate the IPMs of a model with regard to its consensus to $\mathcal{M}_A$ rather than $\mathcal{M}$. Similar to forward feature selection [102], the anchor set can be identified in a forward fashion (i.e., iteratively expanding the set) and cross-validation on the meta-train database.

3.5.1.2 Pairwise Performance Predictor

As shown in §3.3.3, the predictor $f(\cdot)$ maps the vector of IPMs of a pair of models to their performance difference. To that end, we train a LightGBM regressor [135] by enumerating all $\binom{m}{2}$ model pairs for each task in meta-train database, where the hyperparameters are set by cross-validation.

3.5.2 (Online) Model Selection Details for §3.4

3.5.2.1 Hyperparameters

The hyperparameters are chosen by LOOCV on the meta-train set. To that end, we find the following settings works well in both testbeds: (*i*) the size of the neighbor set, $|\mathcal{N}| = t$, equals to 5 (§3.4.1) (*ii*) the initial size of the model subset, $|\boldsymbol{\mathcal{M}}_s|$, equals to 7 (§3.4.2.1) and (*iii*) patience p equals to 17 (§3.4.2.3).

3.5.2.2 Model Subset Initialization

Other than random sampling, we design a coverage-driven strategy for initializing $\boldsymbol{\mathcal{M}}_s$ in §3.4.2.1. Intuitively, we expect $\boldsymbol{\mathcal{M}}_s$ provides differentiability among datasets—the models' performance in $\boldsymbol{\mathcal{M}}_s$ on different datasets should vary. To this end, the coverage-driven strategy iteratively builds the initial $\boldsymbol{\mathcal{M}}_s$ by including the model that performs best (a top model) or worst (a bottom model) on the most meta-train tasks that have *not* been "covered". A task is said "covered" if both its top and bottom models (at least one) are already included in the model subset.

3.6 Experiments

3.6.1 Experiment Setting

Model Set. We configure 8 leading OD algorithms with various different settings of their associated hyperparameters to compose the model set $\boldsymbol{\mathcal{M}}$ with 297 models (based on MetaOD [348]; the only diff. is we set n_neighbors of ABOD to $[3, 5, 10, 15, 20, 25, 50]$ for faster experimentation). We evaluate ELECT in two testbeds introduced below, with 39 and 30 datasets, respectively. For each testbed, we first generate the outlier scores of each model in $\boldsymbol{\mathcal{M}}$ on each dataset, and then record the historical performance matrix $\mathbf{P}$. For models with built-in randomness, e.g., iForest and LODA

with random feature splits, we run 5 random trials and record the average. All OD models are built using the PyOD library [347] on an Intel i7-9700 @3.00 GHz, 64GB RAM, 8-core workstation.

Testbeds. The key mechanism of meta-learning is to leverage the prior knowledge from truly similar tasks—where OD models perform similarly on both tasks. We create two testbeds with varying performance-driven task similarities (see Fig. 3.2).

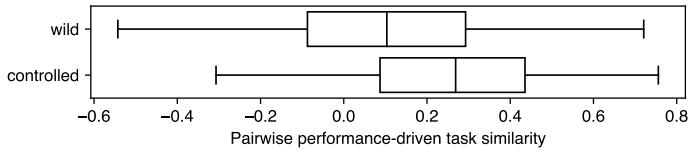


Figure 3.2: Pairwise performance-driven task similarity in the wild (med.=0.1035; lower similarity) and controlled (med.=0.2688; higher similarity) testbed.

(1) **Wild testbed** contains 39 independent datasets from 2 public OD repository (i.e., ODDS [245] and DAMI [51]) which simulates real-world use cases.

(2) **Controlled testbed** contains 30 datasets generated from 10 independent "mothersets" (from the Wild testbed), where 3 types of outliers (global, local, clustered) are injected into each motherset. As such, higher performance-driven task similarities are expected in the datasets with the same type of injected outliers (but from different mothersets), while their meta-feature similarities are low due to the independence of the mothersets. This testbed helps understand (*i*) if meta-feature methods can work when meta-feature similarity disagree with performance-driven similarity and (*ii*) to what extent the level of task similarity affects the performance of ELECT.

Baselines. We include 13 baselines for comparison. As shown in Table 3.1, they can be categorized by (i) whether it selects a model; (ii) whether it is based on meta-learning; and (iii) whether it relies on meta-features. We use all 13 baselines in the wild testbed, and the 5 meta-feature-based methods in the controlled testbed as it is built to contrast performance- vs. meta-feature-based task similarities.

Table 3.1: 13 baselines for comparison with categorization by (first row) whether it is a model selection method (second row) whether it uses meta-learning and (third row) whether it relies on meta-features (last row).

Category	iForest	LOF	ME	MC	SELECT	HITS	GB	ISAC	AS	ALORS	MetaOD	SS	IPM_SS
model selection			•	•	•		•	•	•	•	•	•	•
meta-learning							•	•	•	•	•	•	•
meta-features								•	•	•	•	•	

Briefly, the baselines are organized as: (*i*) ***no model selection***: directly/always use the same popular model **(1) iForest** [180] or **(2) LOF** [46], or the ensemble of all models **(3) Mega Ensemble (ME)**; (*ii*) ***direct use of IPMs for model selection***: **(4) MC** [190], **(5) SELECT** [190], and **(6) HITS** [190]; and (*iii*) ***meta-learning based methods***: **(7) Global Best (GB)** selects the best performing model on meta-train database on average, **(8) ISAC** [132], **(9) ARGOSMART (AS)** [217], **(10) ALORS** [206], **(11) MetaOD** [348] is the SOTA method, **(12) Supervised Surrogates (SS)** [325] directly regresses meta-features to performance $\mathbf{P}$, and **(13) IPM-based Supervised Surrogates (IPM_SS)** is a variant of SS but uses IPMs other than meta-features. Baselines (8)-(12) use meta-features.

Evaluation. In both testbeds, we use leave-one-out cross-validation (LOOCV) to split the meta-train/test. Each time we use one dataset as the input task, and the remaining datasets as meta-train. Meanwhile, we use cross-validation to decide the size of the neighbor set $\mathcal{N}$ for each input task, and use a fixed time budget $b = 50$ as the convergence criteria. We use the area under the precision-recall curve (Average Precision or AP) as the performance measure, while it can be substituted with any other measures, e.g., the area under the receiver operating characteristic curve (ROC). Since the raw performance like AP is not comparable across datasets with varying magnitude, we report the AP-rank of a selected model, ranging from 1 (the best) to 297 (the worst)—thus smaller the better. To compare two methods, we use the paired Wilcoxon signed rank test across all datasets in the testbed (significance level $p < 0.05$).

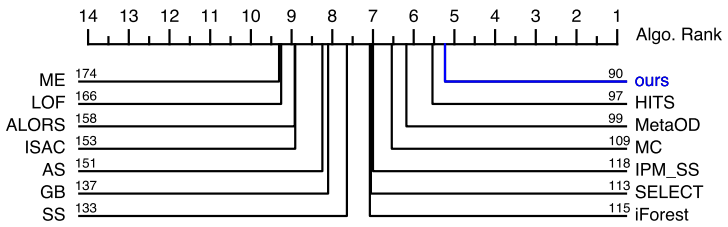


Figure 3.3: Comparison of avg. rank (lower is better) of algorithms w.r.t. performance across datasets in the wild testbed. ELECT outperforms all w/ the lowest avg. rank. Numbers on each line are the avg. AP-rank (lower is better) of the employed model (selected or otherwise) by each method.

3.6.2 Experiment Results

3.6.2.1 Results on Wild Testbed

As shown in Fig. 3.3, ELECT **outperforms all baseline algorithms with the lowest avg. rank** "in the wild". Furthermore, Table 3.2 (left) shows that ELECT is the only algorithm that is not significantly different from the 55-*th* best model. In other words, ELECT can consistently choose the top 18.5% model from a large pool of 297 models. Moreover, ELECT is significantly better than the no model selection baselines, LOF, iForest, ME, and other meta-learning baselines including GB, ISAC, ALORS, SS, and IPM_SS. For other baselines, p-value remains low although not significant at 0.05.

ELECT **achieves the best performance with small (<1 min.) overhead**. Fig. 3.4 shows the running time of the methods versus the avg. AP-rank of the employed model. Based on the avg. selection time per dataset, the methods can be categorized as i) super-fast methods that take less than 1 sec. (red zone); ii) fast methods that take less than 1 min. (blue zone); and iii) slow methods that use up to 10 min.s (green zone). Super-fast methods either directly employ a model (iForest, LOF) or simply report the historical best model (GB) with limited performance, showing the necessity for more effective meta-learning. In time-critical applications, employing iForest is a reasonable choice and it is indeed on the Pareto frontier of the time-performance trade-off. In the fast group, both MetaOD and ELECT are also on the Pareto frontier, showing the premise of effective meta-

Table 3.2: Pairwise statistical tests between ELECT and baselines by Wilcoxon signed rank test (statistically better method at $p<0.05$ in **bold**, both in **bold** if no difference). In wild testbed (left), ELECT is the only approach with no difference from the 55-th best model. In controlled testbed (right), compared with meta-feature based baselines, ELECT is the only method with no difference from the 32-th best model, and statistically better than all baselines.

Ours	Baseline	p-value
ELECT	**55-th Best**	0.0541
ELECT	iForest	0.0008
ELECT	LOF	0.0004
ELECT	ME	0.0188
ELECT	**MC**	0.137
ELECT	SELECT	0.0484
ELECT	**HITS**	0.2142
ELECT	GB	0
ELECT	ISAC	0
ELECT	AS	0.0147
ELECT	ALORS	0.0002
ELECT	**MetaOD**	0.2766
ELECT	SS	0.0128
ELECT	IPM_SS	0.0019

Ours	Baseline	p-value
ELECT	**32-th Best**	0.0631
ELECT	iForest	N/A
ELECT	LOF	N/A
ELECT	ME	N/A
ELECT	MC	N/A
ELECT	SELECT	N/A
ELECT	HITS	N/A
ELECT	GB	N/A
ELECT	ISAC	0.0012
ELECT	AS	0.003
ELECT	ALORS	0.0001
ELECT	MetaOD	0.0016
ELECT	SS	0.0007
ELECT	IPM_SS	N/A

learning and the additional benefit of ELECT. Specifically, ELECT (avg. AP-rank=90) brings 10% performance improvement over MetaOD (avg. AP-rank=99), while being fast (avg. time=47.10s). For the slow group, in contrast, the higher runtime for model and IPM building does not yield improved performance over the fast methods.

IPMs do carry useful signals, and ELECT **can leverage them more effectively**. In fact, IPM-based MC, SELECT, and HITS rank high among all methods (Fig. 3.3), showing their potential in model selection. However, using IPMs directly for model selection incurs a large overhead in building all OD models and IPMs themselves at selection time, and therefore all of them fall in the slow group as shown in Fig. 3.4. Building on top of IPMs, ELECT shows superior results by "juicing out" useful information from IPMs via meta-learning, as well as reducing the runtime via sequential learning to prevent excessive model building.

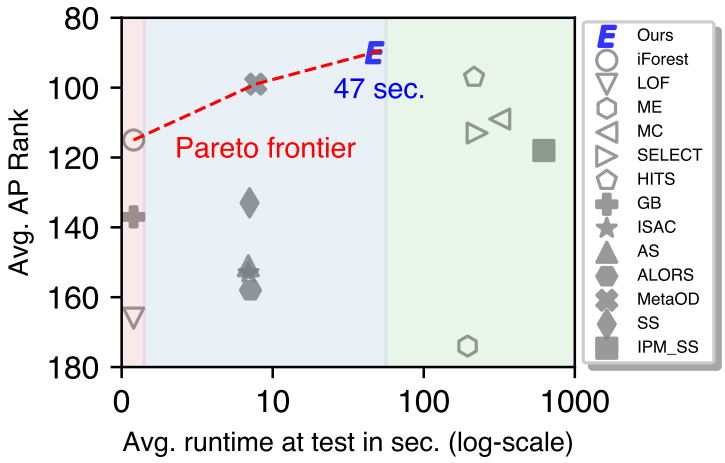


Figure 3.4: Avg. running time (log-scale) vs. avg. model AP-rank. Meta-learning methods depicted w/ solid markers. Based on runtime, methods are categorized as i) super-fast (in ■), ii) fast (in ■), and iii) slow (in ■). Pareto frontier (red dashed line) shows the best method under different time budgets. ELECT outperforms all with small time consumption (on avg. below 1 min. per task).

3.6.2.2 Results on Controlled Testbed

Setup details. Meta-learning facilitates model selection for a new task by leveraging the prior knowledge from its truly similar meta-train tasks—where OD models *perform* similarly. The controlled testbed is built to create the scenario where there exist meta-train tasks with *high performance-driven similarity* but *low meta-feature similarity*, and vice versa. As meta-feature methods assume a high correlation between meta-feature similarity and task-performance similarity, they are likely to do poorly (i.e. select poor models) in this testbed as the assumption is violated.

To create such a setting, we randomly select 10 independent "mothersets" from the wild testbed, and inject one of 3 types of synthetic outliers (global, local, and clustered) by following [281], resulting in 30 datasets. Intuitively, different OD models are good at successfully detecting different types of outliers, irrespective of the underlying motherset. Therefore, we expect the datasets from different mothersets but with the same type of injected outliers to have high performance-driven task similarity yet low meta-feature similarity, while the datasets from the same mothersets but with

different types of injected outliers to be vice versa.

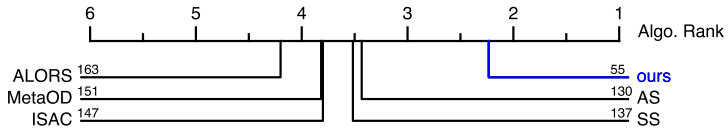


Figure 3.5: Comparison of avg. rank (lower is better) of algorithms w.r.t. performance across datasets in the controlled. ELECT outperforms all baselines.

Results. As shown in Fig. 3.5, ELECT **is superior to all meta-feature-based baselines**. Note that its avg. AP-rank is 55, while the avg. AP-rank of meta-feature baselines are all above 130. All these differences are significant at $p < 0.005$ as shown in Table 3.2 (right). These results together show that the performance of meta-feature baselines suffers when performance-driven similarity is not correlated with meta-feature similarity. Distinctly, ELECT does not rely on meta-features and instead directly focuses on performance-driven similarity, leading to superior selection results.
As a meta-learning method, ELECT **achieves better results with higher task similarity to meta-train**. As shown in Fig. 3.2, the controlled testbed has higher task similarity than the wild. In Table 3.2 (right) ELECT is the only method that shows no statistical difference from the 32-*th* best model, suggesting it can choose the top 11% model from $\mathcal{M}$, an improvement from the top 18.5% in the wild testbed. The selected models' avg. AP-rank reflects the same—55 vs. 90 in the controlled and wild testbed, respectively.

3.6.3 Case Study

It is interesting to trace how ELECT works over iterations on a given dataset, as shown in Table 3.3 for the Waveform dataset from the wild testbed. First we track the changes in the neighbor set: col. 2 reports the avg. similarity between Waveform and the identified neighbor set $\mathcal{N}$, and col. 3 shows the number of ground-truth top 5 most similar meta-train tasks in $\mathcal{N}$. ELECT gradually identifies both more similar and more of the top 5 meta-train tasks. From 1-*st* to the 50-*th* iteration, the avg.

task similarity improves from 0.2276 to 0.3504., and the number of identified top 5 neighbors increases from 1 to 4. Note that out of the 38 meta-train tasks, only 7 of them have higher similarity than 0.3504. By identifying more and more similar meta-train tasks, ELECT gradually converges to kNN models as given in col. 4, which is indeed the best algorithm family for Waveform. Moreover, col. 5 shows that ELECT successfully identifies better models—the selected model's AP-rank decreases from 77 to 38.

Table 3.3: Trace of ELECT on Waveform dataset. Over iterations (col. 1), ELECT gradually identifies more similar meta-train tasks with increasing avg. similarity to Waveform (col. 2), more ground-truth top 5 neighbors (col. 3), and a better selected model with lower rank (col. 4). Best performing algorithm family on Waveform is kNN, which ELECT successfully identifies during its adaptive search.

Iter.	Avg. Sim.	# Matched neighbors	The selected model	The selected model rank
1	0.2776	1	('LOF', ('manhattan', 5))	77
2	0.2876	1	('COF', 10)	70
...	...	...	...	...
11	0.3304	2	('LOF', ('manhattan', 10))	76
12	0.3493	3	('LOF', ('euclidean', 20))	71.5
...	...	...	...	...
21	0.4351	4	('kNN', ('mean', 5))	50
22	0.4351	4	('kNN', ('mean', 5))	50
...	...	...	...	...
31	0.2960	4	('kNN', ('mean', 15))	38
32	0.4351	4	('kNN', ('mean', 5))	50
...	...	...	...	...
49	0.3504	4	('kNN', ('mean', 15))	38
50	0.3504	4	('kNN', ('mean', 15))	38

3.6.4 Ablation Studies and Other Analysis

3.6.4.1 Model initialization

ELECT uses proposed coverage-driven initialization of the model subset $\mathcal{M}_s$ (see §3.5.2.2). Fig. 3.6 shows that its AP-rank (median=87.5) is notably lower than that of random initialization (median=133). The difference, by one-sided Wilcoxon signed rank test, is statistically significant at $p = 0.0204$.

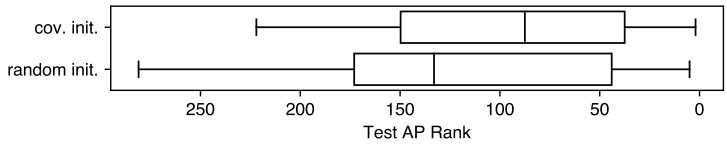


Figure 3.6: Ablation of coverage init. (med.=87.5) vs. random init. (med.=133).

3.6.4.2 The Effect of Model Inclusion Criteria

Fig. 3.7 shows the performance comparison between using EI (balancing both exploitation and exploration) in Eq. (3.7) and the greedy objective (exploitation only) that adds the model with the highest performance on $\mathcal{N}$ during adaptive search (see §3.4.2.2). One-sided Wilcoxon signed rank test shows that the former (median=87.5) is statistically better (at $p = 0.0203$) than the latter (median=105), justifying the use of EI.

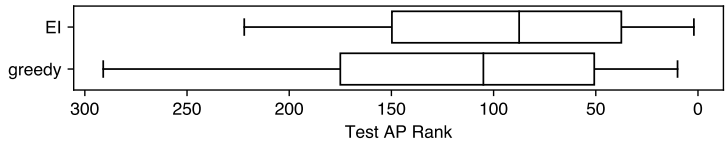


Figure 3.7: Ablation of using EI (med.=87.5) vs. greedy without exploration (med.=105) during the adaptive search.

3.7 Discussions

In the face of numerous outlier detection algorithms with various hyperparameters, there exists a shortage of principled approaches to *unsupervised* outlier model selection—a vastly understudied subject. To fill this gap in the literature, we proposed ELECT, a meta-learning approach that selects a candidate model for a new task based on its *performance-based similarity* to historical (meta-train) tasks. ELECT adaptively identifies these neighbor tasks, as such, it can flexibly output a selected model in an any-time fashion, accommodating varying time budgets. Through extensive experiments, we showed that ELECT significantly outperforms a wide range of prior as well as more recent baselines. Future work includes extending to UOMS for deep learning based outlier models.

4

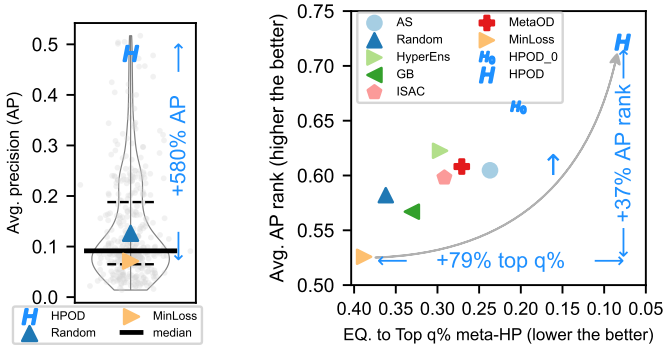
HPOD: Hyperparameter Optimization for Unsupervised Outlier Detection

This chapter is primarily based on:

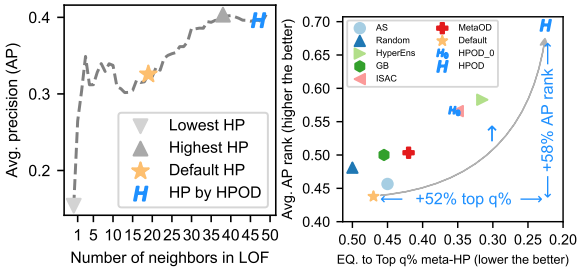
Yue Zhao*, Leman Akoglu. "Hyperparameter Optimization for Unsupervised Outlier Detection." *arXiv preprint arXiv:2208.11727*, 2023.

4.1 Highlight

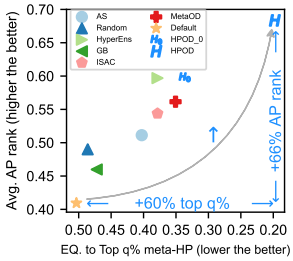
Given an unsupervised outlier detection (OD) algorithm, how can we optimize its hyperparameter(s) (HP) on a *new dataset, without any labels*? In this chapter, we address this challenging hyperparameter optimization for unsupervised OD problem, and propose *the first systematic approach* called HPOD that is based on meta-learning.



(a) (left) large perf. variation over 384 HPs for **RAE** on `Thyroid`—random&min. loss HPs are sub-par; (right) HPOD (***H***) outperforms all.



(b) (left) HP sensitivity of **LOF** on `Vowels`—the default HP is far from optimal; (right) HPOD (***H***) outperforms all baselines with +58% AP rank over random.



(c) Performance comparison on ensemble **iForest**; HPOD (***H***) outperforms all.

Figure 4.1: (*a*) (left) HP sensitivity in deep RAE on `Thyroid`; (right) HPOD outperforms all baselines on a 39-dataset database (§4.4.2.1), with a higher avg. performance rank (y-axis) and comparable to better top q% HP settings in the meta-HP set (x-axis) (*b*) (left) HP sensitivity in LOF on `Vowels`; (right) HPOD outperforms all baselines with huge improvement (e.g., $+58\%$ norm. AP rank) over the default HP (§4.4.2.2) and (*c*) for iForest, HPOD is the best (e.g., $+66\%$ normalized AP rank) over the default HP (§4.4.2.2). See detailed experiment results in §4.4.

Moving beyond proposing another detection algorithm, we study this important HyperParameter

Optimization for unsupervised OD problem, and introduce a systematic approach called HPOD. In a nutshell, HPOD leverages meta-learning to enable (originally supervised) SMBO for efficient unsupervised OD hyperparameter optimization. To overcome the infeasibility of evaluation in unsupervised OD, HPOD uses meta-learning that carries over past experience on prior datasets/tasks to more efficient learning on a new task. To that end, we build a meta-database with historical performances of a large collection of HPs on an extensive corpus of existing OD benchmark datasets, and train a *proxy performance evaluator* to evaluate HPs on a new dataset *without labels* (see §4.3.2). With the evaluator, HPOD can iteratively and efficiently identify promising HPs to evaluate and output the best (see §4.3.3). Also, we use meta-learning to further facilitate HPOD by *initializing* and *transferring knowledge* from similar historical tasks to the surrogate function of the new task (see §4.3.4). We find it important to remark that HPOD is strictly an HPO method other than a new detection algorithm.

Performance. Fig. 4.1a (left) shows the huge performance variation (up to 37×) for a set of 384 deep RAE HPs on `Thyroid` data, where HPOD is significantly better than expectation (i.e. random selection), as well as selection by min. reconstruction loss (MinLoss); ours is one of the top HPs. In Fig. 4.1a (right), we show that HPOD is significantly better than a group of diverse and competitive baselines (see Table 4.1) on a 39 dataset database. We also demonstrate HPOD's generality on non-deep OD algorithm LOF with both discrete and continuous HP spaces in Fig. 4.1b, as well as the popular iForest in Fig. 4.1c. For all three OD algorithms, HPOD is *statistically* better than (most) baselines, including the default HPs of widely used LOF and iForest. In fact, being an ensemble, iForest has been shown to be robust to HPs and outperform many other detectors [76]. Thus, HPOD's improvement over its default HPs is remarkable.

We summarize the key contributions as follows:

- **Novel HPO Framework for Unsupervised Outlier Detection**. We introduce HPOD, a meta-learning approach that capitalizes on historical OD tasks with labels to select effective

HPs for a new task without any labels.

- **Continuous Search**. Superior to all baselines in Table 4.1, HPOD works with both discrete *and* continuous HPs.
- **Generality and Effectiveness**. Extensive results on 39 datasets with (*a*) deep method RAE and classical methods (*b*) LOF and (*c*) iForest show that HPOD outperforms baselines, with an avg. 37%, 58%, and 66% improvement over the minimal loss HPs of RAE and the default HPs of LOF and iForest.

Reproducibility. We open-source HPOD and the meta-train database (anonym.) at https://github.com/review2023/hpod.

4.2 Related Work

4.2.1 Hyperparameter Optimization (HPO) for OD

Category	Default	Random	MinLoss	HE	GB	ISAC	AS	MetaOD	HPOD_o	HPOD
meta-learning	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓
discrete HP	✗	✗	✓	✗	✓	✓	✓	✓	✓	✓
continuous HP	✗	✗	✓	✗	✗	✗	✗	✗	✓	✓

Table 4.1: HPOD and baselines for comparison with categorization by (1st row) whether it uses meta-learning and (2nd & 3rd row) whether it supports *discrete* and *continuous* HPO. Only HPOD and HPOD_o leverage meta-learning and support continuous HPO. See §4.4.1.

We can categorize the short list of HPO methods for OD into two groups. The first group of methods require a hold-out set with ground truth labels for evaluation [30], including AutoOD [172], TODS [150], and PyODDS [173], which do not apply to unsupervised OD. The second group uses the default HP, randomly picking an HP, or averaging the outputs of randomly sampled HPs [309]; we include them as baselines (see col. 2-5 of Table 4.1) with empirical results in §4.4.2.

4.2.2 Hyperparameter Opt. and Meta-Learning

HPO gains great attention due to its advantages in searching and optimizing through complex HP spaces, where learning tasks are costly [134]. Existing methods include simple grid and random search [34] and more efficient Sequential Model-based Bayesian Optimization (SMBO) [131]. Notably, SMBO builds a cheap regression model (called the surrogate function) of the expensive objective function, and uses it to iteratively select the next promising HPs to be evaluated by the objective function (see [342] Appx. A). We cannot directly use these supervised methods for OD. Rather, HPOD leverages meta-learning to enable efficient SMBO for *HPO for OD*.

Other than using (external) ground truth labels, a small number of studies have employed (internal) unsupervised strategies that solely make use of the input data and/or output outlier scores for model evaluation [197, 91, 216, 196, 58]. However, a recent survey [190] shows only very few of these internal strategies perform better than random in model selection—they are generally inferior to meta-learning-based approaches [349].

Meta-learning aims to facilitate new task learning by leveraging and transferring knowledge from prior/historical tasks [292], which has been used in warm-starting [84] and transferring surrogate [332, 314, 316] in SMBO. Recently, meta-learning has also been applied to unsupervised outlier model selection (UOMS), where Zhao *et al.* proposed MetaOD [348] with comparison to baselines including global best (GB), ISAC [132], and ARGOSMART (AS) [217]. We adapt these UOMS methods as baselines for *HPO for OD* (see col. 6-9 of Table 4.1). Although these methods leverage meta-learning, they cannot handle continuous HPO. HPOD outperforms them in all experiments (see §4.4.2).

4.3 HPOD: Hyperparameter Optimization for Unsupervised Outlier Detection

4.3.1 Overview of HPOD

In HPOD, we use *meta-learning* to enable (originally supervised) Sequential Model-based Bayesian Optimization for efficient *HPO for OD* (see definition in §0.3.2), where the key idea is to transfer useful information from historical tasks to a new test task. As such, HPOD takes as input a collection of historical tasks $\boldsymbol{\mathcal{D}}_{\text{train}} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$, namely, a meta-train database with ground-truth labels where $\{\mathcal{D}_i = (\mathbf{X}_i, \mathbf{y}_i)\}_{i=1}^{n}$. Given an OD algorithm M for HPO, we define a finite meta-HP set by discretizing continuous HP domains (if any) to get their cross-product, i.e., $\lambda_{\mathbf{meta}} = \{\lambda_1, \ldots, \lambda_m\} \in \Lambda$. We use M_j to denote detector M with the j-th HP setting $\lambda_j \in \lambda_{\mathbf{meta}}$. HPOD uses $\boldsymbol{\mathcal{D}}_{\text{train}}$ to compute:

- the historical output scores of each detector M_j on each meta-train dataset $\mathcal{D}_i \in \boldsymbol{\mathcal{D}}_{\text{train}}$, where $\mathcal{O}_{i,j} := M_j(\mathcal{D}_i)$ refers to the output outlier scores using the j-th HP setting for the points in the i-th meta-train dataset $\mathcal{D}_i$; and
- the historical performance matrix $\mathbf{P} \in \mathbb{R}^{n \times m}$ of each detector M_j, where $\mathbf{P}_{i,j} := \text{perf}(\mathcal{O}_{i,j})$ is M_j's performance on meta-train dataset $\mathcal{D}_i$.

HPOD consists of two phases. During **the (offline) meta-learning**, it leverages meta-train database with labels to build a *proxy performance evaluator* (PPE), which can predict HP performance on a new task without labels. Also, it trains a *meta-surrogate function* (MSF) for each meta-train dataset to facilitate later HPO on a new dataset. In the **(online) HP Optimization** for a new task, HPOD uses PPE to predict its HPs' performance without using any labels, under the SMBO framework to identify promising HP settings in iteration effectively. Also, we improve the surrogate function in HPOD by transferring knowledge from *similar* meta-train datasets. An outline

Algorithm 3 HPOD: Offline and Online Phases

Input: (Offline) meta-train database $\boldsymbol{\mathcal{D}}_{\text{train}} = \{(\mathbf{X}_i, \mathbf{y}_i)\}_{i=1}^{n}$, OD algorithm M, meta-HP set $\lambda_{\mathbf{meta}} = \{\lambda_1, \ldots, \lambda_m\} \in \Lambda$, performance evaluation perf$(\cdot)$; (Online) new OD dataset $\mathcal{D}_{\text{test}} = (\mathbf{X}_{\text{test}}, \emptyset)$ (no labels), number of iterations E

Output: (Offline) HPOD meta-learners; (Online) the selected hyperparameter setting λ^* for $\mathcal{D}_{\text{test}}$

▶ (Offline) **Meta-train**: *Learn functions for HP performance prediction* **(§4.3.2)**

1: Train detector M with HP $\lambda_j \in \lambda_{\mathbf{meta}}$ on each $\mathbf{X}_i$ of $\mathcal{D}_i \in \boldsymbol{\mathcal{D}}_{\text{train}}$ to get outlier scores $\mathcal{O}_{i,j}$, $\forall\, i = 1 \ldots n,\ j = 1 \ldots m$

2: Evaluate each $\mathcal{O}_{i,j}$ against true labels $\mathbf{y}_i$ to get performance matrix $\mathbf{P} \in \mathbb{R}^{n \times m}$, where $\mathbf{P}_{i,j} :=$ perf$(\mathcal{O}_{i,j}|\mathbf{y}_i)$

3: Extract meta-features (MF) per task, $\mathbf{m}_i := \psi(\mathbf{X}_i)$

4: Compute internal perf. measures (IPM), $\mathbf{I}_{i,j} := \varphi(\mathcal{O}_{i,j})$

5: Train *proxy performance evaluator* (PPE) $f(\cdot)$ to predict the performance $\mathbf{P}_{i,j}$ from the respective {HP Settings λ_j, meta-features $\mathbf{m}_i$, IPMs $\mathbf{I}_{i,j}$} ▶ §4.3.2.1

6: Train each *meta-surrogate function* (MSF) $t(\cdot)$ per meta-train dataset $\boldsymbol{\mathcal{T}} = \{t_1, \ldots, t_n\}$ to predict the perf. $\mathbf{P}_{i,j}$ from only the respective {HP setting λ_j} ▶ §4.3.2.2

7: **Save** MF extractor ψ, IPM extractor φ, PPE f, and MSF $\boldsymbol{\mathcal{T}}$

▶ (Online) **HPO on a new task**: *Iteratively identify promising HP settings and output the best one* **(§4.3.3)**

8: Extract meta-features of the test task $\mathbf{m}_{\text{test}} := \psi(\mathbf{X}_{\text{test}})$

9: Initialize surrogate function $s^{(1)}$ and the evaluation set λ_{eval} by the meta-train database and PPE ▶ §4.3.3.1

10: **for** $e = 1$ to E **do** ▶ §4.3.3.2

11: Transfer meta-surrogate functions $\boldsymbol{\mathcal{T}}$ to surrogate $s^{(e)}$ by performance similarity to meta-train ▶ §4.3.4.2

12: Get the promising HP to evaluate by EI on surrogates' prediction, $\lambda^{(e)} := \arg\max_{\lambda_k \in \lambda_{\mathbf{sample}}} EI(\lambda_k|s^{(e)})$

13: Build M with $\lambda^{(e)}$, and get the corresponding outlier scores $\mathcal{O}_{\text{test}}^{(e)}$ and IPMs $\boldsymbol{\mathcal{I}}_{\text{test}}^{(e)}$

14: Predict performance of $\lambda^{(e)}$ with $f(\cdot)$, i.e., $\widehat{\mathbf{P}}_{\text{test}}^{(e)} := f(\lambda^{(e)}, \mathbf{m}_{\text{test}}, \boldsymbol{\mathcal{I}}_{\text{test}}^{(e)})$

15: Add $\lambda^{(e)}$ to the evaluation set $\lambda_{\text{eval}} := \lambda_{\text{eval}} \cup \lambda^{(e)}$

16: Update to $s^{(e+1)}$ with new pairs of info. $\langle \lambda^{(e)}, \widehat{\mathbf{P}}_{\text{test}}^{(e)} \rangle$

17: **end for**

18: **Output** $\lambda^* \in \lambda_{\text{eval}}$ with the highest predicted performance

of HPOD is given in Algo. 3, where we elaborate on the details of offline meta-training and online *HPO for OD* on a new task in §4.3.2 and §4.3.3.

4.3.2 (Offline) Meta-training

In principle, meta-learning carries over the prior experience of historical (meta-train) tasks to do better on a new task, given the latter at least resembles some of the historical tasks. Due to the lack of ground truth labels and/or a reliable internal objective function, the key challenge in *HPO for OD* is to evaluate the performance of HP settings. Thus, the core of HPOD's meta-learning is *learning* the mapping from *HP settings* onto *ground-truth performance* by the *supervision* from the meta-train database. The first part (lines 1-7) of Algo. 3 describes the core steps, and we further discuss on how to learn this mapping (§4.3.2.1) and transfer additional information for a new task (§4.3.2.2) in the following. Notably, *offline* meta-training is *one-time* and amortized over many test tasks.

4.3.2.1 Proxy Performance Evaluator (PPE)

In HPOD, we learn a regressor $f(\cdot)$ across all meta-train datasets, named *Proxy Performance Evaluator* (PPE), that maps their {`HP settings`, `data characteristics`, `additional signals`} onto ground truth performance. If $f(\cdot)$ only uses HP settings as the input feature, it fails to capture the performance variation of an HP across meta-train datasets. We need additional input features to enable $f(\cdot)$ for quantifying dataset similarity, so that $f(\cdot)$ makes similar HP performance predictions on similar datasets, and vice versa.

How can we capture dataset similarity in OD? Recent work by [348] introduced specialized OD meta-features (MF) to describe general characteristics of OD datasets; e.g., number of samples, basic feature statistics, etc. With the meta-feature extractor, both meta-train datasets and (later) the test dataset can be expressed as fixed-length vectors, and thus any similarity measure applies, e.g., Euclidean distance. To build $f(\cdot)$, we extract meta-features from each meta-train dataset as $\mathbf{M} = \{\mathbf{m}_1, \ldots, \mathbf{m}_n\} = \psi(\{\mathbf{X}_1, \ldots, \mathbf{X}_n\}) \in \mathbb{R}^{n \times d}$, where $\psi(\cdot)$ is the extraction module, and d is the dimension of meta-features (see [348] for details).

Although meta-features describe general characteristics of OD datasets, their similarity does not necessarily correlate with the *actual performance*. Thus, we enrich the input features of $f(\cdot)$ with internal performance measures (IPMs) [190], which are more "performance-driven". IPMs have been proven effective in unsupervised OD model selection [349]. More specifically, IPMs are noisy/weak unsupervised signals that are solely based on the input samples and/or a given model's output (e.g., outlier scores) that can be used to compare two models [91, 196]. In HPOD, we make the best use of these weak signals by *learning* in $f(\cdot)$ to regress the IPMs of a given HP setting (along with other signals) onto its true performance *with supervision*. To build $f(\cdot)$, we extract IPMs of each detector M_j with HP setting $\lambda_j \in \lambda_{\mathbf{meta}}$ on each meta-train dataset $\mathcal{D}_i \in \boldsymbol{\mathcal{D}}_{\text{train}}$, where $\mathbf{I}_{i,j} := \varphi(\mathcal{O}_{i,j})$ refers to the IPMs using the j-th HP for the i-th meta-train dataset, and $\varphi(\cdot)$ is the IPM extractor. See more of IPMs in Chapter 1.

Putting these together, we build *Proxy Performance Evaluator* $f(\cdot)$ to map {`HP setting`, `meta-features`, `IPMs`} of HP $\lambda_j \in \lambda_{\mathbf{meta}}$ on the i-th meta-train dataset onto its ground truth performance, i.e., $f(\lambda_j, \mathbf{m}_i, \mathbf{I}_{i,j}) \mapsto \mathbf{P}_{i,j}$. We provide details of $f(\cdot)$ in [342] Appx. B.2.

We find it critical to remark that provided $\psi(\cdot)$, $\varphi(\cdot)$, and the trained $f(\cdot)$ at test time, predicting the detection performance of HP settings becomes possible for the new task *without* using any ground-truth labels.

4.3.2.2 Meta-Surrogate Functions (MSF)

Different from $f(\cdot)$ that trains on *all* meta-train datasets and leverages *rich input features* (i.e., HPs, MFs, and IPMs) to predict HP performance, we also train *n independent* regressors with *only HPs as input*, $\boldsymbol{\mathcal{T}} = \{t_1, \ldots, t_n\}$. That is, for each meta-train dataset $\mathcal{D}_i \in \boldsymbol{\mathcal{D}}_{\text{train}}$, we train a regressor $t_i(\cdot)$ that simply maps the j-th HP setting $\lambda_j \in \lambda_{\mathbf{meta}}$ to its detection performance on the i-th meta-train dataset, i.e., $t_i(\lambda_j) \mapsto \mathbf{P}_{i,j}$.

Since these independent regressors only use HP settings as input, they can be readily transferred

to the online HPO phase to improve HP performance evaluation on $\mathbf{X}_{\text{test}}$. We defer their specific usage to §4.3.3.2 and §4.3.4.2.

4.3.3 (Online) HPO on a New OD Task

After the meta-training phase, HPOD is ready to optimize HPs for a new dataset. In short, it outputs the HP with the highest predicted performance by $f(\cdot)$, the trained performance evaluator (§4.3.3.1). To explore better HPs efficiently, HPOD leverages Sequential Model-based Optimization to iteratively select promising HPs for evaluation (§4.3.3.2). The second part (lines 8-18) of Algo. 3 shows the core steps.

4.3.3.1 Hyperparameter Optimization via Proxy Performance Evaluator

Given a new dataset $\mathcal{D}_{\text{test}}$, we can sample a set of HPs (termed as the evaluation set $\lambda_{\text{eval}} \in \Lambda$), and use the *proxy performance evaluator* $f(\cdot)$ from meta-training to predict their performance, based on which we can output the one with the highest predicted value as follows.

$$\arg\max_{\lambda_k \in \lambda_{\text{eval}}} f(\lambda_k, \mathbf{m}_{\text{test}}, \mathbf{I}_{\text{test},k}) \tag{4.1}$$

By setting λ_{eval} to some *randomly sampled HPs* and plugging it into Eq. (4.1), we have the "version 0" of HPOD, referred as HPOD_0. However, $f(\cdot)$ needs IPMs (i.e., $\mathbf{I}_{\text{test},k}$ in Eq. (4.1)) as part of the input, requiring detector building at test time. Thus, we should construct λ_{eval} carefully to ensure it captures promising HPs, where random sampling is insufficient. Thus we ask: how to efficiently identify promising HPs for model building and evaluation at test time?

4.3.3.2 Identifying Promising HPs by Sequential Model-based Optimization (SMBO)

As we briefly described in §4.2, SMBO can iteratively optimize an expensive objective [120], and has been widely used in supervised model selection and HPO [35]. Other than sampling HPs randomly, learning-based SMBO shows better efficiency in finding promising HPs to evaluate in iterations. In short, SMBO constructs a cheap regression model (called surrogate function s) and uses it for identifying the promising HPs to be evaluated by the (expensive) true objective function. It then iterates between fitting the surrogate function with newly evaluated HP information and gathering new information based on the surrogate function. We provide the pseudo-code of the *supervised* HPO by SMBO in [342] Appx. Algo. A1, and note that it does not directly apply to *HPO for OD* as performance cannot be evaluated without ground truth labels (line 4).

We thus enable (originally *supervised*) SMBO for *unsupervised* outlier detection HPO by plugging the PPE$f(\cdot)$ from the meta-train in place of HP performance evaluation. The key steps of online HPO are presented below.

Surrogate Function and Initialization. As an approximation of the expensive objection function, surrogate function s only takes HP settings as input, aiming for fast performance evaluation on a large collection of sampled HPs. For the new task $\mathbf{X}_{\text{test}}$ without access to true performance evaluation, HPOD lets s learn a mapping from an HP λ_k to its predicted performance, i.e., $s(\lambda_k) \mapsto f(\lambda_k, \mathbf{m}_{\text{test}}, \mathbf{I}_{\text{test},k})$. To enable $f(\cdot)$ on $\mathbf{X}_{\text{test}}$, HPOD needs one-time computation for the corresponding meta-features as $\mathbf{m}_{\text{test}} := \psi(\mathbf{X}_{\text{test}}) \in \mathbb{R}^d$. We want to remark that s differs from $f(\cdot)$ in two aspects. First, s can make fast performance predictions on HPs as it only needs HPs as input, while $f(\cdot)$ is more costly since IPMs require model building. Second, s is a regression model that can measure both *the predicted performance* of HP settings and *uncertainty (potential) around*

the prediction simultaneously. A popular choice for s is the Gaussian Process (GP) [310]*.

To initialize s, we train it on a small number of HPs. More specifically, we train s with pairs of HPs and their corresponding predicted performance by $f(\cdot)$ on $\mathbf{X}_{\text{test}}$, and also initialize the evaluation set λ_{eval} to these HPs. Although we can randomly sample the initial HPs, we propose to set them to top-performing HPs from similar meta-train tasks. Consequently, our initial s is more accurate in predicting likely well-performing HPs on $\mathbf{X}_{\text{test}}$. We defer the details of this meta-learning-based surrogate initialization to §4.3.4.1.

Iteration: Identifying Promising HPs. Although we can already output an HP from $\lambda_{\mathbf{eval}}$ with the highest predicted performance after initialization, we aim to use s to identify "better and better" HPs.

In each iteration, we use s to predict the performance (denoted as $u_k := s(\lambda_k)$) and the uncertainty around the prediction (denoted as σ_k) of sampled $\lambda_k \in \lambda_{\mathbf{sample}}$†, and then select the most *promising* one to be "evaluated" by $f(\cdot)$. Intuitively, we would like to evaluate the HPs with both high predicted performance u_k (i.e., exploitation) and high potential/prediction uncertainty σ_k (i.e., exploration), which is widely known as "exploitation-exploration trade-off" [265]. Too much exploitation (i.e., always evaluating the similar HPs) will fail to identify promising HPs, while too much exploration (i.e., only considering high uncertainty HPs) may lead to low-performance HPs. Also, note that the quality of identified HPs depends on the prediction accuracy of s, where we propose to transfer knowledge from MSF $\mathcal{T}$ by performance similarity (see technical details in §4.3.4.2.)

How can we effectively balance the trade-off between exploitation and exploration in HPOD? Adapting the idea of SMBO, we use the acquisition function $a(\cdot)$ to factor in the trade-off and pick

*We use GP in HPOD; one may use any regressor with prediction uncertainty estimation, e.g., random forests [44].

† $\lambda_{\mathbf{sample}}$ is a finite HP candidate set that is randomly sampled from the full (continuous) HP space Λ Since s can make fast predictions, $\lambda_{\mathbf{sample}}$'s size can be large, e.g., 10,000 as in [120].

a promising HP setting based on the outputs of the surrogate function. The acquisition function quantifies the "expected utility" of HPs by balancing their predicted performance and the uncertainty. Thus, we output the most promising HP to evaluate by maximizing $a(\cdot)$:

$$\lambda := \underset{\lambda_k \in \lambda_{\mathbf{sample}}}{\arg\max} \quad a(s(\lambda_k)) \tag{4.2}$$

One of the most prominent choices of $a(\cdot)$ is Expected Improvement (EI) [131], which is used in HPOD and can be replaced by other choices. EI has a closed-form expression under the Gaussian assumption, and the EI value of HP setting λ_k is shown below.

$$\begin{aligned} &EI(s(\lambda_k)) := \sigma_k \cdot [u_k \cdot \Phi(u_k) + \phi(u_k)], \quad \text{where} \\ &u_k = \begin{cases} \frac{u_k - \widehat{\mathbf{P}}^*_{\text{test}}}{\sigma_k} & \text{if } \sigma_k > 0 \text{ and} \\ 0 & \text{if } \sigma_k = 0 \end{cases} \end{aligned} \tag{4.3}$$

In the above, $\Phi(\cdot)$ and $\phi(\cdot)$ respectively denote the cumulative distribution and the probability density functions of a standard Normal distribution, u_k and σ_k are the predicted performance and the uncertainty around the prediction of λ_k by the surrogate function s, and $\widehat{\mathbf{P}}^*_{\text{test}}$ is the highest predicted performance by $f(\cdot)$ on λ_{eval} so far. We compare EI with other selection criteria in §4.4.4.1.

At the e-th iteration, we plug the surrogate $s^{(e)}(\cdot)$ into Eq. (4.2), which returns $\lambda^{(e)}$ to evaluate. Next, we train the OD model M with $\lambda^{(e)}$ to get its outlier scores $\mathcal{O}^{(e)}_{\text{test}}$ and IPMs $\boldsymbol{\mathcal{I}}^{(e)}_{\text{test}}$, and then predict its performance by $f(\cdot)$

$$\widehat{\mathbf{P}}^{(e)}_{\text{test}} := f(\lambda^{(e)}, \mathbf{m}_{\text{test}}, \boldsymbol{\mathcal{I}}^{(e)}_{\text{test}}) \tag{4.4}$$

Finally, we add $\lambda^{(e)}$ to the evaluation set $\lambda_{\text{eval}} := \lambda_{\text{eval}} \cup \lambda^{(e)}$, and update the surrogate function to $s^{(e+1)}$ with newly evaluated HP information $\langle \lambda^{(e)}, \widehat{\mathbf{P}}^{(e)}_{\text{test}} \rangle$.

To sum up, HPOD alternates between (*i*) identifying the next promising HP by the surrogate function s and (*ii*) updating s based on newly evaluated HP.

Continuous HP search. Recall that an outstanding property of HPOD compared to the baselines is its capability for continuous HP search (Table 4.1). $\lambda_{\mathbf{sample}}$ can be *any* subset of the full HP space

Λ and not restricted to the discrete $\lambda_{\mathbf{meta}}$.

Time Budget. HPOD is an *anytime* algorithm: at any time the user asks for a result, it can always output the HP with the highest predicted performance in the evaluation set λ_{eval} at the current iteration (Eq. (4.1)). HPOD uses E to denote the max number of iterations.

4.3.4 Details of (Online) HPO

4.3.4.1 Meta-learning-based Surrogate Initialization

Other than initializing on a small set of *randomly sampled* HPs, we design a meta-learning initialization strategy for the surrogate function (see §4.3.3.2), based on the similarity between the test and meta-train datasets. Intuitively, we hope the surrogate function s can make accurate predictions on the top-performing HPs of the test dataset, while the accuracy of the under-performing HP regions is less important. To this end, we use the meta-features (see §4.3.2.1) to calculate the similarity between the test dataset to each meta-train task $\mathcal{D}_i \in \boldsymbol{\mathcal{D}}_{\text{train}}$, and initialize s with the top performing HPs from *the most similar meta-train dataset*; these HPs may yield good performance on the test dataset. See the comparison to random initialization in §4.4.4.2.

4.3.4.2 Surrogate Transfer by Performance Similarity

As introduced in §4.2.2, meta-learning can be used to improve the surrogate function s in SMBO by transferring knowledge from meta-train datasets. In HPOD, we train n independent *Meta-Surrogate Functions* (see §4.3.2.2) for n meta-train datasets, $\boldsymbol{\mathcal{T}} = \{t_1, \ldots, t_n\}$; each with the same regression function as s (i.e., Gaussian Process). To this end, we identify the most similar meta-train dataset $\mathcal{D}_i$ in iteration e, and use the test surrogate s and $\mathcal{D}_i$'s meta-surrogate t_i together to predict the performance of HP λ_k, i.e.,

$$u_k := s^{(e)}(\lambda_k) + w_i^{(e)} \cdot t_i(\lambda_k) \tag{4.5}$$

where $w_i^{(e)}$ is the similarity between the test dataset and $\mathcal{D}_i$ measured in iteration e. While we could use meta-features to measure the dataset similarity, its value does not change in iterations and finds the same meta-train dataset to transfer (even for different OD algorithms).

Instead, we (re-)calculate the *performance similarity* every iteration based on the HPs in λ_{eval} between each meta-train task and the test dataset, and *dynamically* transfer the most similar meta-train dataset's meta-surrogate function. Specifically, HPOD computes a rank-based similarity by weighted Kendall tau [270]) between each meta-train dataset's ground truth performance and the test dataset's predicted performance by $f(\cdot)$ on λ_{eval} (which gets updated in every iteration). See the effect of surrogate transfer in §4.4.4.3.

4.4 Experiments

4.4.1 Experiment Setting

OD Algorithms and Testbeds. We show the results of HPO on (*a*) deep RAE in §4.4.2.1 and (*b*) LOF and (*c*) iForest in §4.4.2.2. Each OD algorithm is evaluated on a 39-dataset testbed ([342] Appx. §D.2, Table D1). Details of each algorithm's HP spaces and the meta-HP set is provided in [342] Appx. D.3. Experiments are conducted on an AMD 5900x@3.7GhZ, 64GB RAM workstation with an NVIDIA RTX A6000.

Baselines. Table 4.1 summarizes the baselines with categorization. We include (*i*) ***Simple methods***: **(1) Default** always employs the same default/popular HP setting (only if specified in the literature) **(2) Random** choice of HPs and **(3) MinLoss** outputs the HP with the lowest internal loss (only applicable to the algorithms with an objective/loss function) and (*ii*) ***Complex methods***: **(4) HyperEnsemble (HyperEns** or **HE)** that averages the results of randomly sampled HPs [309] **(5) Global Best (GB)** selects the best performing HP on meta-train database on average **(6) ISAC** [132] **(7)**

ARGOSMART (AS) [217] and **(8) MetaOD** [348]. Additionally, we include **(9)** HPOD_0, a variant of HPOD that directly uses $f(\cdot)$ to choose from randomly sampled HPs (see §4.3.3.1). Note that the unsupervised OD model selection baselines (5)-(8) are not for *HPO for OD*, i.e. they are infeasible with continuous HP spaces. We adapt them for HPO by selecting from the discrete meta-HP set in §4.3.1. See baseline details in [342] Appx. D.4.

Evaluation. We split the meta-train/test by leave-one-out cross-validation (LOOCV). Each time we use one dataset as the input dataset for HPO, and the remaining datasets as meta-train. We run five independent trials and report the average for the baselines with randomness. We use Average Precision (AP) as the performance measure, while it can be substituted with any other measure. As the raw performance like AP is not comparable across datasets with varying magnitude, we report the normalized AP rank of an HP, ranging from 1 (the best) to 0 (the worst)—thus higher the better. Also, we provide an additional metric called "top q%", denoting that an HP's performance has no statistical difference from the top q% HP from the meta HP-set, ranging from 0 (the best) to 1 (the worst)—thus lower the better. To compare two methods, we use the paired Wilcoxon signed rank test across all 39 datasets (significance level $p<0.05$). We give the full performance results in [342] Appx. D.5.

Hyperparameters. HPOD's HPs are chosen by LOOCV on the meta-train. We initialize the surrogate function with 10 HPs. Also, we report the results within a 30-minute budget for deep RAE and 10-minute budget for LOF and iForest.

4.4.2 Experiment Results

4.4.2.1 Results on (Deep) Robust Autoencoder

Fig. 4.1a (right) and 4.2a show that HPOD **outperforms all baselines w.r.t. both the best avg. normalized AP rank and the top q% value**. Furthermore, HPOD is also statistically better than all

baselines as shown in Table 4.2a, including strong meta-learning baseline MetaOD (p=0.0398). Its advantages can be credited to two reasons. First, meta-learning-based HPOD leverages prior knowledge on similar historical tasks to predict HP perf. on the new dataset, whereas simple baselines like Random and MinLoss cannot. Second, only HPOD and HPOD_o can select HPs from continuous spaces, while other meta-learning baselines are limited to finite discrete HPs as specified for the meta-train datasets which are too few to capture optimal HPs (especially for deep models with huge HP spaces).

HPO by the internal objective(s) is insufficient. Fig. 4.2a shows that selecting HP by minimal reconstruction loss (i.e., MinLoss) has the worst performance for RAE, even if it can work with continuous spaces. This suggests that internal loss does not necessarily correlate with external performance. On avg., HPOD has 37% higher normalized AP rank, showing the benefit of transferring supervision via meta-learning.

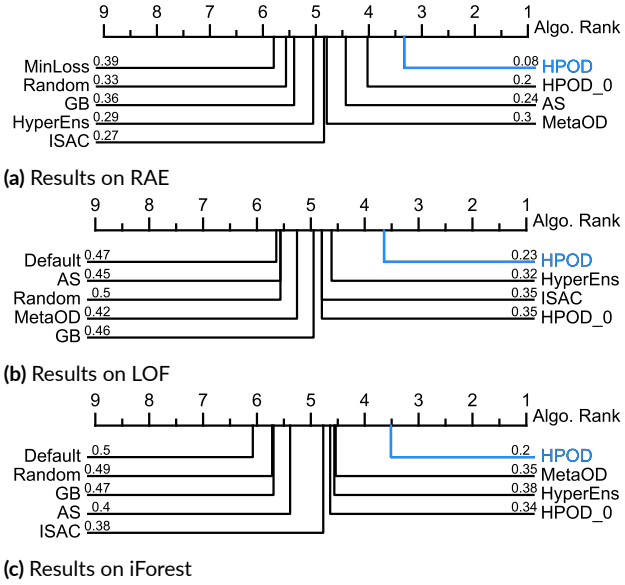


Figure 4.2: Comparison of avg. rank (lower is better) of algorithm performance across datasets on three algorithms. HPOD outperforms all w/ the lowest avg. algo. rank. The numbers on each line are the top q% value (lower is better) of the employed HP (or the avg.) by each method. HPOD shows the best performance in all three exp.

4.4.2.2 Results on LOF and iForest.

In addition to deep RAE, HPOD **shows generality on diverse OD algorithms**, including non-deep LOF (Fig. 4.1b and 4.2b) with mixed HP spaces (see details in Appx. Table D2) as well as ensemble-based iForest (Fig. 4.1c and 4.2c). HPOD achieves the best performance in both with the best norm. AP rank and top q%.

HPOD **is statistically better than the default HPs of LOF (p=0.0029) and iForest (p=0.0013)** (see Table 4.2b and 4.2c). More specifically, we find that HPOD provides +58% and +66% performance (i.e., normalized AP rank) improvement over using the default HPs of LOF (Fig. 4.1b (right)) and iForest (Fig. 4.1c). In fact, note that the default HPs rank the lowest for both LOF (Fig. 4.2b) and iForest (Fig. 4.2c), justifying the importance of HPO methods in unsupervised OD.

HyperEns that averages outlier scores from randomly samples HPs yield reasonable performance, which agrees with the observations in the literature [71]. However, HyperEns has a higher inference cost as it needs to save and use all base models, not ideal for time-critical applications. Using a single model with the selected HPs by HPOD offers not only better accuracy but also efficiency.

4.4.3 Case Study

We trace how HPOD identifies better HPs over iterations. Example of tuning LOF on `Cardiotocography` dataset is provided in Table 4.3. Among 200 candidate HP settings (see [342] Appx. Table D6), the optimal HP setting is {'Chebyshev', 79} with AP=0.3609. In 30 iterations, HPOD gradually identifies better HPs (closer to optimal), i.e., {'Chebyshev', 73}. Its AP improves from 0.2866 (1-*st* iteration) to 0.357 (30-*th* iteration). See the full trace in [342] Appx. D.6.

Ours	baseline	p-value	Ours	baseline	p-value
HPOD	AS	0.0309	**HPOD**	ISAC	0.0028
HPOD	Random	0.0014	**HPOD**	MetaOD	0.0398
HPOD	HyperEns	0.0382	**HPOD**	MinLoss	0.0003
HPOD	GB	0.0002	**HPOD**	HPOD_o	0.0201

(a) On RAE, HPOD is statistically better than all baselines.

Ours	baseline	p-value	Ours	baseline	p-value
HPOD	AS	0.0023	**HPOD**	ISAC	0.0246
HPOD	Random	0.0001	**HPOD**	MetaOD	0.0088
HPOD	**HyperEns**	0.0607	**HPOD**	Default	0.0029
HPOD	GB	0.0017	**HPOD**	HPOD_o	0.0016

(b) On LOF, HPOD is statistically better than all (except HyperEnsemble (HE)), including the *default* HP setting.

Ours	baseline	p-value	Ours	baseline	p-value
HPOD	AS	0.0055	**HPOD**	ISAC	0.0088
HPOD	Random	0.0003	**HPOD**	MetaOD	0.0289
HPOD	HyperEns	0.0484	**HPOD**	Default	0.0013
HPOD	GB	0.0027	**HPOD**	HPOD_o	0.003

(c) On iForest, HPOD is statistically better than all baselines, including the *default* HP setting.

Table 4.2: Pairwise statistical test results between HPOD and baselines by Wilcoxon signed rank test. Statistically better method shown in **bold** (both marked **bold** if no significance).

4.4.4 Ablation Studies and Other Analysis

4.4.4.1 The Choices of Acquisition Function

HPOD uses the EI acquisition to select an HP based on the surrogate function's prediction (see §4.3.3.2). We compare it with the random and greedy acquisition (latter picks the HP with the highest predicted performance, ignoring uncertainty) in Fig. 4.3, where EI-based acquisition performs best.

# Iter	Dist. metric	# Neighbors	AP Value
1	Manhattan	23	0.2866
...	...	...	...
6	Cosine	42	0.327
...	...	...	...
10	Cosine	55	0.3438
...	...	...	...
20	Chebyshev	72	0.3569
...	...	...	...
30	Chebyshev	73	0.357
Optimal	Chebyshev	79	**0.3609**

Table 4.3: Trace of HPOD on `Cardiotocography` dataset. Over iterations (col. 1), HPOD gradually identifies better HPs (col. 2 &3), with higher AP (col. 4). The optimal HP on from the meta-HP set is {'Chebyshev', 79}, which HPOD gets closer to the optimal HP during its adaptive search.

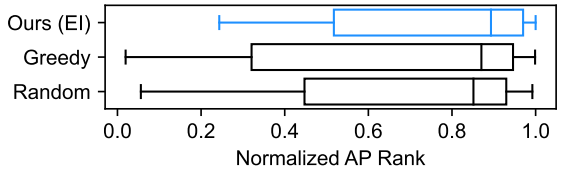


Figure 4.3: Ablation of EI (med.=0.893) vs. the greedy (med.=0.870) and random acquisition (med.=0.851).

4.4.4.2 Surrogate Initialization

HPOD uses meta-learning to initialize the surrogate (see §4.3.4.1). Fig. 4.4 shows its advantage over random initialization with higher performance.

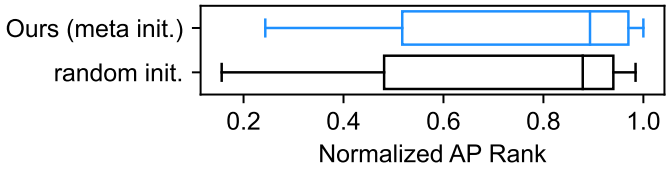


Figure 4.4: Ablation of meta- (med.=0.893) vs. random-initialization (med. =0.874) of the surrogate function.

4.4.4.3 The Effect of Surrogate Transfer

To improve the prediction performance of the surrogate function, HPOD transfers meta-surrogate functions from similar meta-train tasks (§4.3.4.2). Fig. 4.5 shows the transfer helps find better HPs, demonstrating the added value of meta-learning besides PPE training of $f(\cdot)$ and surrogate initialization.

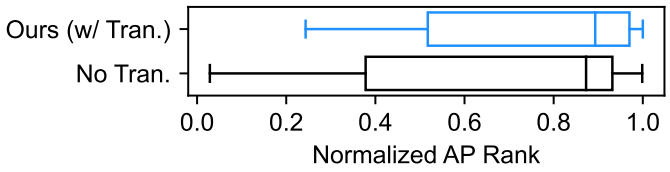


Figure 4.5: Ablation of ours w/ surrogate transfer (med.=0.893) vs. without transfer (med. =0.872).

4.5 Discussions

We introduce (to our knowledge) the *first systematic continuous hyperparameter optimization (HPO) approach for unsupervised outlier detection (OD)*. The proposed HPOD is a meta-learner, and builds on an extensive pool of *existing* OD benchmark datasets based on which it trains a performance predictor (offline). Given a new task without labels (online), it capitalizes on the performance predictor to enable (originally supervised) sequential model-based optimization for identifying promising HPs iteratively. Notably, HPOD stands out from all prior work on *HPO for OD* in being capable of handling both discrete *and* continuous HPs. Extensive experiments on three (including both deep and shallow) OD algorithms show its generality, where it significantly outperforms a diverse set of baselines. Future work will consider joint algorithm selection and continuous hyperparameter optimization for unsupervised outlier detection.

Part II

Systems

5

PyOD: A Comprehensive and Scalable Detection System on CPU

This chapter is primarily based on:

Yue Zhao*, Zain Nasrullah, and Zheng Li. "PyOD: A Python Toolbox for Scalable Outlier Detection." *Journal of Machine Learning Research (JMLR)*, 2019.

PyOD is an open-source Python toolbox for performing scalable outlier detection on multivariate data. Uniquely, it provides access to a wide range of outlier detection algorithms, including established outlier ensembles and more recent neural network-based approaches, under a single, well-documented API designed for use by both practitioners and researchers. With robustness and scalability in mind, best practices such as unit testing, continuous integration, code coverage, maintainability checks, interactive examples, and parallelization are emphasized as core components in the toolbox's development. PyOD is compatible with both Python 2 and 3 and can be installed through Python Package Index (PyPI) or https://github.com/yzhao062/pyod.

5.1 Introduction

Outlier detection, also known as anomaly detection, refers to the identification of rare items, events, or observations that differ from the general distribution of a population. Since the ground truth is often absent in such tasks, dedicated outlier detection algorithms are extremely valuable in fields that process large amounts of unlabelled data and require a means to reliably perform pattern recognition [7]. Industry applications include fraud detection in finance [11], fault diagnosis in mechanics [271], intrusion detection in network security [89] and pathology detection in medical imaging [31].

To help approach these problems, established outlier detection packages exist in various programming languages such as ELKI Data Mining [5] and RapidMiner [249] in Java and outliers [141] in R. However, Python, one of the most important languages in machine learning, still lacks a dedicated toolkit for outlier detection. Existing implementations either stand as single algorithm tools like PyNomaly [59] or exist as part of a general-purpose framework like scikit-learn [235] which does not cater specifically to anomaly detection. To fill this gap, we propose and implement PyOD—a comprehensive Python toolbox for scalable outlier detection.

Table 5.1: Select outlier detection models in PyOD. See https://github.com/yzhao062/pyod for the full list.

Method	Year	Category	JIT	Multi-core
LOF [46]	2000	Proximity	No	Yes
kNN [243]	2000	Proximity	No	Yes
AvgkNN [26]	2000	Proximity	No	Yes
CBLOF [112]	2003	Proximity	Yes	No
OCSVM [262]	2001	Linear Model	No	No
LOCI [231]	2003	Proximity	Yes	No
PCA [272]	2003	Linear Model	No	No
MCD [106]	2004	Linear Model	No	No
Feature Bagging [155]	2005	Ensembling	No	Yes
ABOD [145]	2008	Proximity	Yes	No
Isolation Forest [180]	2008	Ensembling	No	Yes
HBOS [92]	2012	Proximity	Yes	No
SOS [123]	2012	Proximity	Yes	No
AutoEncoder [256]		Neural Net	Yes	No
AOM [7]		Ensembling	No	No
MOA [7]		Ensembling	No	No
SO-GAAL [184]	2019	Neural Net	No	No
MO-GAAL [184]	2019	Neural Net	No	No
XGBOD [344]	2018	Ensembling	No	Yes
LSCP [346]	2019	Ensembling	No	No
COPOD [174]	2020	Probabilistic	No	Yes
ROD [19]	2020	Ensembling	No	No
ECOD [175]	2022	Probabilistic	No	Yes
LUNAR [95]	2022	Neural Net	No	No

Compared to existing libraries, PyOD has six distinct advantages. Firstly, it contains more than 40 algorithms that cover both classical techniques such as local outlier factor and recent neural network architectures such as autoencoders or adversarial models. Secondly, PyOD implements combination methods for merging the results of multiple detectors and outlier ensembles which are an emerging set of models. Thirdly, PyOD includes a unified API, detailed documentation and interactive examples across all algorithms for clarity and ease of use. Fourthly, all models are covered by unit testing with cross-platform continuous integration, code coverage and code maintainability checks. Fifthly,

optimization instruments are employed when possible: just-in-time (JIT) compilation and parallelization are enabled in select models for scalable outlier detection. Lastly, `PyOD` is compatible with both Python 2 and 3 across major operating systems (`Windows`, `Linux` and `MacOS`). Popular detection algorithms implemented in `PyOD` (version 1.0.7) are summarized in Table 5.1.

5.2 Project Focus

Build robustness. Continuous integration tools by GitHub Action are leveraged to conduct automated testing under various versions of Python and operating systems. Tests are executed daily, when commits are made to the development and master branches, or when a pull request is opened.

Quality assurance. The project follows the `PEP8` standard; maintainability is ensured by *CodeClimate*, an automated code review and quality assurance tool. Additionally, code blocks with high cognitive complexity are actively refactored and a standard set of unit tests exist to ensure 95% overall code coverage. These design choices enhance collaboration and this standard is enforced on all pull requests for consistency.

Community-based development. `PyOD`'s code repository is hosted on GitHub* to facilitate collaboration. At the time of this writing, eight contributors have helped develop the code base and others have contributed in the form of bug reports and feature requests.

Documentation and examples. Comprehensive documentation is developed using `sphinx` and `numpydoc` and rendered using *Read the Docs*†. It includes detailed API references, an installation guide, code examples and algorithm benchmarks. An interactive Jupyter notebook is also hosted on *Binder* allowing others to experiment prior to installation.

Project relevance. `PyOD` has been used in various academic and commercial projects. The GitHub repository receives more than 500,000 PyPI downloads per month.

*https://github.com/yzhao062/pyod
†https://pyod.readthedocs.io

5.3 Library Design and Implementation

PyOD is compatible with both Python 2 and 3 using six; it relies on numpy, scipy and scikit-learn as well. Neural networks such as autoencoders and SO_GAAL additionally require Keras. To enhance model scalability, select algorithms (Table 5.1) are optimized with JIT using numba. Parallelization for multi-core execution is also available for a set of algorithms using joblib. Inspired by scikit-learn's API design [49], all implemented outlier detection algorithms inherit from a base class with the same interface: (i) fit processes the train data and computes the necessary statistics; (ii) decision_function generates raw outlier scores for unseen data after the model is fitted; (iii) predict returns a binary class label corresponding to each input sample instead of the raw outlier score and (iv) predict_proba offers the result as a probability using either normalization or Unification [144]. Within this framework, new models are easy to implement by taking advantage of inheritance and polymorphism. Base methods can then be overridden as necessary.

Once a detector has been fitted on a training set, the corresponding train outlier scores and binary labels are accessible using its decision_scores_ and labels_ attributes. Once fitted, the model's predict, decision_function and predict_proba methods may be called for use on new data. An example showcasing the ease of use of this API is shown in Code Snippet 1 with an angle-based outlier detector (ABOD).

In addition to the outlier detection algorithms, a set of helper and utility functions (generate_data, evaluate_print and visualize) are included in the library for quick model exploration and evaluation. The two-dimensional artificial data used in the example is created by generate_data which generates inliers from a Gaussian distribution and outliers from a uniform distribution. An example of using visualize is shown in Figure 5.1.

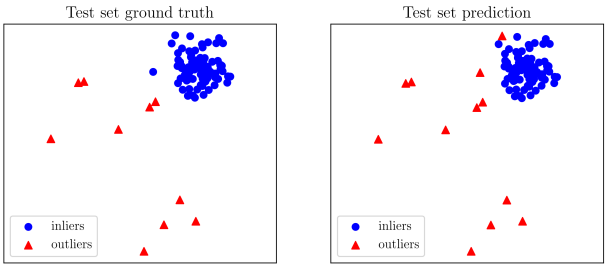


Figure 5.1: Demonstration of using PyOD in visualizing prediction result

```
>>> from pyod.models.abod import ABOD
>>> from pyod.utils.data import generate_data
>>> from pyod.utils.data import evaluate_print
>>> from pyod.utils.example import visualize
>>>
>>> X_train, y_train, X_test, y_test = generate_data(\
...   n_train=200, n_test=100, n_features=2)
>>> clf = ABOD(method="fast")               # initialize detector
>>> clf.fit(X_train)                        # fit the model with data
>>>
>>> y_test_pred = clf.predict(X_test)       # binary labels
>>> y_test_scores = clf.decision_function(X_test) # raw outlier scores
>>> y_test_proba = clf.predict_proba(X_test) # outlier probability
>>>
>>> evaluate_print("ABOD", y_test, y_test_scores) # performance evaluation
ABOD Performance; ROC: 0.934; Precision at n: 0.902
>>> visualize(y_test, y_test_scores)        # prediction visualization
```

Code Snippet 1: Demo of PyOD API with the ABOD detector

5.4 Discussions

This chapter presents PyOD, a comprehensive Python toolbox for scalable outlier detection. It includes more than 40 classical and emerging detection algorithms. As avenues for future work, we plan to enhance the toolbox by implementing models that work well with time series and geospatial data, improving computational efficiency through distributed computing, and addressing engineering challenges such as handling sparse matrices or memory limitations. Part of the above limitations is addressed in Chapter 6 (i.e., distributed learning) and 7 (i.e., memory efficient GPU distributed learning).

6

SUOD: Accelerating Heterogeneous Outlier Detection on CPU

This chapter is primarily based on:

Yue Zhao*, Xiyang Hu*, Cheng Cheng, Cong Wang, Changlin Wan, Wen Wang, Jianing Yang, Haoping Bai, Zheng Li, Cao Xiao, Yunlong Wang, Zhi Qiao, Jimeng Sun, and Leman Akoglu. "SUOD: Accelerating Large-scale Unsupervised Heterogeneous Outlier Detection." *Conference on Machine Learning and Systems (MLSys)*, 2021.

6.1 Highlight

Due to the lack of ground truth labels in OD, practitioners often have to build many unsupervised, heterogeneous models (i.e., different algorithms with varying hyperparameters) for further combination and analysis, rather than relying on a single model. We discuss this heterogeneous outlier ensemble problem in §0.4.

How to *accelerate* the training and scoring on new-coming samples by outlyingness (referred as prediction throughout this chapter) with *a large number of unsupervised, heterogeneous* OD models?

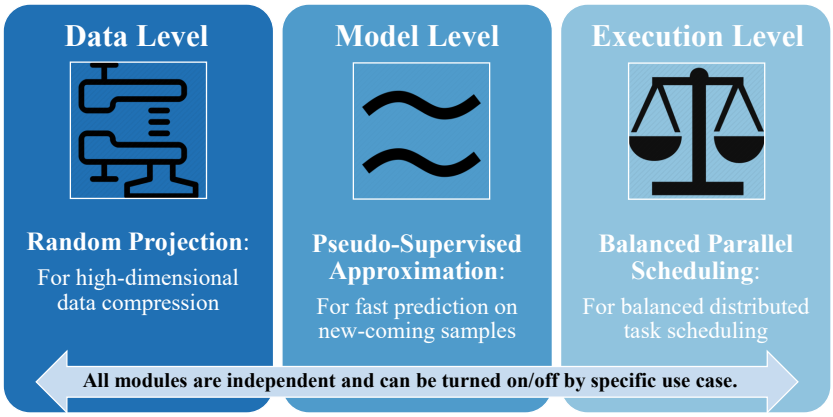


Figure 6.1: SUOD focuses on three independent levels.

As shown in Fig. 6.1, SUOD has three modules focusing on complementary levels: random projection (**data level**), pseudo-supervised approximation (**model level**), and balanced parallel scheduling (**execution level**). For high-dimensional data, SUOD generates a random low-dimensional subspace for each base model by Johnson-Lindenstrauss projection [130], in which the corresponding base model is trained. If prediction on new-coming samples is needed, fast supervised models are employed to approximate costly unsupervised outlier detectors (e.g., kNN and LOF). To train the supervised approximators, we regard the unsupervised models' outputs on the train set as "pseudo ground truth". Intuitively, this may be viewed as distilling knowledge from complex unsupervised

models [113] by fast and more interpretable supervised models. We also build a taskload predictor to reduce the scheduling imbalance in a distributed environment. Other than generically assigning an equal number of models to each worker, our balanced parallel scheduling mechanism forecasts OD model cost, e.g., training time, before scheduling them, so that the taskload is evenly distributed among workers. Notably, all three acceleration modules are designed to be independent but complementary, which can be used alone or combined as a system. It is noted that offline training and prediction is one of the primary scenarios in machine learning applications [22, 209].

Our contributions are summarized as follows:

1. **The First Comprehensive OD Acceleration System**: We propose SUOD, (to our knowledge) the most comprehensive system for heterogeneous OD acceleration by a holistic view of data, model, and execution level.

2. **Analysis of Data Compression**: We examine various data compression methods and identify the best performing method(s) for large-scale outlier ensembles.

3. **Exploration of Model Approximation for OD**: We analyze the use of pseudo-supervised regression models' performance in approximating costly unsupervised OD models, as the first research effort on this topic.

4. **Forecasting-based Scheduling System**: We fix an imbalance scheduling issue in distributed heterogeneous OD efficiency, saving up to 61% execution time.

5. **Effectiveness**: We conduct extensive experiments to show the effectiveness of the acceleration modules independently, and of the entire framework as a whole, along with a real-world case on fraud detection.

6.2 Related Work

6.2.1 Outlier Detection and Ensemble Learning

Outlier detection has numerous important applications, such as rare disease detection [171], healthcare utilization analysis [115], video surveillance [189], and network intrusion detection [155]. Yet, detecting outliers is challenging due to various reasons [7, 346, 347]. Most of the existing detection algorithms are unsupervised as ground truth is often absent in practice, and acquiring labels can be prohibitively expensive. We include 8 popular OD algorithms in this study for experimentation, including Isolation Forest [180], Local Outlier Factor (LOF) [46], Angle-based Outlier Detection (ABOD) [145], Feature Bagging [156], Histogram-based Outlier Score (HBOS) [92], and Clustering-Based Local Outlier Factor (CBLOF) [112]. See §6.4 for details on OD models.

Consequently, relying on a single unsupervised model has inherently high risk, and outlier ensembles that leverage a group of diversified (e.g., heterogeneous) detectors have become increasingly popular [7, 359, 9]. There are a group of unsupervised outlier ensemble frameworks proposed in the last several years from a simple combination like averaging [8] to more complex model selection approaches like SELECT [246], LSCP [346], and MetaOD [348]. In addition to fully unsupervised outlier ensembles, there are (semi-)supervised ensembling frameworks that can incorporate existing label information such as XGBOD [344]. For both unsupervised and (semi-)supervised methods, a large group of unsupervised, heterogeneous OD models is used as the base for robustness and performance—SUOD is hereby proposed to accelerate this scenario.

6.2.2 Scalability and Efficiency Challenges in OD

Efforts have been made through various channels to accelerate large-scale OD. On the **data level**, researchers try to project high-dimensional data onto lower-dimensional subspaces [4], including sim-

ple Principal Component Analysis (PCA) [272] and more complex subspace method HiCS [136]. However, deterministic projection methods, e.g., PCA, are not ideal for building diversified heterogeneous OD—they lead to the same or similar subspaces with limited diversity by nature, resulting in the loss of outliers [7]. Complex projection and subspace methods may bring performance improvement for outlier mining, but the generalization capacity is limited. Hence, projection methods preserving pairwise distance relationships for downstream tasks should be considered. SUOD's considers both diversity induction and pairwise distance preservation for downstream tasks, leading to diversified and meaningful feature spaces (see §6.3.3).

On the **model level**, knowledge distillation emerges as a good way of compressing large neural networks [113], while its usage in outlier detection is still underexplored. Knowledge distillation refers to the notion of compressing a large, often cumbersome model(s) into a small and more interpretable one(s). Under the context of OD, proximity-based models, such as LOF, can be slow (high time complexity) in predicting new-coming samples with limited interpretability, which severely restricts their usability. SUOD adapts a similar idea to outlier mining by "distilling" complex unsupervised models. Although SUOD shares a similar concept as knowledge distillation for computational cost optimization, there are a few notable differences (see §6.3.4).

There are also engineering cures on the **execution level**. For various reasons, OD has no mature and efficient distributed frameworks with thousands of clusters—distributed computing for OD mainly falls into the category of "scale-up" which focuses on leveraging multiple local cores on a single machine more efficiently. To this end, specific OD algorithms can be accelerated by distributed computing with multiple workers (e.g., CPU cores) [218, 188]. However, these frameworks are not designed for a group of heterogeneous models but only a single algorithm, which limits their usability. It is noted that a group of heterogeneous detection models can have significantly varied computational costs. As a simple example, let us split 100 heterogeneous models into 4 groups for parallel training. If group #2 takes significantly longer time than the others to finish, it behaves like the bot-

tleneck of the system. More formally, imbalanced task scheduling causes the system efficiency to be curbed by the worker taking the most time. There is also a line of system research focus on more efficient task scheduling for "shorter tasks on larger degree of parallelism", e.g., Sparrow [221] and Pigeon [306]. For instance, Sparrow discusses scheduling millions of tasks (at a millisecond scale) on thousands of machines. Differently, Heterogeneous OD applications typically use a few OD models (e.g., 5 to 1,000) and each of them takes a few seconds to hours to run (a considerable number of tasks; each takes time to run), which is "longer tasks with a small number of workers". SUOD is, therefore, proposed to reduce the inefficiency in distributed heterogeneous OD specifically.

6.3 System Design

6.3.1 Problem Formulation

OD applications and research are primarily running a single, on-prime, powerful machine with multiple cores/workers, due to its high-stake nature (e.g., data sensitive of financial transactions). Therefore, we formulate unsupervised heterogeneous OD training and prediction tasks with:

- m unsupervised heterogeneous OD models, where $\mathcal{M} = \{M_1, ..., M_m\}$. We refer to the combination of an algorithm and its hyperparameters as a model.
- train data $\mathbf{X}_{\text{train}} \in \mathbb{R}^{n \times d}$ without ground truth labels.
- (optional) test data $\mathbf{X}_{\text{test}} \in \mathbb{R}^{l \times d}$ for prediction. The OD models should be trained first.
- (optional) t available workers for distributed computing, e.g., t cores on a single machine. This constructs the worker pool as $\mathcal{W} = \{W_1, ..., W_t\}$. By default, a single worker setting ($t = 1$) is assumed.

Without SUOD, one will train each model in $\mathcal{M}$ on $\mathbf{X}_{\text{train}}$ iteratively, e.g., with a for loop. If there are multiple workers available ($t > 1$), a generic scheduling system will equally split m models

into t groups, so each available worker will process roughly $\lceil \frac{m}{t} \rceil$ models. Prediction on new-coming samples $\mathbf{X}_{\text{test}}$ follows a similar manner as training. See Algorithm 4 for a detailed symbol definition.

Algorithm 4 SUOD: Training and Prediction

Input: m unsupervised OD models $\mathcal{M}$; train data $\mathbf{X}_{\text{train}} \in \mathbb{R}^{n \times d}$; target dimension k; the number of available workers t (optional, default to 1); test data $\mathbf{X}_{\text{test}} \in \mathbb{R}^{l \times d}$ (optional); supervised regressor R (optional)

Output: Trained OD models $\mathcal{M}$; fitted pseudo-supervised regressors R (optional); test prediction results $\hat{y}_{\text{test}}$ (optional)

for each model M_i in $\mathcal{M}$ **do**
  if random projection is enabled (§6.3.3) **then**
    Initialize a JL transformation matrix $\mathbf{W} \in \mathbb{R}^{d \times k}$
    Get feature subspace $\psi_i := \langle \mathbf{X}_{\text{train}}, \mathbf{W} \rangle \in \mathbb{R}^{n \times k}$
  else
    Use the original space $\psi_i := \mathbf{X}_{\text{train}} \in \mathbb{R}^{n \times d}$
  end if
end for
if number of available workers $t > 1$ **then**
  Scheduling the training of m models to t workers by minimizing Eq. 6.2 (see §6.3.5). Models are trained on the corresponding feature space $[\psi_1, ..., \psi_m]$.
else
  Train each model M_i in $\mathcal{M}$ on its corresponding ψ_i.
end if
Return trained models $\mathcal{M}$

if Scoring on newcoming samples $\mathbf{X}_{\text{test}}$ **then**
  Acquire the pseudo ground truth $target^{\psi_i}$ as the output of M_i on ψ_i, i.e., $target^{\psi_i} := M_i(\psi_i)$
  for each costly model M_i in $\mathcal{M}_c$ **do**
    Initialize a supervised regressor R_i
    Train R_i by $\{\psi_i, target^{\psi_i}\}$ (see §6.3.4)
    Predict by supervised R_i, $\hat{y}^i_{\text{test}} = R_i.\text{predict}(\mathbf{X}_{\text{test}})$
  end for
  Return $\hat{y}_{\text{test}}$ and approximation regressors R
end if

6.3.2 System Design

SUOD is designed to accelerate the above procedures with three independent modules targeting different levels (data, model, and execution). **Each module can be flexibly enabled or disabled** as shown in Algorithm 4. For high-dimensional data, SUOD can randomly project the original feature onto low-dimensional spaces (§6.3.3). Pairwise distance relationships are expected to be maintained, and diversity is induced for ensemble construction. A fast supervised regressor could be used to approximate the output of each slow and costly unsupervised detector. We could use the supervised regressor for fast prediction (§6.3.4). If there are multiple available workers for distributed computing, we propose a forecasting-based scheduling mechanism (§6.3.5) to reduce taskload imbalance in heterogeneous OD.

6.3.3 Data Level: Random Projection (RP) for Data Compression

For high-dimensional datasets, many proximity-based OD algorithms suffer from the curse of dimensionality [156]. A widely used dimensionality reduction to cure this is the Johnson-Lindenstrauss (JL) projection [130], which has been applied to OD because of its great scalability [261]. Unlike PCA discussed in §6.2.2, JL projection could compress the data without heavy distortion on the Euclidean space—*outlyingness information is therefore preserved in the compression*. Moreover, its built-in randomness can be useful for diversity induction in heterogeneous OD—data randomness can also serve as a source of heterogeneity. Additionally, JL projection ($\mathcal{O}(ndk)$) is more efficient than popular PCA ($\mathcal{O}(nd^2 + n^3)$) with lower time complexity, where k is the target dimension for compression.

JL projection is defined as given a set of n samples $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ...\mathbf{x}_n\}$, each $\mathbf{x}_i \in \mathbb{R}^d$, let $\mathbf{W}$ be a $k \times d$ projection matrix with each entry drawing independently from a predefined distribution F, e.g., Gaussian, so that $\mathbf{W} \sim F$. Then the JL projection is a function $f : \mathbb{R}^d \rightarrow \mathbb{R}^k$ such that

$f(\mathbf{x}_i) = \frac{1}{\sqrt{k}}\mathbf{x}_i\mathbf{W}^T$. JL projection randomly projects high-dimensional data (d dimensions) to lower-dimensional subspaces (k dimensions), but preserves the distance relationship between points. In fact, if we fix some $\mathbf{v} \in \mathbb{R}^d$, for every $\varepsilon \in (0, 3)$, we have [261]:

$$P\left[(1-\varepsilon)\|\mathbf{v}\|^2 \leq \|\frac{1}{\sqrt{k}}\mathbf{v}\mathbf{W}^T\|^2 \leq (1+\varepsilon)\|\mathbf{v}\|^2\right] \leq 2e^{-\varepsilon^2\frac{k}{6}} \tag{6.1}$$

Let $\mathbf{v}$ to be the differences between vectors. Then, the above bound shows that for a finite set of n vectors $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ...\mathbf{x}_n\} \in \mathbb{R}^d$, the pairwise Euclidean distance is preserved within a factor of $(1 \pm \varepsilon)$, reducing the vectors to $k = \mathcal{O}(\frac{log(n)}{\varepsilon^2})$ dimensions.

Four distributions F for JL projection are considered in this study: (i) *basic*: the transformation matrix is generated by standard Gaussian; (ii) *discrete*: the transformation matrix is picked randomly from Rademacher distribution (uniform in $\{-1, 1\}$); (iii) *circulant*: the transformation matrix is obtained by rotating the subsequent rows from the first row which is generated from standard Gaussian and (iv) *toeplitz*: the first row and column of the transformation matrix is generated from standard Gaussian, and each diagonal uses a constant value from the first row and column. A more thorough empirical study on JL methods can be found in [297].

For $\mathbf{X}_{\text{train}}$, RP can reduce the original feature space d to the target dimension k. Specifically, SUOD initializes a JL transformation matrix $\mathbf{W} \in \mathbb{R}^{d\times k}$ by drawing from one of the four distributions F. $\mathbf{X}_{\text{train}}$ is therefore projected onto the k dimension feature space as $\mathbf{X}'_{\text{train}} = \langle \mathbf{X}_{\text{train}}, \mathbf{W} \rangle \in \mathbb{R}^{n\times k}$. The transformation matrix $\mathbf{W}$ should be kept for transforming newcoming samples: $\mathbf{X}'_{\text{test}} = \langle \mathbf{X}_{\text{test}}, \mathbf{W} \rangle \in \mathbb{R}^{m\times k}$. It is noted that RP module should be used with caution. First, projection may not be helpful or even detrimental for subspace methods like Isolation Forest and HBOS. Second, if the number of samples n is too small, the bound in Eq. (6.1) does not hold.

6.3.4 Model Level: Pseudo-Supervised Approximation (PSA) for Fast Prediction

PSA module is designed to speed up prediction on new-coming samples. Specifically, after the models in $\mathcal{M}$ are trained, SUOD uses PSA to approximate and replace each **costly unsupervised model** by a **faster supervised regressor** for **fast offline prediction**. Notably, only costly unsupervised models should be replaced; the cost can be measured through time complexity analysis. For instance, proximity-based algorithms like kNN and LOF are costly in prediction (upper bounded by $\mathcal{O}(nd)$), and can be effectively replaced by "cheaper" supervised models like random forest [44] (upper bounded by $\mathcal{O}(ph)$ where p denotes the number of base trees and h denotes the max depth of a tree; often $p \ll n$ and $h \leq d$). This "pseudo-supervised" model uses the output of unsupervised models (outlyingness score) as "the pseudo ground truth"—*the goal is to approximate the output of the underlying unsupervised model*. It is noted that the approximator's prediction cost (i.e., time complexity) must be lower than the underlying unsupervised model, while maintaining a comparable level of prediction accuracy. For instance, fast (low time complexity) OD algorithms like Isolation Forest and HBOS should not be approximated and replaced. To facilitate this process, we predefine the pool of costly OD algorithm $\mathcal{M}_c$. If a model M_i is in $\mathcal{M}_c$, it will be approximated by default.

As shown in Algorithm 4, for each costly trained unsupervised model M_i belonging to $\mathcal{M}_c$, a supervised regressor R_i is trained by $\{\mathbf{X}_{\text{train}}, \mathbf{y}_i\}$; $\mathbf{y}_i$ is the outlyingness score by M_i on the train set (referred as pseudo ground truth)*. R_i is then used to predict on unseen data $\mathbf{X}_{\text{test}}$.

Remark 1: Supervised tree ensembles are recommended for PSA due to their outstanding scalability, robustness to overfitting, and interpretability (e.g., feature importance). In addition to the execution time reduction, supervised models are generally more interpretable. For instance, random

*If RP is enabled, $\mathbf{X}_{\text{train}}$ is replaced by the compressed space.

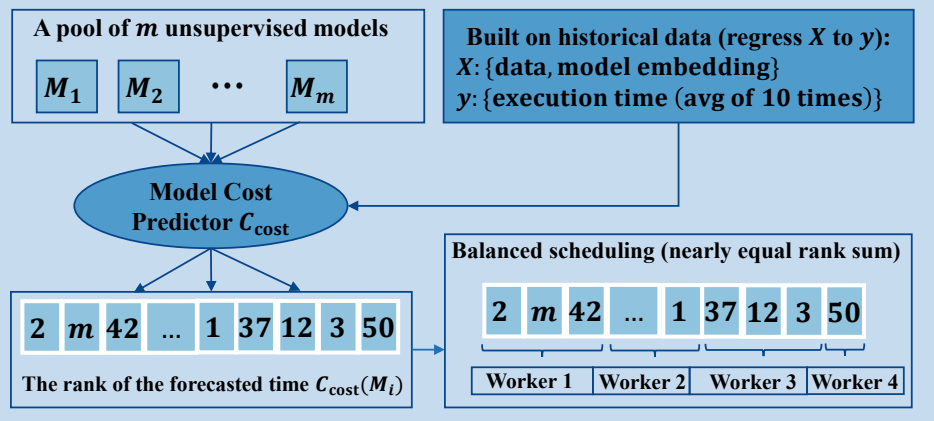


Figure 6.2: Flowchart of balanced parallel scheduling, which aims to assign nearly equal rank sum by model cost predictor C_{cost}.

forests used in the experiments can yield feature importance automatically to facilitate understanding.

Remark 2: Notably, PSA may be viewed as using supervised regressors to distill knowledge from unsupervised OD models. However, it works in a fully unsupervised manner, unlike the classic distillation under supervised settings.

6.3.5 Execution level: Balanced Parallel Scheduling (BPS) for Taskload Imbalance Reduction

Taskload Imbalance within Distributed Systems: It is likely to observe system inefficiency due to taskload imbalance among workers when a large number of heterogeneous models are used. For instance, we want to train 25 OD models with varying parameters from each of the four algorithms {kNN, Isolation Forest, HBOS, OCSVM} (100 heterogeneous models in total). The existing distributed frameworks, e.g., the voting machine in scikit-learn [235] or general frameworks like joblib[†], may simply split the models into 4 subgroups by order and schedule the first 25 models (all kNNs) on worker 1, the next 25 models on worker 2, etc. This does not account for the fact that

[†] https://github.com/joblib/joblib

within a group of heterogeneous models, the computational cost varies. Scheduling the task with the equal number of models can result in highly imbalanced load. In the worst-case scenario, one worker may be assigned significantly more load than the rest, resulting in halt to the entire process. In this example, the *k*NN subgroup will be the system curb due to high time complexity. One naive solution is to shuffle the base models randomly. However, there is no guarantee this heuristic could work, and it may be practically infeasible.

The proposed BPS focuses on delivering a more balanced task scheduling among workers via forecasting their cost in advance. Ideally, all workers can finish the scheduled tasks within a similar duration and return the results. To achieve this goal, SUOD comes with a *model cost predictor* C_{cost} to forecast the model execution time (sum of 10 trials) given the meta-features (descriptive features) of a dataset [348]. C_{cost} is trained on 11 algorithm families with 47 benchmark datasets by 10-fold cross-validation, yielding an effective regressor (random forest is used in this study). C_{cost}'s outputs show high Spearman's Rank correlation [280] ($r_s > 0.9$) to the true model cost rank with low p-value ($p < 0.0001$), in all folds. Given a dataset and a model, C_{cost} can predict the execution time of the model's execution time with high accuracy. It is noted that the model cost predictor is only trained for the major methods in Python Outlier Detection Toolbox (PyOD) [347]. For unseen models, they are classified as "unknown" to be assigned with the max cost to prevent over-optimistic scheduling.

As a result, a scheduling strategy is proposed by enforcing a nearly equal rank sum by the forecasted execution time among all available workers. Fig. 6.2 provides a simple example of scheduling m models to 4 workers. More formally, before scheduling m models for training (or prediction), cost predictor C_{cost} is invoked to forecast the execution time of each model M in $\mathcal{M}$ as $C_{\text{cost}}(M)$ and output the model cost rank of each model in $\{1, m\}$ (the higher the rank, the longer the forecasted execution time). If there are t cores (workers), each worker will be assigned a group of models to achieve the objective of minimizing the taskload imbalance among workers (Eq. 6.2).

$$\min_{\mathcal{W}} \sum_{i=1}^{t} \left| \sum_{M_j \in \mathcal{W}_i} C_{\text{cost}}(M_j) - \frac{m^2+m}{2t} \right| \tag{6.2}$$

Consequently, each worker is assigned a group of models with a rank sum close to the average rank sum $\frac{(1+m)m}{2}/t = \frac{m^2+m}{2t}$. Indeed, the accurate running time prediction is less relevant as it depends on the hardware—the rank is more useful as a relevance measure with the transferability to other hardware. That is, the running time will vary on different machines, but the relative rank should preserve. One issue around the sum of ranks is the overestimation of high-rank models. For instance, rank f-th model will be counted f times more heavily than rank 1 model during the sum calculation, even though their actual running time difference will not be as big as f times. To fix this, we introduce a discounted rank by rescaling model rank f to $1 + \frac{\alpha f}{m}$, where α denotes the scaling strength (default to 1). A larger α, therefore, puts more emphasis on costly models.

6.4 Experiments

First, three experiments are conducted to understand the effectiveness of individual modules independently: **Q1**: how will different compression methods affect the performance of downstream OD accuracy (§6.4.1); **Q2**: will use pseudo-supervised regressors lead to more efficient prediction in comparison to the original unsupervised models (§6.4.2) and **Q3**: how does the proposed balanced scheduling strategy perform under different settings (varying number of models m, number of workers t, etc.) (§6.4.3). Then, the full SUOD with all three modules enabled is evaluated regarding time cost and prediction accuracy (on new samples) (§6.4.4). Finally, a real-world deployment case on fraudulent claim analysis at IQVIA (a leading healthcare organization), is described (§6.4.5).

Datasets. Table 6.1 describes the selected outlier detection benchmark datasets, and more than 20 outlier detection benchmark datasets are used in this study[‡,§]. The data size n varies from 452

[‡]ODDS Library: http://odds.cs.stonybrook.edu
[§]DAMI Datasets: http://www.dbs.ifi.lmu.de/research/outlier-evaluation/DAMI

Table 6.1: Selected real-world benchmark datasets

Dataset	Pts (n)	Dim (d)	Outliers	% Outlier
Annthyroid	7200	6	534	7.41
Arrhythmia	452	274	66	14.60
Breastw	683	9	239	34.99
Cardio	1831	21	176	9.61
HTTP	567479	3	2211	0.40
Letter	1600	32	100	6.25
MNIST	7603	100	700	9.21
Musk	3062	166	97	3.17
PageBlock	5393	10	510	9.46
Pendigits	6870	16	156	2.27
Pima	768	8	268	34.90
Satellite	6435	36	2036	31.64
Satimage-2	5803	36	71	1.22
seismic	2584	10	170	6.59
Shuttle	49097	9	3511	7.15
SpameSpace	4207	57	1679	39.91
speech	3686	400	61	1.65
Thyroid	3772	6	93	2.47
Vertebral	240	6	30	12.50
Vowels	1456	12	50	3.43
Waveform	3443	21	100	2.90
Wilt	4819	5	257	5.33

(**Arrhythmia**) to 567,479 (**HTTP**) samples and the dimension d ranges from 3 to 274. For both random projection and parallel scheduling experiments, the full datasets are used for model building (training). For the pseudo-supervised approximation experiments and the full framework assessment, 60% of the data is used for training and the remaining 40% is set aside for validation.

OD Models. As shown in Table 6.2, we use a large group of outlier detection models in the experiment by varying algorithms and their corresponding hyperparameters. We build the model pool $\mathcal{M}$ by sampling from it.

Table 6.2: Outlier Detection Models in SUOD; see parameter definitions from PyOD [347]

Method	Parameter 1	Parameter 2
ABOD [145]	n_neighbors: [3, 5, 10, 15, 20, 25, 50, 60, 70, 80, 90, 100]	N/A
CBLOF [112]	n_clusters: [3, 5, 10, 15, 20]	N/A
Feature Bagging [156]	n_estimators: [10, 20, 30, 40, 50, 75, 100, 150, 200]	N/A
HBOS [92]	n_histograms: [5, 10, 20, 30, 40, 50, 75, 100]	tolerance: [0.1, 0.2, 0.3, 0.4, 0.5]
iForest [180]	n_estimators: [10, 20, 30, 40, 50, 75, 100, 150, 200]	max_features: [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
kNN [243]	n_neighbors: [1, 5, 10, 15, 20, 25, 50, 60, 70, 80, 90, 100]	method: ['largest', 'mean', 'median']
LOF [46]	n_neighbors: [1, 5, 10, 15, 20, 25, 50, 60, 70, 80, 90, 100]	method: ['manhattan', 'euclidean', 'minkowski']
OCSVM [262]	nu (train error tol): [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]	kernel: ['linear', 'poly', 'rbf', 'sigmoid']

Evaluation. For all experiments, performance is evaluated by taking the average of 10 independent trials using area under the receiver operating characteristic (ROC) curve and precision at rank n (P@N)—here n denotes the actual number of outliers. Both metrics are widely used in outlier research [359, 184].

6.4.1 Q1: Comparison of Model Compression Methods

In this section, **we demonstrate the effectiveness of RP module in high-dimensional OD tasks.** To evaluate the effect of data projection, we choose three costly outlier detection algorithms, ABOD, LOF, and kNN to measure their execution time, and prediction accuracy (ROC and P@N), before and after projection. These methods directly or indirectly measure sample similarity in Euclidean space, e.g., pairwise distance, which is prone to the curse of dimensionality, where data compression can help.

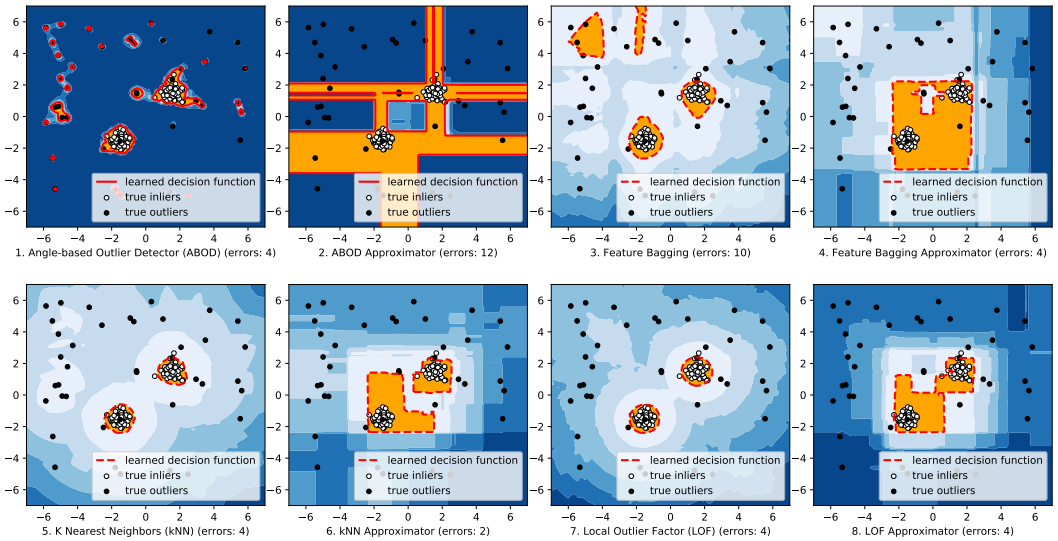


Figure 6.3: Decision surface comparison among unsupervised models and their pseudo-supervised approximators (in pairs). The approximator's decision boundary shows a tentative regularization effect, leading to even fewer detection errors.

Table 6.7 shows the comparison results on four datasets; the reduced dimension is set as $k = \frac{2}{3}d$ (33% compression). We compare the proposed four JL projection methods (see §6.3.3 for details of *basic*, *discrete*, *circulant*, and *toeplitz*) with **original** (no projection), **PCA**, and **RS** (randomly select k features from the original d features, used in Feature Bagging [156] and LSCP [346]). First, all compression methods show superiority regarding time cost. Second, **original** (no compression) method rarely outperforms, possibly due to the curse of dimensionality and lack of diversity [346]. Third, **PCA** is inferior to **original** regarding prediction accuracy (see LOF performance in Table 6.7e, 6.7h, and 6.7k). The observation supports our claim that PCA is not suited in this scenario (see §6.2.2). Fourth, JL methods generally lead to equivalent or better prediction performance than **original** regarding both time and prediction accuracy. Lastly, among all JL methods, *circulant* and *toeplitz* generally outperform others. For instance, *toeplitz* is chosen as the default choice since it brings more than 60% time reduction for kNN.

6.4.2 Q2: The Visual and Quantitative Analysis of PSA

Through both visualization and quantitative analysis, we observe PSA is useful for accelerating prediction of proximity-based OD algorithms. To better understand the effect of pseudo-supervised approximation, we first generate a synthetic dataset with 200 two-dimensional samples, consisting of 40 outliers generated by Normal distribution and 160 normal samples generated from Uniform distribution. In Fig. 6.3, we plot the decision surfaces of four costly unsupervised models (ABOD, Feature Bagging, *k*NN, and LOF) and of their corresponding supervised approximators (random forest regressor), with accuracy errors reported. In general, the faster pseudo supervised approximators do not lead to more errors, justifying the effectiveness of approximation. Fig. 6.3 subfigures 4 and 6 show that the approximators have even lower errors than the original (Feature Bagging and *k*NN). With a closer look at the decision surfaces, we assume that the approximation process improves the generalization ability of the model by "ignoring" some overfitted points. However, the approximation does not work with ABOD, possibly due to its extremely coarse decision surface (see Fig. 6.3, subfigure 1).

Table 6.3 and 6.4 compare prediction performance (scoring on new-coming samples) between the original unsupervised models and pseudo-supervised approximators on 10 datasets with 6 costly algorithms, regarding ROC and P@N. Since these algorithms are more computationally expensive than random forest regressors for prediction (by time complexity analysis), we skip the prediction time comparison where the gain is clear. Consequently, the focus is on whether the approximators could predict unseen samples as well as the original unsupervised models. The tables reveal that not all the algorithms are suited for PSA, which is in line with the visual analysis. For instance, ABOD shows a performance decrease in half of the datasets. Notably, ABOD looks for a low-dimensional subspace to embed the normal samples [7], leading to a complex decision surface to approximate. In contrast, proximity-based models benefit from the approximation. Both tables show, *k*NN, LoF,

and akNN (average kNN) experience a performance gain. Specifically, all three algorithms yield around 100% ROC increase on **HTTP**. Other algorithms, such as Feature Bagging and CBLOF, show a minor performance variation within the acceptable range. In other words, it is useful to perform PSA for these estimators as the time efficiency is greatly improved with little to no loss in prediction accuracy.

Table 6.3: Prediction ROC scores of unsupervised models (Orig) and their pseudo-supervised approximators (Appr) by the average of 10 independent trials. The better method within each pair is indicated in **bold**. The approximators (Appr) outperform in most cases.

Dataset	Annthyroid		Breastw		Cardio		HTTP		MNIST		Pendigits		Pima		Satellite		Satimage-2		Thyroid	
Model	Orig	Appr	Orig	Appr	Orig	Appr	Orig	Appr	Orig	Appr	Orig	Appr	Orig	Appr	Orig	Appr	Orig	Appr	Orig	Appr
ABOD	**0.83**	0.71	0.92	**0.93**	**0.63**	0.53	**0.15**	0.13	**0.81**	0.79	0.67	**0.82**	0.66	**0.70**	0.59	**0.68**	0.89	**0.99**	**0.96**	0.67
CBLOF	0.67	**0.68**	0.96	**0.98**	0.73	**0.76**	**1.00**	**1.00**	0.85	**0.89**	**0.93**	**0.93**	0.63	**0.68**	0.72	**0.77**	**1.00**	**1.00**	0.92	**0.97**
FB	**0.81**	0.45	**0.34**	0.10	0.61	**0.70**	0.34	**0.97**	0.72	**0.83**	0.39	**0.51**	0.59	**0.63**	0.53	**0.64**	0.36	**0.40**	**0.83**	0.46
kNN	**0.80**	0.79	**0.97**	**0.97**	0.73	**0.75**	0.19	**0.85**	0.85	**0.86**	0.74	**0.87**	0.69	**0.71**	0.68	**0.75**	0.96	**0.99**	0.97	**0.98**
akNN	0.81	**0.82**	**0.97**	**0.97**	0.67	**0.72**	0.19	**0.88**	0.84	**0.85**	0.72	**0.87**	0.69	**0.71**	0.66	**0.74**	0.95	**0.99**	0.97	**0.98**
LOF	0.74	**0.85**	0.44	**0.45**	0.60	**0.68**	0.35	**0.75**	0.72	**0.76**	0.38	**0.47**	0.59	**0.65**	0.53	**0.66**	0.36	**0.38**	0.80	**0.95**

Table 6.4: Prediction P@N scores of unsupervised models (Orig) and their pseudo-supervised approximators (Appr) by the average of 10 independent trials. The better method within each pair is indicated in **bold**. The approximators (Appr) outperform in most cases.

Dataset	Annthyroid		Breastw		Cardio		HTTP		MNIST		Pendigits		Pima		Satellite		Satimage-2		Thyroid	
Model	Orig	Appr	Orig	Appr	Orig	Appr	Orig	Appr	Orig	Appr	Orig	Appr	Orig	Appr	Orig	Appr	Orig	Appr	Orig	Appr
ABOD	**0.31**	0.08	0.80	**0.83**	**0.27**	0.20	0.00	0.00	**0.40**	0.27	**0.05**	**0.05**	0.48	**0.52**	0.41	**0.46**	0.21	**0.64**	**0.36**	0.00
CBLOF	**0.25**	0.24	0.86	**0.90**	0.31	**0.34**	**0.02**	0.01	0.42	**0.48**	0.35	**0.36**	0.43	**0.48**	0.54	**0.57**	**0.96**	**0.96**	0.26	**0.38**
FB	**0.24**	0.02	0.03	**0.07**	0.23	**0.26**	0.02	**0.04**	0.34	**0.36**	0.03	**0.07**	0.37	**0.44**	0.37	**0.42**	0.03	**0.04**	**0.05**	0.02
kNN	0.30	**0.32**	**0.89**	**0.89**	0.37	**0.46**	**0.03**	**0.03**	0.42	**0.45**	**0.08**	0.06	**0.47**	**0.47**	0.49	**0.53**	0.32	**0.43**	0.33	**0.42**
AkNN	0.30	**0.33**	0.88	**0.89**	0.34	**0.40**	**0.03**	**0.03**	0.41	**0.45**	0.05	**0.13**	0.48	**0.49**	0.47	**0.52**	0.25	**0.43**	0.31	**0.44**
LOF	0.27	**0.36**	0.19	**0.35**	**0.23**	**0.23**	0.01	**0.03**	**0.33**	0.32	0.03	**0.08**	0.40	**0.44**	0.37	**0.42**	0.04	**0.07**	0.19	**0.25**

6.4.3 Q3: Time Reduction of Balanced Scheduling

To evaluate the effectiveness of the proposed BPS algorithm, we run the following experiments by varying: (i) the size (n) and the dimension (d) of the datasets, (ii) the number of estimators (m) and (iii) the number of CPU cores (t). Due to the space limit, we only show the training time comparison between the generic scheduling and BPS on **Cardio**, **Letter**, **PageBlock**, and **Pendigits**, by setting $m \in \{100, 500\}$ and $t \in \{2, 4, 8\}$, consistent with the single machine setting in real-world applications.

Table 6.5: Training time comparison (in seconds) between Simple scheduling and BPS against various numbers of OD models and workers. Percent of time reduction, Redu (%), is indicated in **bold**. BPS consistently outperforms Generic scheduling

Dataset	n	d	m	t	Generic	BPS	Redu (%)
Cardio	1831	21	500	2	240.12	221.34	**7.82**
Cardio	1831	21	500	4	185.44	154.43	**16.72**
Cardio	1831	21	500	8	140.63	120.02	**14.65**
Cardio	1831	21	1000	2	199.77	185.63	**7.08**
Cardio	1831	21	1000	4	130.82	110.60	**15.45**
Cardio	1831	21	1000	8	97.75	73.43	**24.88**
Letter	1600	32	500	2	111.95	109.52	**2.17**
Letter	1600	32	500	4	92.69	86.24	**6.94**
Letter	1600	32	500	8	57.21	48.72	**14.84**
Letter	1600	32	1000	2	224.61	222.59	**0.90**
Letter	1600	32	1000	4	228.08	172.07	**24.56**
Letter	1600	32	1000	8	109.50	89.51	**17.80**
PageBlock	5393	10	100	2	51.11	35.17	**31.19**
PageBlock	5393	10	100	4	42.49	16.23	**61.80**
PageBlock	5393	10	100	8	38.45	16.97	**55.86**
PageBlock	5393	10	500	2	197.84	137.46	**30.52**
PageBlock	5393	10	500	4	167.36	76.14	**54.51**
PageBlock	5393	10	500	8	127.08	66.29	**47.84**
Pendigits	6870	16	500	2	351.97	287.14	**18.42**
Pendigits	6870	16	500	4	288.51	146.50	**49.22**
Pendigits	6870	16	500	8	180.86	102.11	**43.33**
Pendigits	6870	16	1000	2	697.20	561.15	**19.51**
Pendigits	6870	16	1000	4	579.70	288.11	**50.33**
Pendigits	6870	16	1000	8	365.20	182.32	**50.08**

Table 6.5 shows that the proposed BPS has a clear edge over the generic scheduling mechanism (denoted as **Generic** in the tables) that equally splits the tasks by order. It yields a significant time reduction (denoted as **% Redu** in the table), which gets more remarkable if more cores are used along with large datasets. For instance, the time reduction is more than 40% on **PageBlock** and **Pendigits** when 8 cores are used. This agrees with our assumption that model cost vary more drastically on large datasets given the time complexity increase non-linearly to the size—the proposed BPS method is particularly helpful.

Table 6.6: Comparison between the baseline (denoted as _B) and SUOD (denoted as _S) regarding time cost, and prediction accuracy (ROC and P@N). The better method within each pair is indicated in **bold** (Optdigits fail to yield meaningful P@N). SUOD generally brings time reduction with no loss in prediction accuracy on a majority of datasets.

Data Information				Time Cost (in seconds)				Ensemble Model Performance (ROC)				Ensemble Model Performance (P@N)			
Dataset	*n*	*d*	*t*	Fit_B	Fit_S	Pred_B	Pred_S	Avg_B	Avg_S	MOA_B	MOA_S	Avg_B	Avg_S	MOA_B	MOA_S
Annthyroid	7200	6	5	73.91	**65.23**	47.48	**44.26**	0.91	**0.93**	0.91	**0.93**	0.46	**0.54**	0.46	**0.55**
Annthyroid	7200	6	10	71.00	**42.94**	44.68	**38.66**	0.91	**0.93**	0.92	**0.93**	0.46	**0.54**	0.46	**0.54**
Annthyroid	7200	6	30	42.80	**33.98**	30.92	**25.67**	0.91	**0.93**	0.92	**0.93**	0.46	**0.54**	0.46	**0.54**
Cardio	1831	21	5	**78.84**	79.70	**46.09**	46.68	0.91	**0.93**	0.91	**0.93**	0.46	**0.54**	0.45	**0.55**
Cardio	1831	21	10	72.04	**53.43**	44.57	**38.31**	0.91	**0.93**	0.91	**0.93**	0.46	**0.54**	0.46	**0.54**
Cardio	1831	21	30	47.53	**44.57**	**31.31**	31.43	0.91	**0.93**	0.92	**0.93**	0.46	**0.54**	0.46	**0.55**
MNIST	7603	100	5	856.53	**748.40**	453.39	**324.76**	0.77	**0.81**	0.77	**0.81**	0.29	**0.35**	0.28	**0.34**
MNIST	7603	100	10	726.76	**573.66**	367.85	**328.95**	0.78	**0.81**	0.78	**0.81**	0.29	**0.35**	0.30	**0.34**
MNIST	7603	100	30	357.40	**329.71**	260.80	**134.08**	0.78	**0.81**	0.78	**0.81**	0.29	**0.35**	0.29	**0.34**
Optdigits	5216	64	5	295.38	**267.71**	162.28	**149.19**	0.73	**0.75**	0.75	**0.77**	0.00	0.00	0.00	0.00
Optdigits	5216	64	10	247.24	**224.82**	136.12	**125.54**	0.73	**0.75**	0.74	**0.75**	0.00	0.00	0.00	0.00
Optdigits	5216	64	30	825.23	**791.95**	110.06	**62.63**	0.73	**0.75**	0.73	**0.76**	0.00	0.00	0.00	0.00
Pendigits	6870	16	5	287.75	**282.25**	184.20	**158.26**	0.92	**0.95**	0.92	**0.94**	0.19	**0.23**	0.19	**0.20**
Pendigits	6870	16	10	281.49	**155.06**	179.83	**160.94**	0.92	**0.95**	0.92	**0.94**	0.19	**0.25**	0.19	**0.23**
Pendigits	6870	16	30	149.93	**145.59**	104.25	**89.85**	0.92	**0.94**	0.93	**0.94**	0.19	**0.25**	0.19	**0.22**
Pima	768	8	5	**28.72**	31.94	**21.16**	23.79	**0.71**	**0.71**	**0.71**	0.70	**0.51**	**0.51**	**0.53**	0.51
Pima	768	8	10	27.38	**20.15**	**20.81**	25.03	**0.71**	0.70	**0.71**	0.70	**0.51**	**0.51**	**0.51**	**0.51**
Pima	768	8	30	19.36	**17.89**	**13.83**	17.43	**0.71**	0.70	**0.71**	0.70	**0.51**	0.50	**0.52**	0.50
Shuttle	49097	9	5	3326.54	**1453.93**	2257.50	**1956.12**	**0.99**	**0.99**	**0.99**	**0.99**	**0.95**	**0.95**	**0.95**	**0.95**
Shuttle	49097	9	10	2437.10	**1396.21**	1549.97	**1321.16**	**0.99**	**0.99**	**0.99**	**0.99**	**0.95**	**0.95**	**0.95**	**0.95**
Shuttle	49097	9	30	1378.29	**1258.69**	837.41	**651.00**	**0.99**	**0.99**	**0.99**	**0.99**	**0.95**	**0.95**	**0.95**	**0.95**
SpamSpace	4207	57	5	247.98	**244.39**	130.95	**110.08**	**0.57**	0.56	**0.56**	**0.56**	**0.45**	**0.45**	**0.46**	0.45
SpamSpace	4207	57	10	233.39	**186.91**	128.24	**115.83**	**0.57**	0.56	**0.56**	**0.56**	**0.46**	0.45	**0.46**	**0.46**
SpamSpace	4207	57	30	604.00	**538.91**	70.19	**61.38**	**0.57**	0.56	**0.57**	0.56	**0.46**	**0.46**	**0.46**	0.45
Thyroid	3772	6	5	87.90	**71.34**	49.51	**48.20**	0.91	**0.93**	0.91	**0.93**	0.46	**0.54**	0.46	**0.55**
Thyroid	3772	6	10	74.76	**46.91**	44.81	**38.60**	0.91	**0.93**	0.91	**0.93**	0.46	**0.54**	0.46	**0.54**
Thyroid	3772	6	30	45.84	**43.86**	28.90	**26.75**	0.91	**0.93**	0.92	**0.93**	0.46	**0.54**	0.46	**0.54**
Waveform	3443	21	5	167.98	**147.00**	109.94	**94.46**	**0.78**	0.76	**0.78**	0.76	0.11	**0.13**	0.11	**0.13**
Waveform	3443	21	10	154.72	**94.36**	91.69	**55.17**	**0.78**	0.76	**0.78**	0.77	**0.11**	**0.11**	**0.11**	**0.11**
Waveform	3443	21	30	97.11	**95.77**	53.47	**48.04**	**0.78**	0.76	**0.78**	0.76	0.11	**0.13**	0.11	**0.13**

6.4.4 Full System Evaluation with All Modules

Table 6.6 shows the performance of SUOD with all three modules enabled, even though not all of them are always needed in practice. In total, 600 hundred randomly selected OD models from `PyOD` are trained and tested on 10 datasets. To simulate the "worst-case scenario", we build the model pool $\mathcal{M}$ by randomly selecting OD models, which minimizes the intrinsic task load imbalance. In real-world applications, this order randomization may not be possible as discussed in §6.3.5—two adjacent models are often from the same algorithm family and more prone to scheduling imbalance. **Although this setting will make the effectiveness of BPS module less impressive, we choose it to provide an empirical worst-case performance guarantee—the framework should generally perform better in practice.**

SUOD consistently yields promising results even we deliberately choose the unfavored setting. **Fit_B** and **Pred_B** denote the fit and prediction time of the baseline setting (no compression, no approximation, generic parallel task scheduling; see §6.2.2). In comparison, SUOD (denoted as **Fit_S** and **Pred_S**) brings time reduction on the majority of the datasets with minor to no performance degradation. To measure the prediction performance, we measure the ROC and P@N by averaging the base model results (denoted as **Avg_**) and the maximum of average of the base models (denoted as **MOA_**), a widely used two-phase outlier score combination framework [9]. Surprisingly, SUOD even leads to a small performance boost in scoring new samples on most of the datasets (**Annthyroid**, **Cardio**, **MNIST**, **Optdigits**, **Pendigits**, and **Thyroid**). This performance gain may be jointly credited to the regularization effect by the randomness injected in JL projection (§6.3.3) and the pseudo-approximation (§6.3.4)—the baseline setting may be overfitted on certain datasets. It is noted that SUOD leads to more improvement on high-dimensional, large datasets. For instance, the fit time is significantly reduced on **Shuttle** (more than 50%). On the contrary, SUOD is less meaningful for small datasets like **Pima** and **Cardio**, although they may also yield per-

formance improvement. Again, our settings mimics the worst case scenario for SUOD (the model order is already randomly shuffled) but still observe a great performance improvement; real-world applications should generally expect more significant results.

6.4.5 Real-World Deployment: Fraudulent Medical Claim Analysis at IQVIA

Estimated by the United States Government Accountability Office and Federal Bureau of Investigation, healthcare frauds cost American taxpayers tens of billions dollars a year [17, 29]. Detecting fraudulent medical claims is crucial for taxpayers, pharmaceutical companies, and insurance companies. To further demonstrate SUOD's performance on industry data, we deploy it on a proprietary pharmacy claim dataset owned by IQVIA (a leading healthcare firm) consisting of 123,720 medical claims among which 19,033 (15.38%) are labeled as fraudulent. In each of the claims, there are 35 features including information such as drug brand, copay amount, insurance details, location, and pharmacy/patient demographics. The current system in use is based on a group of selected detection models in `PyOD`, and an averaging method is applied on top of the base model results as the initial result. The cases marked as high risk are then transferred to human investigators in the special investigation unit (SIU) for verification. It is important to provide prompt and accurate first-round screening for SIU, which leads to huge expense savings.

SUOD is applied on top of the aforementioned dataset (74,220 records are used for training and 49,500 records are set aside for validation). Similarly to the full framework evaluation in §6.4.4, the new system with SUOD (all three modules enabled) is compared with the current distributed system on 10 cores. The fit time is reduced from 6232.54 seconds to 4202.30 seconds (32.57% reduction), and the prediction time is reduced from 3723.45 seconds reduced to 2814.92 seconds (24.40%). In addition to the time reduction, ROC and P@N also show improvements at 3.59% and 7.46%, respectively. Through this case, we are confident the proposed framework can be useful for many real-world applications for scalable outlier detection.

6.5 Discussions

In this work, we propose SUOD to expedite the training and prediction of a large number of unsupervised heterogeneous outlier detection models. It consists of three modules with focus on different levels (data, model, execution): (i) Random Projection module compresses data into low-dimensional subspaces to alleviate the curse of dimensionality; (ii) Pseudo-supervised Approximation module could accelerate costly unsupervised models' prediction by replacing them with faster supervised regressors, which also brings the extra benefit, e.g., interpretability and (iii) Balanced Parallel Scheduling module ensures that a nearly equal amount of workload is assigned to available workers in distributed computing. The extensive experiments on more than 20 benchmark datasets and a real-world claim fraud analysis case show the great potential of SUOD, and many intriguing results are observed.

More investigations are currently underway. First, we plan to demonstrate SUOD's effectiveness as an end-to-end framework on more complex downstream combination models like unsupervised LSCP [346] and supervised XGBOD [344]. Second, we would further emphasize the interpretability provided by the pseudo-supervised approximation, which can be beyond the feature importance provided in tree regressors. Third, there is room to investigate why and how the pseudo-supervised approximation could work in a more strict and theoretical way. This study, as the first step, empirically shows that proximity-based models can benefit from the approximation. , whereas linear models may not. Lastly, we may incorporate the emerging automated OD, e.g., MetaOD [348], to trim down the model space for further acceleration.

Table 6.7: Comparison of various data compression methods on different outlier detectors and datasets. Each column corresponds to an evaluation metric (execution time is measured in seconds); the best performing method is indicated in **bold**. JL projection methods, especially *circulant* and *toeplitz*, outperform w.r.t time cost and prediction accuracy.

(a) ABOD on **Cardio**

Method	Time	ROC	P@N
original	0.98	0.59	0.25
PCA	**0.82**	0.59	0.26
RS	0.92	**0.63**	**0.29**
basic	0.83	0.62	0.28
discrete	**0.82**	0.62	0.28
circulant	0.83	0.62	0.27
toeplitz	0.83	0.62	0.28

(b) LOF on **Cardio**

Method	Time	ROC	P@N
original	0.08	0.55	0.17
PCA	**0.04**	0.56	0.19
RS	**0.04**	0.57	0.15
basic	**0.04**	**0.60**	0.20
discrete	**0.04**	0.59	0.19
circulant	**0.04**	0.59	0.20
toeplitz	**0.04**	**0.60**	**0.21**

(c) kNN on **Cardio**

Method	Time	ROC	P@N
original	0.09	0.71	0.34
PCA	**0.03**	0.73	0.34
RS	**0.03**	0.69	**0.38**
basic	**0.03**	**0.74**	0.35
discrete	**0.03**	**0.74**	0.37
circulant	**0.03**	**0.74**	0.34
toeplitz	**0.03**	0.73	0.35

(d) ABOD on **MNIST**

Method	Time	ROC	P@N
original	12.89	0.80	**0.39**
PCA	8.93	**0.81**	0.37
RS	**8.27**	0.74	0.32
basic	8.94	0.80	0.38
discrete	8.86	0.80	**0.39**
circulant	9.33	0.80	0.38
toeplitz	8.96	0.80	0.38

(e) LOF on **MNIST**

Method	Time	ROC	P@N
original	7.64	0.68	0.29
PCA	4.92	0.67	0.27
RS	**3.65**	0.63	0.23
basic	4.87	0.70	0.31
discrete	5.21	0.70	**0.32**
circulant	5.06	0.69	0.31
toeplitz	4.97	**0.71**	0.31

(f) kNN on **MNIST**

Method	Time	ROC	P@N
original	7.13	**0.84**	**0.42**
PCA	3.92	**0.84**	0.40
RS	**3.33**	0.77	0.34
basic	4.17	**0.84**	**0.42**
discrete	4.11	**0.84**	0.41
circulant	4.13	**0.84**	0.41
toeplitz	4.11	**0.84**	**0.42**

(g) ABOD on **Satellite**

Method	Time	ROC	P@N
original	4.03	0.59	0.41
PCA	**3.01**	0.62	0.44
RS	3.53	0.63	0.44
basic	3.10	0.64	0.45
discrete	3.12	0.65	0.46
circulant	3.14	**0.66**	**0.48**
toeplitz	3.14	**0.66**	0.47

(h) LOF on **Satellite**

Method	Time	ROC	P@N
original	0.82	**0.55**	0.37
PCA	**0.23**	0.54	0.36
RS	0.39	0.54	0.37
basic	0.31	0.54	0.37
discrete	0.32	0.54	0.37
circulant	0.39	**0.55**	**0.38**
toeplitz	0.37	0.54	0.37

(i) kNN on **Satellite**

Method	Time	ROC	P@N
original	0.71	0.67	0.49
PCA	**0.18**	0.67	0.50
RS	0.31	0.68	0.49
basic	0.24	0.68	0.49
discrete	0.25	0.69	0.50
circulant	0.33	**0.70**	0.50
toeplitz	0.30	**0.70**	**0.51**

(j) ABOD on **Satimage-2**

Method	Time	ROC	P@N
original	3.68	0.85	0.28
PCA	**2.70**	0.88	0.30
RS	3.20	0.89	0.28
basic	2.78	0.91	0.29
discrete	2.79	0.91	**0.31**
circulant	2.85	0.91	0.29
toeplitz	2.83	**0.92**	0.30

(k) LOF on **Satimage-2**

Method	Time	ROC	P@N
original	0.79	0.54	0.07
PCA	**0.20**	0.52	0.04
RS	0.37	0.53	0.08
basic	0.29	0.52	0.08
discrete	0.30	0.53	0.07
circulant	0.43	**0.59**	**0.11**
toeplitz	0.32	0.54	0.09

(l) kNN on **Satimage-2**

Method	Time	ROC	P@N
original	0.68	0.94	**0.39**
PCA	**0.15**	0.94	**0.39**
RS	0.29	0.94	0.38
basic	0.23	0.94	0.38
discrete	0.20	0.95	0.37
circulant	0.36	**0.96**	0.37
toeplitz	0.25	**0.96**	**0.39**

7

TOD: Efficient and Scalable Outlier Detection on Multiple GPUs

This chapter is primarily based on:

Yue Zhao*, George H. Chen, Zhihao Jia. "TOD: GPU-accelerated Outlier Detection via Tensor Operations." *International Conference on Very Large Data Bases (VLDB)*, 2023.

7.1 Highlight

As discussed in §0.5, it is critical to design systems that support diverse OD algorithms for hardware acceleration. In this chapter, we demonstrate TOD, the first tensor-based system for efficient and scalable outlier detection on **distributed multi-GPU machines**. A key idea behind TOD is decomposing complex OD applications into a small collection of basic tensor algebra operators. This decomposition enables TOD to accelerate OD computations by leveraging recent advances in deep learning infrastructure in both hardware and software.

Advances in systems for deep neural networks. Deep neural networks (DNNs) have revolutionized computer vision, natural language processing, and various other fields [94] over the last decade. This success is largely due to the recent development of DNN systems (e.g., TensorFlow [2] and PyTorch [233]). These systems enable fast tensor algebra computations (e.g., matrix multiplication, convolution, etc.) on modern hardware accelerators (e.g., GPUs and TPUs) and use efficient parallelization strategies (e.g., data, model, and pipeline parallelism [128, 210, 289]) to aggregate the compute resources across multiple accelerators, enabling efficient and scalable DNN computations.

this chapter explores a new approach to building GPU-accelerated OD systems. Instead of following the methodology used in existing GPU-based OD frameworks (i.e., providing efficient GPU implementations tailored to specific OD applications), we ask: *can we leverage the compilation and optimization techniques in DNN systems to minimize the time and memory consumption of a wide range of common OD computations?*

Our Approach. In this chapter, we present TOD, a tensor-based outlier detection system that abstracts OD applications into tensor algebra operations for efficient GPU acceleration. TOD leverages both the software and hardware optimizations in modern DNN frameworks to enable efficient and scalable OD computations on distributed multi-GPU clusters. To the best of our knowledge, TOD is the *first* GPU-based system for *generic* OD applications. Fig. 7.1 shows an overview of

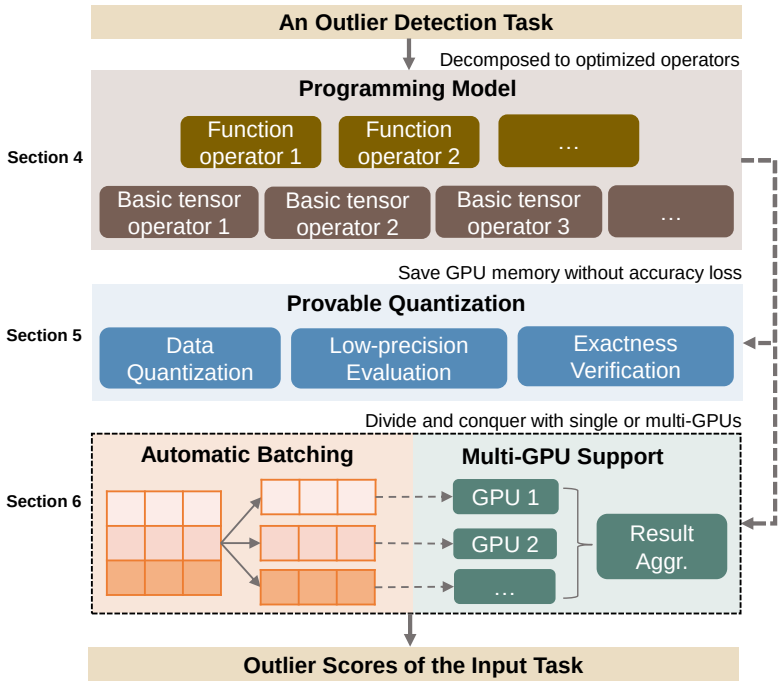


Figure 7.1: TOD's overview. TOD decomposes an OD task into fine-grained tensor operators and optimizes OD computations across multiple GPUs using provable quantization and automatic batching.

TOD. Building a tensor-based OD system requires addressing three major obstacles.

Representing OD computations using tensor operations. Unlike DNN models, which are represented as a pipeline of tensor algebra operators (e.g., matrix multiplication), many OD applications involve a diverse collection of operators that have traditionally not been implemented in terms of tensor operations, such as proximity-based algorithms, statistical approaches, and linear methods (see an overview of OD applications in §7.2.1). Implementing OD applications one at a time and accounting for software and hardware optimizations is labor-intensive. To address this obstacle, TOD introduces a new programming model for OD that decomposes a broad set of OD applications into a small collection of *basic tensor operators* and *functional operators*, which significantly reduces the implementation and optimization effort and opens the possibility of easily supporting new OD algorithms.

Quantizing OD computations. *Quantization* is commonly used in existing DNN frameworks

to reduce the run time and memory consumption of DNN computations by computing intermediate results in a DNN model using a lower-precision floating-point representation. Quantization in general does not preserve the end-to-end equivalence of a DNN model and therefore may introduce potential accuracy losses. To apply quantization, the current practice is fine-tuning a quantized DNN model on the training dataset in a *supervised* fashion and assessing the accuracy loss; however, this approach is not directly applicable to OD, since most OD algorithms are *unsupervised* and thus lack a direct way for measuring accuracy. To address this challenge, TOD introduces a novel quantization technique called *provable quantization*, which leverages the numerical insensitivity of OD algorithms (e.g., k-nearest-neighbors may return identical results for some samples when computing with different floating-point precisions) and *automatically* performs specific OD operators in lower precision. In contrast to prior quantization techniques that use lower precision calculations at the expense of accuracy, provable quantization guarantees *no* accuracy loss.

Enabling scalable OD computations. Existing DNN frameworks cannot directly support large-scale OD applications, since modern DNN systems are designed to iteratively process a small *batch* of training samples even though the entire training dataset can be large. For example, to train ResNet-50 [109] on the ImageNet dataset [66], existing DNN systems only handle small mini-batches (e.g., 256 samples) in each training iteration, while the dataset contains more than 14 million samples. However, many OD applications involve operating on *all* samples, such as computing distances between all sample pairs. Executing such an application on a single GPU would typically run out of memory because GPUs nowadays have limited memory capacities (e.g., compared to those of CPU DRAM). To overcome the difficulty of executing OD applications in iterations and the resource limits of a single GPU, TOD uses an *automatic batching* mechanism to execute memory-intensive OD operators in small batches, which are distributed across *multiple* GPUs in parallel in a pipeline fashion. The automatic batching and multi-GPU support allow TOD to scale to datasets as large as those commonly encountered in deep learning tasks.

We compare TOD against existing CPU- and GPU-based OD systems on both real-world and synthetic datasets. TOD is on average 10.9× faster than PyOD, a state-of-the-art comprehensive CPU-based OD system [347], and can process a million samples within an hour while PyOD cannot. Compared to existing GPU-based OD systems, TOD can handle much larger datasets, while the GPU baselines run out of GPU memory. Our evaluation further shows that provable quantization, automatic batching, and multi-GPU support are all critical for efficient and scalable OD computations.

In summary, this chapter makes the following contributions:

- We propose TOD, the first tensor-based system for generic outlier detection, enabling efficient and scalable OD computations on distributed multi-GPU machines.
- TOD uses a new programming model that abstracts complex OD applications into a small collection of basic tensor operators for efficient GPU acceleration.
- We introduce provable quantization that accelerates unsupervised OD computations by performing specific floating-point operators in lower precision while provably guaranteeing no accuracy loss.

Extensibility and integration. TOD is open-sourced*, which enables easy development of new OD algorithms by leveraging highly optimized tensor operators or including new operators (see examples in §7.7.1). This extensibility yields a large number of yet-to-be-discovered OD methods. Thus, we believe that TOD also provides a platform that enables rapid research and development of new OD methods.

*Open-sourced library and online appendix: https://github.com/yzhao062/pytod

Table 7.1: Key OD algorithms for tabular data and their time and space complexity with a *brute-force* implementation (additional optimization is possible but not considered here), where n is the number of samples, and d is the number of dimensions. Note that ensemble-based methods' complexities depend on the underlying base estimators. Algorithms that can be accelerated in TOD are marked with ✓.

Category	Algorithm	Time Compl.	Space Compl.	Optimized in TOD
Proximity	kNNOD	$O(n^2)$	$O(n^2)$	✓
Proximity	COF	$O(n^3)$	$O(n^2)$	✓
Proximity	LOF	$O(n^2)$	$O(n^2)$	✓
Proximity	LOCI	$O(n^2)$	$O(n^2)$	✓
Statistical	KDE	$O(n^3)$	$O(n^2)$	✗
Statistical	HBOS	$O(nd)$	$O(nd)$	✓
Statistical	COPOD	$O(nd)$	$O(nd)$	✓
Statistical	ECOD	$O(nd)$	$O(nd)$	✓
Ensemble	LODA	N/A	N/A	✓
Ensemble	FB	N/A	N/A	✓
Ensemble	iForest	N/A	N/A	✗
Ensemble	LSCP	N/A	N/A	✓
Linear	PCA	$O(nd)$	$O(n)$	✓
Linear	OCSVM	$O(n^3)$	$O(n^2)$	✗

7.2 Related Work

In this section, §7.2.1 summarizes existing OD algorithms for tabular data, §7.2.2 introduces existing DNN infrastructure, which is leveraged by TOD to accelerate OD computations, and §7.2.3 describes modern OD systems. Note that TOD primarily focuses on OD in tabular data due to its popularity [7]; TOD can be extended to support OD in other data types with modifications.

7.2.1 Existing OD Algorithms and Scalability

Outlier detection (also called anomaly detection) is a key machine learning task that aims to find data points that deviate from a general distribution [7, 105]. As shown in Table 7.1, non-deep-learning OD algorithms are grouped into four categories (see the book by Aggarwal [7] for more details on algorithms): (*i*) proximity-based algorithms that rely on measuring sample similarity including kNN [26], ABOD [145], COF [285], LOF [46], and LOCI [231]; (*ii*) statistical approaches including KDE [260], HBOS [92], COPOD [174], and ECOD [175]; (*iii*) ensemble-based meth-

ods that build a collection of detectors for aggregation like iForest [180], LODA [236], and LCSP [346]; and (*iv*) linear models such as PCA [272].

Many OD algorithms suffer from scalability issues [219, 345]. For example, Table 7.1 shows that various proximity-based OD algorithms have at least $O(n^2)$ time and space complexities—they all require estimating and storing pairwise distances among all n samples. The high time complexities of many OD algorithms limit their applicability in real-world applications that require either real-time responses (e.g., fraud detection [185]) or the concurrent processing of millions of samples [355]. As shown in the table, TOD supports nearly all the OD algorithms mentioned.

7.2.2 DNN Infrastructure and Acceleration

Deep neural networks have dramatically improved the accuracy of artificial intelligence systems across numerous fields [94, 228]. Its success is fueled by recent advances in both DNN hardware and software [158]. Specifically, DNN systems depend on tensor operations that can often be parallelized and executed in small batches. These operations are well-suited for GPUs, especially as a single GPU nowadays often has many more cores than a single CPU; while GPU cores are not as general purpose as CPU cores, they suffice in executing the tensor operations of deep learning. Moreover, the maturity of DNN programming frameworks such as PyTorch [233] and TensorFlow [2] makes developing machine learning models easy with a wide range of GPUs. Multiple works attempt to leverage DNN systems for accelerating training data science and ML tasks, including Hummingbird [209], Tensors [142], and AC-DBSCAN [126].

Differently, TOD for the first time, extends the acceleration usage of DNN systems to OD algorithms. We build TOD using the DNN ecosystem, taking advantage of its established hardware acceleration and software accessibility. This design choice also opens the opportunity for unifying classical OD algorithms (see §7.2.1) and DNN-based OD algorithms on the same platform—this emerging direction has gained increasing attention in OD research [255].

7.2.3 Outlier Detection Systems

CPU-based systems . Over the years, comprehensive OD systems on CPUs that cover a diverse group of algorithms have been developed in different programming languages, including ELKI Data Mining [5] and RapidMiner [249] in Java, and PyOD [347] in Python. Among these, PyOD is the state-of-the-art (SOTA) with deep optimization including just-in-time compilation and parallelization. It is widely used in both academia and industry, with hundreds of citations [341] and millions of downloads per year [274]. Recently, the PyOD team proposed an acceleration system called SUOD to further speed up the training and prediction in PyOD with a large collection of heterogeneous OD models [345]. Specifically, SUOD uses algorithmic approximation and efficient parallelization to reduce the computational cost and therefore runtime. There are other distributed/parallel systems designed for specific (family of) OD algorithms with non-GPU nodes (e.g., CPUs): (*i*) Parallel Bay, Parallel LOF, DLOF for local OD algorithms [188, 218, 327], (*ii*) DOoR for distance-based OD [37], (*iii*) distributed OD for mixed-attributed data [220] (*iv*) PROUD for stream data [288] and (*v*) Sparx for Apache Spark [338]. These distributed non-GPU systems do not constitute as baselines as TOD is a comprehensive system covering different types OD algorithms, while the specialized systems only cover specific algorithms. Thus, we consider the SOTA comprehensive system, PyOD, as the primary baseline (see exp. results in §7.7.3).

GPU-based systems . There are efforts to use GPUs for fast OD calculations for LOF [21], distance-based methods [25], KDE [28], and data stream [157]. These approaches rely on exploring the characteristics of a specific OD algorithm for GPU acceleration. This limits their generalization to a wide collection of OD algorithms. Furthermore, none has direct multi-GPU support, leading to limited scalability. To the best of our knowledge, there is no existing comprehensive GPU-based OD system that covers a diverse group of algorithms. Thus, we use direct GPU implementations of OD algorithms and selected works above as GPU baselines when appropriate (see details in §7.7.2 and

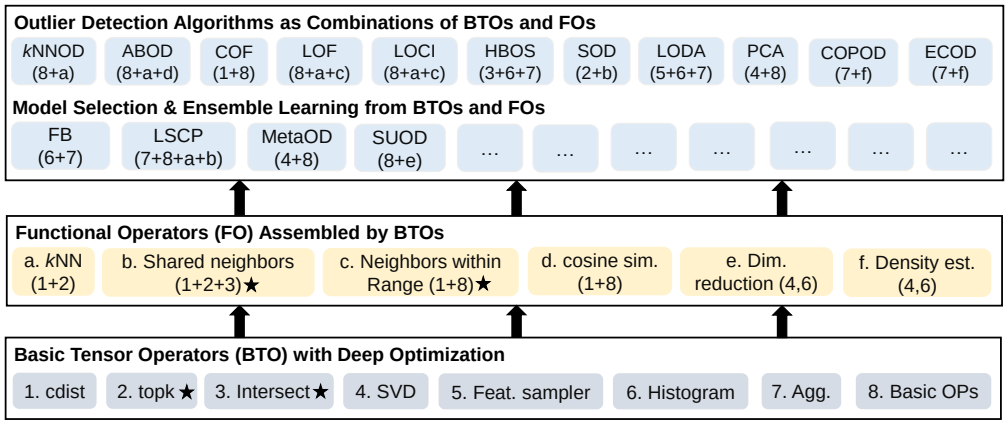


Figure 7.2: With algorithmic abstraction, more than 20 OD algorithms (denoted by blue squares) are abstracted into eight basic tensor operators (in yellow squares) and six functional operators (in gray squares). This abstraction reduces the implementation and optimization effort, and opens the possibility of including new algorithms. All operators are executed on GPUs using automatic batching (see §7.6), and operators marked with ★ are further accelerated using provable quantization (see §7.5).

Table 7.3).

7.3 System Overview

7.3.1 Definition and Problem Formulation

An OD system supports a collection of OD models $\mathcal{M} = \{M_1, ..., M_m\}$ such that given a user-specified OD model $M \in \mathcal{M}$ and an input dataset $\mathbf{X} \in \mathbb{R}^{n \times d}$ *without* ground truth labels (rows of $\mathbf{X}$ are data points, while columns are features), the system outputs outlier scores $\mathbf{O} := M(\mathbf{X}) \in \mathbb{R}^n$, which should be roughly deterministic and irrespective of the underlying system (higher values in $\mathbf{O}$ correspond to data points more likely to be identified as outliers; threshold on outlier scores can be used to determine which points are outliers). Given a hardware configuration $\mathcal{C}$ (e.g., CPUs and GPUs), the system's performance can be measured in *efficiency* (both runtime and memory consumption).

7.3.2 TOD's Overview

TOD is a comprehensive (i.e., covering a diverse group of methods) OD system as outlined in Fig. 7.1 and Table 7.1. For an outlier detection task, TOD decomposes it into a combination of predefined tensor operators via the proposed *programming model* for direct GPU acceleration (§7.4). Notably, TOD opportunistically performs *provable quantization* on tensor operators to enable faster computation and reduce memory requirements, while provably maintaining model accuracy (§7.5). To overcome the resource limitation of a single GPU, we further introduce *automatic batching* and *multi-GPU* support in TOD (§7.6).

7.4 Programming Model

Motivation. As a comprehensive system, TOD aims to include a diverse collection of OD algorithms, including proximity-based methods, statistical methods, and more (see §7.2.1). However, not all algorithms can be directly converted into tensor operations for GPU acceleration. A key design goal of TOD is to support various OD algorithms by piecing together commonly recurring building blocks. In particular, rather than manually implementing many OD algorithms, which is a labor-intensive process, we instead define OD algorithms as compositions of basic OD building blocks, each of which only needs to be implemented once. Moreover, the building blocks can be optimized independently.

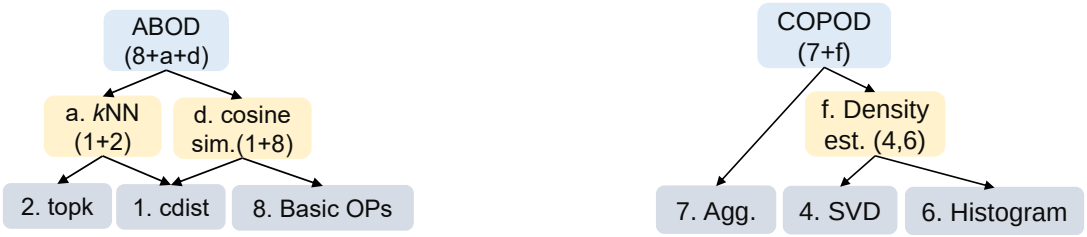
7.4.1 Algorithmic Abstraction

The key idea of our programming model is to decompose existing OD algorithms into a set of low-level *basic tensor operators* (BTOs), which can directly benefit from GPU acceleration. On top of these BTOs, we introduce higher-level OD operators called *functional operators* (FOs) with richer semantics. Consequently, OD algorithms can be constructed as combinations of BTOs and FOs.

Fig. 7.2 shows the hierarchy of TOD's programming abstraction in a bottom-up way, with increasing dependency: 8 BTOs are first constructed as the foundation of TOD (shown at the bottom in gray), while 6 FOs are then created on top of them (shown in the middle). Finally, OD algorithms and key functions on the top of the figure can be assembled using BTOs and FOs. In other words, TOD represents an OD algorithm using a tree-structured dependency graph, where the BTOs (as leaves of the tree) are fully independent for deep optimization, and all the algorithms depending on these BTOs can be collectively optimized. Clearly, this abstraction process reduces the repetitive implementation and optimization effort, and improves TOD's efficiency and generalizability. Additionally, it also facilities fast prototyping and experimentation with new algorithms.

7.4.2 Building End-to-end OD Algorithms

It is easy to build an end-to-end OD application by constructing its computation graph using FOs and BTOs. For instance, angle-based outlier detection (ABOD) is a classical OD algorithm [145], where each sample's outlier score is calculated as the average cosine similarity of its neighbors. Fig. 7.3(a) highlights an implementation of ABOD that uses an FO (i.e., *k*NN) to obtain a list of neighbors for each sample and then applies another FO (i.e., `cosine sim.`) for calculating cosine similarity. Note that each FO is a combination of BTOs. For example, *k*NN is implemented as calculating pairwise sample distance using `cdist` and then identifying the *k* "neighbor" with smallest distances using `topk`. Additionally, Fig. 7.3(b) shows the abstraction graph of copula-based outlier detection (COPOD) [174]. Other OD algorithms follow the same abstraction protocol and are represented as combinations of BTOs and FOs.



(a) Abstraction of ABOD

(b) Abstraction of COPOD

Figure 7.3: Examples of building complex OD algorithms with FO and BTO conveniently.

7.5 Provable Quantization

Motivation. OD operators mainly depend on floating-point arithmetic, namely $\{+, -, \times, /\}$. For these operations, the main source of imprecision is rounding [164]. Rounding errors enlarge when storing numbers using fewer bits, e.g., 16-bit floating-points lead to more inaccuracy than 64-bit floating-points. Therefore, many machine learning algorithms use high-precision floating-points when possible to minimize the impact of rounding errors. However, using high-precision floating-points can increase computation time and memory consumption. This is especially critical for GPUs with limited memory.

To reduce memory usage and run time, *quantization* has been applied to many machine learning algorithms [41] and data-driven applications [23, 301, 303]. Simply put, it refers to executing an operator (function) with lower-precision floating representations. If we denote the original function by $r(x)$ and its quantization by $r_q(x)$, the rounding error $\mathrm{Err}(\cdot)$ of quantization is defined as the output difference between $r(x)$ and $r_q(x)$, namely $\mathrm{Err}(r_q(x)) = r(x) - r_q(x)$. Intuitively, quantization can save memory at the cost of accuracy. How to balance the tradeoff between the *memory cost* and *algorithm accuracy* is a key challenge for quantizing in machine learning [61]. In supervised ML, one may measure the inaccuracy caused by quantization using ground truth labels. However, this is infeasible under unsupervised OD settings, where no ground truth labels is available for evaluation as described in §7.3. Thus, existing quantization techniques for supervised ML do not suit the need

of unsupervised OD.

In TOD, we design a correctness-preserving quantization technique for (unsupervised) OD applications, termed *provable quantization*. The key idea behind provable quantization is that depending on the operator used, it is possible to apply quantization to reduce memory consumption with *no* loss in accuracy. As a motivating example, consider the sign function $r(x)$ that returns "+" if $x > 0$ and returns "−" otherwise. Clearly, even if we quantize x to have a single bit (that precisely indicates the sign of x), we can achieve an exact answer for $r(x)$ that is the same as if instead x had more bits. Similarly, the ranking between two floating-point numbers often only depends on the most significant digits. Building on this simple intuition, we introduce *provable quantization* for a collection of OD operators, where the output and accuracy of the operators remain provably unchanged before and after quantization.

7.5.1 $(1+\varepsilon)$-property for Rounding Errors

Provable quantization relies on a standard analysis technique for floating-point numbers called the "$(1+\varepsilon)$-property" [164]. Let $\mathbb{F}$ denote the set of 64-bit floating-point numbers. For $x, y \in \mathbb{F}$, we define the floating-point operation "$\circledast$" as $x \circledast y \triangleq \mathrm{fl}(x * y)$, where $* \in \{+, -, \times, /\}$ and $\mathrm{fl}(\cdot)$ refers to the IEEE 754 standard for rounding a real number to a 64-bit floating-point number [133]. For example, $\oplus$ is floating-point addition and $\otimes$ is floating-point multiplication. The standard technique for calculating the rounding errors in floating-point operations is the $(1+\varepsilon)$-property [164], which is formally defined as follows.

Theorem 1 (Theorem 3.2 of Lee et al. 164) *Let $x, y \in \mathbb{F}$, and $* \in \{+, -, \times, /\}$. Suppose that $|x * y| \leq \max \mathbb{F}$. Then when we compute $x * y$ in floating-point, there exist multiplicative and additive error terms $\delta \in \mathbb{R}$ and $\delta' \in \mathbb{R}$ respectively such that*

$$x \circledast y = (x * y)(1+\delta) + \delta', \quad \textit{where } |\delta| \leq \varepsilon, |\delta'| \leq \varepsilon'. \tag{7.1}$$

In the above equation, ε and ε' are constants that do not depend on x or y. For instance, when working with 64-bit floating-point numbers, $\varepsilon = 2^{-53}$ and $\varepsilon' = 2^{-1075}$.

As discussed by Lee et al. [164, Section 5], this property can be further simplified when the exact result of the floating operation is not in the so-called "subnormal" range: the addictive error term δ' can be soundly removed, leading to a simplified $(1+\varepsilon)$-property:

$$x \circledast y = (x * y)(1 + \delta), \quad \text{where } |\delta| \leq \varepsilon. \tag{7.2}$$

7.5.2 Provable Quantization in TOD

TOD applies provable quantization for an applicable operator $r(\cdot)$ with input x in three steps: (*i*) input quantization, (*ii*) low-precision evaluation, and (*iii*) exactness verification and, if needed, recalculation. In a nutshell, the input is first quantized into a lower precision, and then the operator is evaluated in the lower precision, where $r(x)$ is evaluated as $r_q(x)$. To verify the exactness of the quantization, we calculate the rounding error $\text{Err}(r_q(x))$ by the simplified $(1+\varepsilon)$-property in eq. (7.2) and then check whether the result of $r(x)$ may change with the rounding error. If it passes the verification, then we output $\text{Err}(r_q(x))$ as the final result; otherwise, we use the original precision of x to recalculate $r(x)$. Note that we apply this technique to the entire input data $\mathbf{X} \in \mathbb{R}^{n \times d}$, and only need to recalculate on the subset of $\mathbf{X}$ where the verification fails. A limitation of provable quantization is that it does not apply to all operators; we elaborate on this criteria in §7.5.4.

7.5.3 Case Study: Neighbors Within Range

We show the usage of provable quantization in TOD on neighbors within range (`NWR`, one of the FOs of our programming model), a common step in many OD algorithms, e.g., LOF [46] and LOCI [231]. `NWR` identifies nearest neighbors within a preset distance threshold (usually a small

number), which may be considered as a variant of k nearest neighbors. More formally, given an input tensor $\mathbf{X} := [X_1, X_2, \ldots, X_n] \in \mathbb{R}^{n \times d}$ (n samples and d dimensions) and the distance threshold φ, `NWR` first calculates the pairwise distance among each sample via the `cdist` operator, yielding a distance matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$, where $\mathbf{D}_{i,j}$ is the pairwise distance between X_i and X_j. Then, each pairwise distance in $\mathbf{D}$ is compared with φ, and `NWR` outputs the indices of samples where $\mathbf{D}_{i,j} \leq \varphi$. As the pairwise distance calculation in `NWR` requires $O(n^2)$ space, provable quantization can provide significant GPU memory savings.

`NWR` meets the criterion we outline in §7.5.2, where eq. (7.2) can be applied to estimate the rounding errors. Recall that the pairwise Euclidean distance between two samples is

$$\mathbf{D}_{ij} = \left\|X_i - X_j\right\|_2^2 = \|X_i\|_2^2 + \left\|X_j\right\|_2^2 - 2X_i^T X_j. \tag{7.3}$$

We shall compute this in floating-point. Importantly, in our analysis to follow, the ordering of floating-point operations matters in determining rounding errors. To this end, we calculate distance using the right-most expression in eq. (7.3) (note that we do *not* first compute the difference $X_i - X_j$ and then compute its squared Euclidean norm). When calculating the first term in the RHS of eq. (7.3) via floating-point operations, we get

$$\begin{aligned} \text{fl}(\|X_i\|_2^2) &= (X_{i,1} \otimes X_{i,1}) \oplus (X_{i,2} \otimes X_{i,2}) \oplus \cdots \oplus (X_{i,d} \otimes X_{i,d}) \\ &= (X_{i,1}^2(1+\delta_1)) \oplus (X_{i,2}^2(1+\delta_2)) \oplus \cdots \oplus (X_{i,d}^2(1+\delta_d)), \end{aligned}$$

where the second equality uses eq. (7.2) and we note that the errors $\delta_1, \ldots, \delta_d$ across the floating-point multiplications (for squaring) need not be the same (in fact, these need not be the same across samples $i = 1, 2, \ldots, n$ but we omit this indexing to keep the equation from getting cluttered).

Next, by defining $x_{\max} \triangleq \max_{i \in \{1,\ldots,n\}, k \in \{1,\ldots,d\}} |X_{i,k}|$ and recalling from Theorem 1 that each of $\delta_1, \delta_2, \ldots, \delta_d$ above is at most ε, we get

$$
\begin{aligned}
\text{fl}(\|X_i\|_2^2) &= (X_{i,1}^2(1+\delta_1)) \oplus (X_{i,2}^2(1+\delta_2)) \oplus \cdots \oplus (X_{i,d}^2(1+\delta_d)) \\
&\leq \underbrace{[x_{\max}^2(1+\varepsilon)] \oplus [x_{\max}^2(1+\varepsilon)] \oplus \cdots \oplus [x_{\max}^2(1+\varepsilon)]}_{d \text{ terms added via floating-point addition}} \\
&\leq d \cdot x_{\max}^2 (1+\varepsilon)^{1+\lceil \log_2 d \rceil},
\end{aligned}
$$

where for the last step, $\log_2 d$ shows up since summation of d elements in lower-level programming languages is implemented in a divide-and-conquer manner that reduces to $\lceil \log_2 d \rceil$ operations (there is still a "1+" term in the exponent for the floating-point multiplication/squaring). The rounding error is bounded as follows:

$$
\begin{aligned}
\text{Err}(\|X_i\|_2^2) &= \|X_i\|_2^2 - \text{fl}(\|X_i\|_2^2) \\
&\leq d \cdot x_{\max}^2 - \text{fl}(\|X_i\|_2^2) \\
&\leq \left| d \cdot x_{\max}^2 - \text{fl}(\|X_i\|_2^2) \right| \\
&\leq d \cdot x_{\max}^2 [(1+\varepsilon)^{1+\lceil \log_2 d \rceil} - 1].
\end{aligned}
$$

This same analysis can be used to bound the floating-point errors of the other terms in eq. (7.3). Overall, we get

$$
\text{Err}(\mathbf{D}_{ij}) \leq 4d \cdot x_{\max}^2 [(1+\varepsilon)^{1+\lceil \log_2 d \rceil + 2} - 1], \tag{7.4}
$$

where the "+2" shows up in the exponent due to the addition and subtraction in the RHS of (7.3) that we compute in floating-point.

Inequality (7.4) provides a numerical way for checking whether a single entry $\mathbf{D}_{ij}$ is within the range of φ as $|\mathbf{D}_{ij} - \varphi| > \text{Err}(\mathbf{D}_{ij})$. More conveniently, we could scale the input $\mathbf{X}$ into the range of $[0, 1]$ before the distance calculation [244], so that $x_{\max} \leq 1$ and the implementation complexity can be further reduced. With this treatment, a large amount of GPU memory can be saved in `NWR` operations (see §7.7.5 for results).

7.5.4 Applicability and Opportunities of Provable Quantization

Not all operators can benefit from provable quantization. To benefit from provable quantization, an operator needs to satisfy two criteria. First, the operator's output values cannot require a floating-point representation in the original precision, otherwise the exactness verification would require executing the operator in the original precision, resulting in no memory or time savings. For example, provable quantization is not applicable to `cdist` since its outputs are raw floating-point pairwise distances, and verifying its exactness requires calculating `cdist` in the original precision. Second, the performance gain in low-precision evaluation of the operator needs to be larger than the overhead of verification. Based on these two criteria, we mark the operators generally applicable for provable quantization by ★ in Fig. 7.2. Also see §7.7.5 for experimental results on this. Although the design of provable quantization is motivated by unsupervised OD algorithms with extensive ranking and selecting operations, other ML algorithms can also benefit if they meet the above criteria. For OD algorithms in which provable quantization does not apply, they could still benefit from TOD's other optimizations such as automatic batching (§7.6).

7.6 Automatic Batching and Multi-GPU Support

Motivation. Unlike CPU nodes with up to terabytes of DRAM, today's GPU nodes face much more stringent memory limitations [203, 90, 204, 163]—modern GPUs are mostly equipped with 4-40 GB of on-device memory [200, 283]. Out-of-memory (OOM) errors have thus become common in GPU-based systems for machine learning tasks [88]. To overcome this challenge, we design an *automatic batching* mechanism to decompose memory-intensive BTOs into multiple batches, which are executed on GPUs in a pipeline fashion. Automatic batching also allows TOD to equally distribute OD computation across multiple GPUs. TOD applies different mechanisms to decompose an operator into batches based on its data dependency. Specifically, an operator has *inter-*

sample dependency if the computation of each sample requires accessing other samples, such as `cdist`. On the other hand, an operator has *inter-feature dependency* if the computation of each feature depends on other features, such as `Feat. sampler`. Fig. 7.4 summarizes TOD batching mechanisms for operators with and without these dependencies.

Direct batching. TOD automatically decomposes an operator into small batches if the operator is: (*i*) *sample-independent*: the estimation of each sample is independent, or (*ii*) *feature-independent*: the contribution of each feature is independent. When either condition holds, TOD directly partitions the operator into multiple batches by splitting along the sample or feature dimension, computes these batches in a pipeline fashion, and aggregates individual batches to produce the final output, as shown in Fig. 7.5. For instance, `topk` is a sample-independent operator. For an input tensor $\mathbf{X} \in \mathbb{R}^{n \times d}$, `topk` outputs the indices of the largest k values for each sample (i.e., row), resulting in an output tensor $\mathbf{I}_\mathbf{X} := \texttt{topk}(\mathbf{X}, k) \in \mathbb{R}^{n \times k}$. Therefore, we could split $\mathbf{X}$ into batches with full features, each with b samples (i.e.,, $\{\mathbf{X}_1, \mathbf{X}_2, ...\} \in \mathbb{R}^{b \times d}$). As another example, `Histogram` outputs the frequencies of each feature's values, which is feature-independent. Thus, we could partition $\mathbf{X}$ into blocks of b features, e.g., $\{\mathbf{X}_1, \mathbf{X}_2, ...\} \in \mathbb{R}^{n \times b}$.

Customized batching. For operators with *both* inter-sample and inter-feature dependencies, the computation of each data point involves all samples and features, and therefore cannot be directly decomposed into batches along the same or feature dimension. TOD provides customized batching strategies for these operators. For example, to automatically batch `cdist`, TOD uses the approach introduced in Neeb & Kurrus [211], which splits an input dataset across samples and calculates the pairwise distance of *each pair of split* in batches.

7.6.1 Sequential Batching and Operator Fusion

Simple concatenation. Since BTOs are independent from each other, executing a sequence of BTOs in batches is straightforward, i.e., simply feed the output of a batch operator as an input to

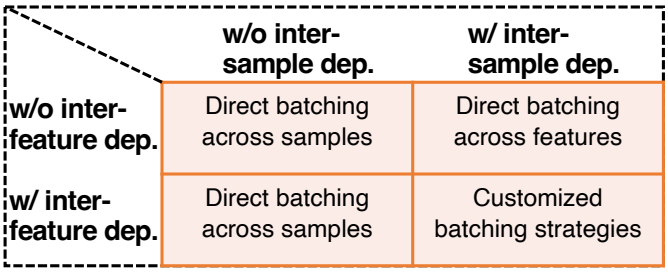


Figure 7.4: TOD applies automatic batching to operators without inter-sample or inter-feature dependency, and uses customized batching strategies for operators with both data dependencies.

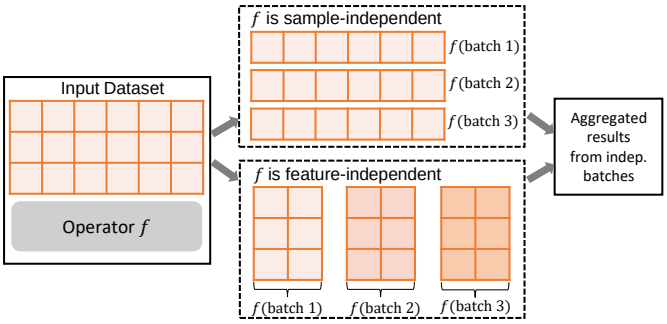


Figure 7.5: Direct batching with independence assumption creates batches along the sample or feature index.

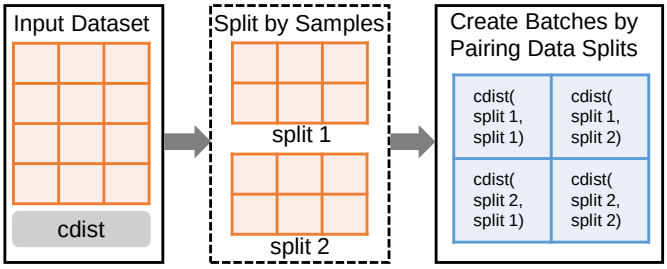


Figure 7.6: Customized batching solution for cdist in TOD.

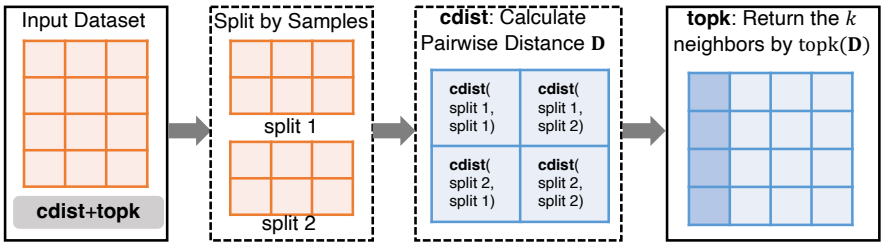
another one. For example, *k*NN (see Fig. 7.2) finds the k nearest neighbors by first calculating pairwise distances of input samples via the `cdist` BTO and then returns the index of k items with the smallest distance via the `topk` BTO. Thus, *k*NN batching is achieved by running `cdist` and `topk` sequentially, where each uses automatic batching and the output of the former is the input of the latter. Note that simple concatenation applies to all BTOs and FOs as the default choice.

Operator fusion. Although the simple concatenation discussed above is straightforward, a closer

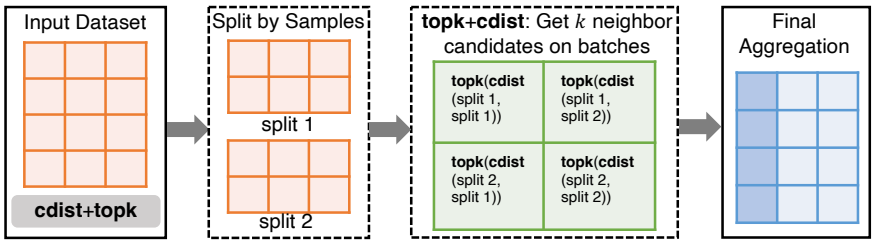
look unlocks deeper optimization opportunities in automatic batching with a sequence of operators. Notably, the output of *k*NN is the indices of the *k* nearest neighbors of an input dataset, where the pairwise distance generated by `cdist` is only used in an intermediate step but not returned. If we could prevent moving this large distance matrix between operators (i.e., `cdist` and `topk`), space efficiency can be improved. In deep learning systems, *operator fusion* is a common optimization technique to fuse multiple operators into a single one in a computational graph [213, 187, 299, 42, 201]. Fig. 7.7 compares simple concatenation (subfigure a) and operator fusion (subfigure b) on *k*NN. Specifically, the latter executes the `topk` BTO on the `cdist` BTO's individual batches separately rather than running `topk` on the full distance matrix outputted by `cdist`. Note that the global *k* nearest neighbors (of the full dataset) can be identified from the *k* local neighbor candidates from batches in the final aggregation, so the result is still exact. This prevents moving the entire $n \times n$ distance matrix between operators, which often causes OOM. TOD uses a *rule-based* approach to opportunistically fusing operators to reduce the kernel launch overhead and data transfers between CPUs and GPUs. TOD provides an interface that allows users to add fusion rules for new OD operators.

7.6.2 Multi-GPU Support

To further reduce the execution time of OD algorithms on GPUs, TOD also supports multi-GPU execution, which is important for time-critical OD applications and has been widely used for other data-intensive applications such as graph neural networks [127]. Intuitively, if there is only one GPU, TOD iterates across multiple batches sequentially and aggregates the results. When multiple GPUs are available, we could achieve better performance by executing OD computations concurrently on multiple homogeneous GPUs. Specifically, TOD first applies automatic batching to an underlying task—multiple subtasks are created and assigned to available GPUs. TOD creates a *subprocess* for each available GPU to execute the assigned subtasks and a *shared global container* to store



(a) Simple concatenation: `topk` is invoked on the full result of `cdist`—we need to communicate the large distance matrix $\mathbf{D}$.



(b) Operator fusion: `topk` is directly invoked on the batch result of `cdsit`, preventing the communication of distance matrix $\mathbf{D}$.

Figure 7.7: The comparison of automatic batching for kNN between simple concatenation and operator fusion. The latter has better scalability by not creating and moving large distance matrix $\mathbf{D}$.

the results returned from each GPU. Since we equally distribute subtasks across GPUs, we deem the runtime of each GPU is close. Once all the subtasks are complete, the final output is generated by aggregating the results in the global container. For example, automatically batching `cdist` in Fig. 7.6 leads to 4 subtasks, each of which calculates the pairwise distances for a pair of splits (denoted as blue blocks in the figure). Each of the four available GPUs executes an assigned subtask and sends the `cdist` results to the global container. Finally, the full `cdist` result is obtained by aggregating the intermediate results in the global container. Note that the multi-GPU execution is at the operator level (e.g., `cdist`). §7.7.7 evaluates TOD's scalability across multiple GPUs.

7.7 Experiments

Our experiments answer the following questions:

1. Is TOD more efficient (in time and space) than SOTA *CPU-based* OD system (i.e., PyOD)

and selected *GPU* baselines? (§7.7.3)

2. How scalable is TOD while handling more and more data? (§7.7.4)
3. How effective are provable quantization and automatic batching), in comparison to PyTorch implementation? (§7.7.5 & 7.7.6)
4. How much performance gain can TOD achieve on the multiple GPUs? (§7.7.7)

7.7.1 Implementation and Environment

TOD is implemented on top of PyTorch [233]. We extend PyTorch in the following aspects to support efficient OD. First, we implement a set of BTOs and FOs (see Fig. 7.2) for fast tensor operations in OD. Second, for operators that support provable quantization (see §7.5) and batching (see §7.6), we create corresponding versions of them to improve scalability. Additionally, we enable specialized multi-GPU support in TOD by leveraging PyTorch's `multiprocessing`.

Adding new operators . In addition to the BTOs and FOs listed in Fig. 7.2, users can add new operators in TOD by defining the operator's interface (i.e., the input and output tensors of the operator) and providing an implementation of the operator in PyTorch. This implementation will be used by TOD to decompose the operator into PyTorch's tensor algebra primitives and execute these primitives in parallel on multiple GPUs. For operators that do not have inter-sample *or* inter-feature dependency (see §7.6), TOD automatically decomposes the operator's computation into multiple batches. For operators that involve both inter-sample *and* inter-feature dependencies, TOD require users to provide a customized strategy to decompose the operator into batches.

Experimental setup. All the experiments were performed on an Amazon EC2 cluster with an Intel Xeon E5-2686 v4 CPU, 61GB DRAM, and an NVIDIA Tesla V100 GPU. For the multi-GPU support evaluation, we extend it to multiple NVIDIA Tesla V100 GPUs with the same CPU node.

7.7.2 Datasets, Baselines, and Evaluation Metrics

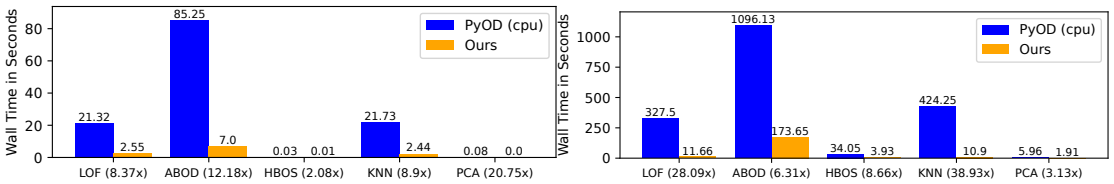
Datasets. Table 7.2 shows the 11 real-world benchmark datasets used in this study, which are widely evaluated in OD research [51, 346, 255, 174] and available in the latest ADBench[†] [105]. Given the limited size of real-world OD datasets, we also build data generation function in TOD to create larger synthetic datasets (up to 1.5 million samples) to evaluate the scalability of TOD (see §7.7.4 for details).

OD algorithms and operators. Throughout the experiments, we compare the performance of five representative but diverse OD algorithms across different systems (see §7.2.1): proximity-based algorithms including LOF [46], ABOD [145], and *k*NN [26]; statistical method HBOS [92], and linear model PCA [272]. We also provide an operator-level analysis on selected BTOs and FOs to demonstrate the effectiveness of certain techniques.

Evaluation metrics. Since TOD and the baselines do not involve any approximation, the output results are exact and consistent across systems. Therefore, we omit the accuracy evaluation, and compare the wall-clock time and GPU memory consumption as measures of time and space efficiency.

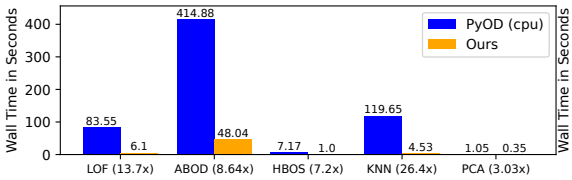
Baselines. As discussed in Section 7.2, there is no existing GPU system that covers a diverse group of OD algorithms (not even the above five algorithms) for a fair comparison. Therefore, we use the SOTA comprehensive system PyOD [347] as a *CPU* baseline in §7.7.3 and 7.7.4, which is deeply optimized with JIT compilation and parallelization. Regarding *GPU* baselines, we compare two representative OD algorithms (i.e., *k*NN-CUDA [25] and LOF-CUDA [21]) that have GPU support in §7.7.3, and direct implementation of operators in PyTorch in §7.7.5, 7.7.6, and 7.7.7. Note that the implementation of *k*NN-CUDA and LOF-CUDA are not open-sourced, so we follow the original papers to implement.

[†]Datasets available at ADBench: `https://github.com/Minqi824/ADBench`

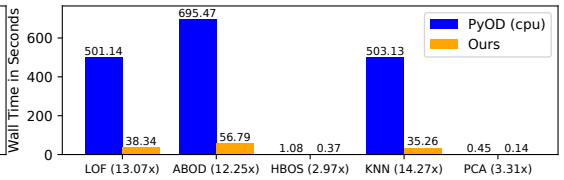


(a) ALOI (49,534 samples with 27 features)

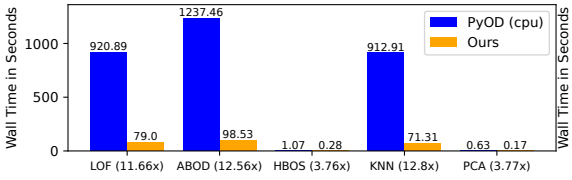
(b) cifar-10 (60,000 samples with 784 features)



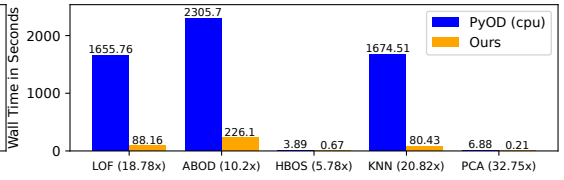
(c) fashion-mnist (60,000 samples with 3,072 features)



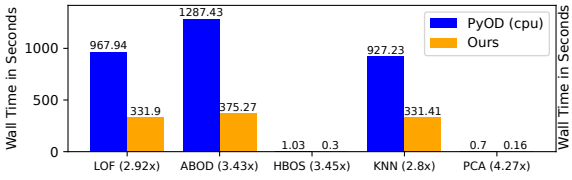
(d) celeba (202,599 samples with 39 features)



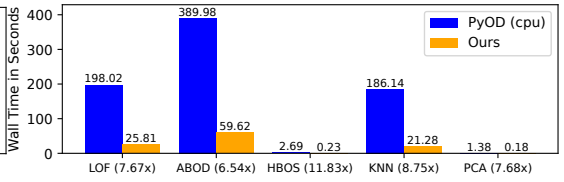
(e) fraud (284,807 samples with 29 features)



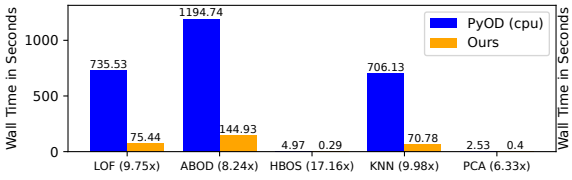
(f) census (299,285 samples with 500 features)



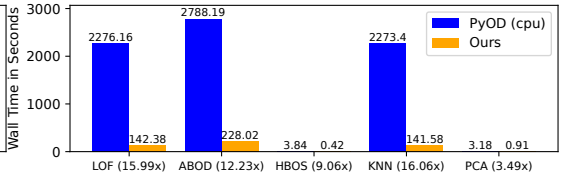
(g) donors (619,326 samples with 10 features)



(h) Synthetic dataset 1 (100,000 samples with 200 features)



(i) Synthetic dataset 2 (200,000 samples with 200 features)



(j) Synthetic dataset 3 (400,000 samples with 200 features)

Figure 7.8: Runtime comparison between PyOD and TOD in seconds on both real-world and synthetic datasets. TOD significantly outperforms PyOD in all w/ much smaller runtime, where the speedup factor is shown in parenthesis by each algorithm. On avg., TOD is 10.9× faster than PyOD (up to 38.9×).

Table 7.2: Real-world OD datasets used in the experiments. To demonstrate the results on larger datasets, we also create and use synthetic datasets throughout the experiments.

Dataset	Pts (n)	Dim (d)	% Outlier
musk	3,062	166	3.17
speech	3,686	400	1.65
mnist	7,603	100	9.21
mammography	11,183	6	2.32
ALOI	49,534	27	3.04
fashion-mnist	60,000	784	10
cifar-10	60,000	3072	10
celeba	202,599	39	2.24
fraud	284,807	29	0.17
census	299,285	500	6.20
donors	619,326	10	5.93

7.7.3 End-to-end Evaluation

TOD **is significantly faster than the leading CPU-based system**. We first present the runtime comparison between TOD and PyOD in Fig. 7.8 using seven real datasets (ALOI, fashion-mnist, and cifar-10) and three synthetic datasets (where Synthetic 1 contains 100,000 samples, Synthetic 2 contains 200,000 samples, and Synthetic 3 contains 400,000 samples (all are with 200 features). The results show that TOD is on average 10.9× faster than PyOD on the five benchmark algorithms (13.0×, 15.9×, 9.3×, 7.2×, and 8.9× speed-up on LOF, kNN, ABOD, HBOS, and PCA, respectively). For proximity-based algorithms, a larger speed-up is observed for datasets with a higher number of dimensions: LOF and kNN are 28.1× and 38.9× faster on cifar-10 with 3,072 features. This is expected as GPUs are well-suited for dense tensor multiplication, which is essential in proximity-based methods. Separately, a larger improvement can be achieved for HBOS and PCA on datasets with larger sample sizes. For instance, HBOS is 11.83× faster on Synthetic 1 (100,000 samples), while the speedup is 17.16× on Synthetic 2 (200,000 samples). This is expected as HBOS treats each feature independently for density estimation on GPUs, so a large number of samples with a

Table 7.3: Runtime comparison among selected GPU baselines (*k*NN-CUDA [25] and LOF-CUDA [21]; neither supports multi-GPU directly), and TOD (single GPU) and TOD-8 (8 GPUs). The first column shows three synthetic datasets with an increasing number of samples (100 dimensions), and the most efficient result is highlighted in bold for each setting. TOD-8 outperforms in all cases due to multi-GPU support, while TOD with a single GPU is faster or on par with the baselines. Note that the GPU baselines run out-of-memory (OOM) on large datasets (e.g., the last row), while TOD does not.

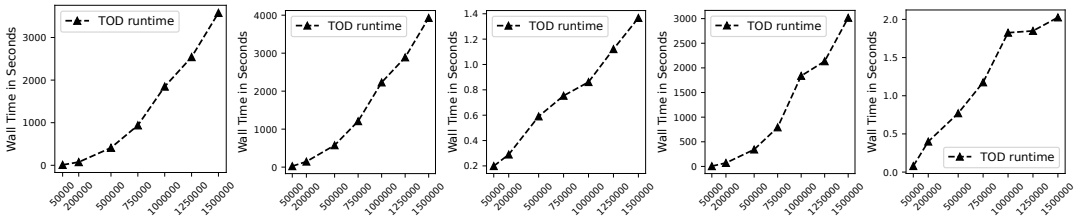
Dataset	***k*NN-CUDA**	TOD	TOD-8	**LOF-CUDA**	TOD	TOD-8
500,000	205.33	208.84	**28.59**	312.55	209	**28.64**
1,000,000	850.12	827	**112.35**	OOM	819	**113.72**
2,000,000	OOM	3173.39	**430.29**	OOM	3174.05	**434.18**

small number of features should yield a significant speed-up. In summary, all the OD algorithms tested are significantly faster in TOD than in the SOTA PyOD system, with the precise amount of speed improvement varying across algorithms.

TOD **can handle larger datasets than the GPU baselines**. Due to the absence of GPU systems that support all five OD algorithms, we specifically compare the performance of TOD to specialized GPU algorithms *k*NN-CUDA [25] and LOF-CUDA [21]. Table 7.3 shows that TOD with 8-GPUs outperforms in all three datasets due to the multi-GPU support, which enables it to handle data more efficient than the baselines and TOD with a single GPU. By focusing on the use of a single GPU, we find that TOD is faster or on par with both baselines due to provable quantization (e.g., 33.13% speedup to LOF-CUDA). Also note that the GPU baselines face the out-of-memory issue (OOM) on large datasets (e.g., the last row of the table with 2,000,000 samples), while TOD can still handle it due to automatic batching. In comparison to these specialized GPU baselines, TOD does not only provide more coverage of diverse algorithms, but also yields better efficiency and scalability.

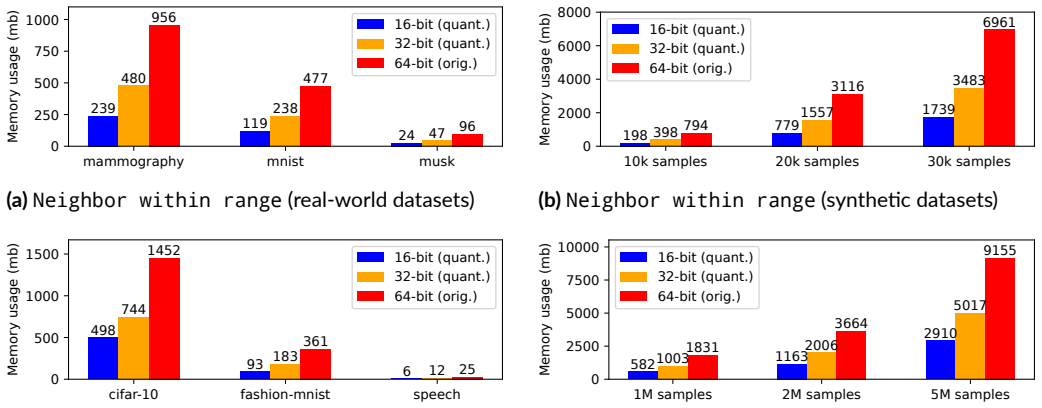
7.7.4 Scalability of TOD

We now gauge the scalability of TOD on datasets of varying sizes, including ones larger than fashion-mnist and cifar-10. In Fig. 7.9, we plot TOD's runtime with five OD algorithms on the synthetic



(a) LOF (quadratic) **(b)** ABOD (quadratic) **(c)** HBOS (linear) **(d)** kNN (quadratic) **(e)** PCA (linear)

Figure 7.9: Scalability plot of selected algorithms in TOD, where it scales well with an increasing number of samples.



(a) `Neighbor within range` (real-world datasets)

(b) `Neighbor within range` (synthetic datasets)

(c) `topk` (real-world datasets)

(d) `topk` (synthetic datasets)

Figure 7.10: GPU memory consumption comparison between using provable quantization (16-bit and 32-bit) and the full precision (64-bit). Clearly, provable quantization leads to significant memory consumption saving on `nwr` and `topk`.

datasets with sample sizes ranging from 50,000 to 1,500,000 (all with 200 features). To the best of our knowledge, none of the existing comprehensive OD systems can handle datasets with more than a million samples within a reasonable amount of time [7, 345], as most of the OD algorithms are associated with quadratic time complexity. Fig. 7.9 shows that TOD can process million-sample OD datasets within an hour, providing a scalable approach to deploying OD algorithms in many real-world tasks.

Table 7.4: Comparison of operator runtime (in seconds) with provable quantization (i.e., 16-bit and 32-bit) and without quantization (i.e., 64-bit) for `nwr` and `topk`. The best model is highlighted in bold (per column), where provable quantization in 16-bit outperforms the rest in most cases.

Prec.	mammog.	mnist	musk	10k	20k	30k
16-bit	**2.66**	**0.54**	**0.06**	**0.87**	**3.02**	**6.56**
32-bit	2.92	1.51	0.09	2.57	10.09	22.51
64-bit	3.31	1.53	0.09	3.49	12.7	27.32

(a) For `nwr`, 16-bit provable quantization outperforms in all

Prec.	cifar-10	f-mnist	speech	1M	2M	5M
16-bit	**0.31**	0.09	0.0054	**0.58**	**1.16**	**2.90**
32-bit	0.32	0.1	0.0048	0.70	1.40	3.52
64-bit	0.34	**0.07**	**0.0038**	0.71	1.73	3.88

(b) For `topk`, 16-bit provable quantization wins for large data

7.7.5 Provable Quantization

Provable quantization (§7.5) in TOD can optimize the operator memory usage while provably preserving correctness (i.e., no accuracy degradation). To demonstrate its effectiveness, we compare the GPU memory consumption of two applicable operators, `nwr` and `topk`, with and without provable quantization using the GPU baseline. Multiple real-world and synthetic datasets are used in the comparison (see Table 7.2), where synthetic datasets' names, such as "10k" and "1M", denote their sample sizes. We deem the 64-bit floating point as the ground truth, and evaluate the provable quantization results in 32- and 16-bit floating-point.

The results demonstrate that provable quantization always leads to memory savings. Specifically, Fig. 7.10 (a-b) shows `nwr` with provable quantization on average saves 71.27% (with 16-bit precision) and 47.59% (with 32-bit precision) of the full 64-bit precision GPU memory. Similarly, Fig. 7.10 (c-d) shows that `topk` with provable quantization saves 73.49% (with 16-bit precision) and 49.58% (with 32-bit precision) of the full precision memory. Regarding the runtime comparison, operating in lower precision may also lead to an edge. Table 7.4 shows the operator runtime comparison

between using provable quantization (in 16-bit and 32-bit precision) and using the full 64-bit precision. It shows that provable quantization in 16-bit precision is faster than the computations in full precision in most cases, especially for large datasets (e.g., the last three columns of Table 7.4). This empirical finding can be attributed to lower-precision operations typically being faster, and this speed improvement outweighs the overhead of post verification (see §7.5.4). For small datasets (the first three columns of Table 7.4), provable quantization does not necessarily improve the run time due to the additional verification and data movement, both of which finish in 0.1 seconds.

Case study on runtime breakdown . In addition to comparing the total runtime of an operator with or without provable quantization (§7.7.5), it is interesting to see the time breakdown of each phase of provable quantization. Specifically, the runtime of provable quantization can be divided into (*i*) operator evaluation in lower precision, (*ii*) result verification and (*iii*) recalculation in the original precision for the ones that fail in the verification. Taking `nwr` on 30k samples as an example (the last column of Table 7.4a), we show the time breakdown of (*i*) low-precision evaluation (*ii*) correctness verification and (*iii*) recalculation in higher precision in Table 7.5. In this case, the performance improvement of provable quantization comes from the reduced evaluation time in lower precision, which outweighs the cost of verification and recalculation. To further demonstrate the necessity of post-verification, we also measure the accuracy variation (e.g., ROC-AUC [7]) by simply running *k*NN detector on fraud, census, and donors datasets in 16-bit precision without post-verification, which leads to -3.48%, $+1.05\%$, -4.27% accuracy variation. These results show the merit of provable quantization over direct quantization.

7.7.6 Automatic Batching

To evaluate the effectiveness of automatic batching, we compare the runtime of multiple BOs and FOs under (*i*) an Numpy implementation on a CPU [107], (*ii*) a direct PyTorch GPU implementation without batching [233], and (*iii*) TOD's automatic batching.

Table 7.5: Runtime breakdown of using provable quantization on nwr operator with a 30,000 sample synthetic dataset. Column 1 shows the runtime for low-precision evaluation, where column 2 and 3 show the runtime for correctness verification and recalculation, respectively. It shows the primary speed-up comes from low-precision evaluation, while the overhead of verification and recalculation is marginal.

Prec.	Low Prec.	Verification	Recalculation	Total
16-bit	5.91	0.05	0.61	6.56
32-bit	22	0.05	0.47	22.51
64-bit	N/A	N/A	N/A	27.32

Table 7.6: Operator runtime comparison among implementations in NumPy (no batching), PyTorch (no batching) and TOD (with automatic batching); the most efficient result is highlighted in bold per row. Automatic batching in TOD prevents out-of-memory (OOM) errors yet shows great efficiency, especially on large datasets.

Operator	Size	NumPy	PyTorch	TOD
`topk`	10,000,000	7.88	**1.08**	1.09
`topk`	20,000,000	15.77	OOM	**2.44**
`topk`	100,000,000	OOM	OOM	**10.83**
`intersect`	20,000,000	1.99	**0.12**	0.14
`intersect`	100,000,000	11	**0.63**	**0.63**
`intersect`	200,000,000	21.65	OOM	**2.11**
*k*NN	50,000	20.15	**0.27**	0.28
*k*NN	200,000	194.43	OOM	**19.65**
*k*NN	400,000	818.32	OOM	**71.22**

Table 7.6 compares the three implementations of key operators in OD systems. Clearly, TOD with automatic batching achieves the best balance of efficiency and scalability, leading to 7.22×, 17.46×, and 11.49× speedups compared to a highly optimized NumPy implementation on CPUs. TOD can also handle more than 10× larger datasets where the direct PyTorch implementation faces out of memory (OOM) errors. TOD is only marginally slower than PyTorch when the input dataset is small (see the first row of each operator). In this case, batching is not needed, and TOD is equivalent to PyTorch; TOD is slightly slower due to the overhead of TOD deciding whether or not to enable automatic batching.

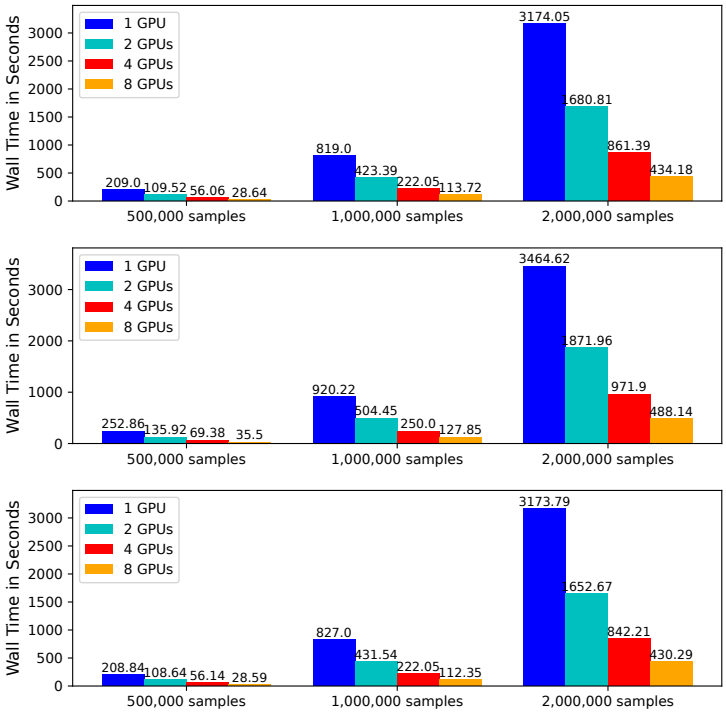


Figure 7.11: Runtime comparison of using different numbers of GPUs (**top**: LOF; **middle**: ABOD; **bottom**: kNN). TOD can efficiently leverage multiple GPUs for faster OD.

7.7.7 Multi-GPU Results

We now evaluate the scalability of TOD on multiple GPUs on a single compute node. Specifically, we compare the run time of three compute-intensive OD algorithms (i.e., LOF, ABOD, and kNN) with 1,2,4, and 8 NVIDIA Tesla V100 GPUs.

Fig. 7.11 shows that for three OD algorithms tested, TOD can achieve nearly linear speed-up with more GPUs—the GPU efficiency is mostly above 90%. For instance, the kNN result shows that using 2, 4, and 8 GPUs are 1.91×, 3.73×, and 7.34× faster than the single-GPU performance. As a comparison, using 2, 4, and 8 GPUs with ABOD and LOF are 1.85×, 3.63×, 7.14× faster and 1.91×, 3.70×, 7.27× faster, respectively. First, there is inevitably a small overhead in multi-processing, causing the speed-up to not exactly be linear. Second, the minor efficiency difference between kNN and ABOD is due to most OD operations in the former being executed on multiple

GPUs, while the latter involves several sequential steps that have to be run on CPUs. To sum up, TOD can leverage multiple GPUs efficiently to process large datasets.

7.8 Limitations and Future Directions

Tree-based OD algorithms. One limitation of TOD is that it does not support tree-based OD algorithms such as isolation forests [180]. Tree-based operations involve random data access [165], which is not friendly for GPUs designed for batch operations. Future work can consider converting trees to tensor operations [209] for acceleration.

Approximate solutions. Our focus in this chapter has been on *exact* efficient, scalable implementations of OD algorithms. In particular, we have not considered implementations that intentionally are meant to be approximate, where for instance, one could trade off between accuracy, computation time, and memory usage. Note that even with our provable quantization technique, we ask for the lower-precision computation to yield the correct (exact) output. A future research direction is to extend TOD to support approximate solutions for even better scalability and efficiency when reduced accuracy is acceptable. For instance, exact nearest neighbor search in can be switched to approximate nearest neighbor search [87, 300].

Heterogeneous GPUs. Thus far, we have not studied the use of TOD with heterogeneous GPUs. In this setting, incorporating a cost model could be helpful in balancing the workload between the different GPUs, accounting for their different characteristics such as varying memory capacities.

Gradient-based operators. Currently, TOD does not support operators that involve solving an optimization problem via gradient descent. Future work may consider incorporating optimization-based operators to support (a small group of) optimization-based OD algorithms, e.g., OCSVM [258].

Extension to classification. It is possible to use TOD to build classification models, as the oper-

ators in TOD are generic and may serve the tasks beyond OD. Note that classification tasks often involve optimization (e.g., via gradient descent), which we already pointed out that TOD does not yet support. Thus, only calculation-based classifiers can be implemented by TOD for now.

7.9 Discussions

In this chapter, we propose the first comprehensive GPU-based outlier detection system called TOD, which is on average 10.9 times faster than the leading system PyOD and is capable of handling larger datasets than existing GPU baselines. The key idea is to decompose complex outlier detection algorithms into a combination of tensor operations for effective GPU acceleration. Our system enables many large-scale real-world outlier detection applications that could have stringent time constraints. With the ease of extensibility, TOD can prototype and implement new detection algorithms.

Part III

Benchmarks and Applications

8

ADBench: Systematic Evaluation of Tabular Detection Algorithms

This chapter is primarily based on:

Songqiao Han*, Xiyang Hu*, Hailiang Huang*, Minqi Jiang*, and **Yue Zhao***†. "ADBench: Anomaly Detection Benchmark." *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. (*Co-first Author; †Corresponding Author)

8.1 Highlight

Although there are already some benchmark and evaluation works for tabular AD [51, 72, 76, 93, 281], they generally have the limitations as follows: (*i*) primary emphasis on unsupervised methods only without including emerging (semi-)supervised AD methods; (*ii*) limited analysis of the algorithm performance concerning anomaly types (e.g., local vs. global); *(iii)* the lack of analysis on model robustness (e.g., noisy labels and irrelevant features); *(iv)* the absence of using statistical tests for algorithm comparison; and *(v)* no coverage of more complex CV and NLP datasets, which have attracted extensive attention nowadays.

To address these limitations, we design (to our best knowledge) the most comprehensive tabular anomaly detection benchmark called ADBench. By analyzing both research needs and deployment requirements in the industry, we design the experiments with three major angles in anomaly detection (see §8.3.3): (*i*) the availability of supervision (e.g., ground truth labels) by including 14 unsupervised, 7 semi-supervised, and 9 supervised methods; (*ii*) algorithm performance under different types of anomalies by simulating the environments with four types of anomalies; and (*iii*) algorithm robustness and stability under three settings of data corruptions. Fig. 8.1 provides an overview of ADBench.

Key takeaways: Through extensive experiments, we find (*i*) surprisingly none of the benchmarked unsupervised algorithms is statistically better than others, emphasizing the importance of algorithm selection; (*ii*) with merely 1% labeled anomalies, most semi-supervised methods can outperform the best unsupervised method, justifying the importance of supervision; (*iii*) in controlled environments, we observe that the best unsupervised methods for specific types of anomalies are even better than semi- and fully-supervised methods, revealing the necessity of understanding data characteristics; (*iv*) semi-supervised methods show potential in achieving robustness in noisy and corrupted data, possibly due to their efficiency in using labels and feature selection. See §8.4 for

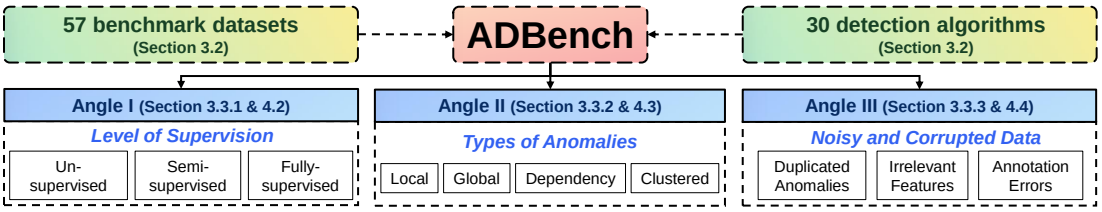


Figure 8.1: The design of the proposed ADBench is driven by research and application needs.

additional results and insights.

We summarize the primary contributions of ADBench as below:

1. **The most comprehensive AD benchmark**. ADBench examines 30 detection algorithms' performance on 57 benchmark datasets (of which 47 are existing ones and we create 10).

2. **Research and application-driven benchmark angles**. By analyzing the needs of research and real-world applications, we focus on three critical comparison angles: availability of supervision, anomaly types, and algorithm robustness under noise and data corruption.

3. **Insights and future directions for researchers and practitioners**. With extensive results, we show the necessity of algorithm selection, and the value of supervision and prior knowledge.

8.2 Comparisons Among Un-, Semi-, and Fully-supervised Detection Methods

8.2.1 Anomaly Detection Algorithms

We organize all the algorithms in ADBench into un-, semi-, and fully-supervised methods categories as outlined in §0. See [105] for hyperparameter settings.

Table 8.1: Comparison among ADBench and existing benchmarks, where ADBench comprehensively includes the most datasets and algorithms, uses both benchmark and synthetic datasets, covers both shallow and deep learning (DL) algorithms, and considers multiple comparison angles.

Benchmark	Coverage (§8.3.2)		Data Source		Algorithm Type		Comparison Angle (§8.3.3)		
	# datasets	# algo.	Real-world	Synthetic	Shallow	DL	Supervision	Types	Robustness
Ruff et al. [254]	3	9	✓	✓	✓	✓	✗	✓	✗
Goldstein et al. [93]	10	19	✓	✗	✓	✗	✗	✓	✗
Domingues et al. [72]	15	14	✓	✗	✓	✗	✗	✗	✓
Soenen et al. [277]	16	6	✓	✗	✓	✗	✗	✗	✗
Steinbuss et al. [281]	19	4	✗	✓	✓	✗	✗	✓	✗
Emmott et al. [76]	19	8	✓	✓	✓	✗	✗	✓	✓
Campos et al. [51]	23	12	✓	✗	✓	✗	✗	✗	✗
ADBench (ours)	57	30	✓	✓	✓	✓	✓	✓	✓

8.2.2 Existing Datasets and Benchmarks for Tabular AD

AD Datasets in Literature. Existing benchmarks mainly evaluate a part of the datasets derived from the ODDS Library [245], DAMI Repository [51], ADRepository [228], and Anomaly Detection Meta-Analysis Benchmarks [76]. In ADBench, we include almost all publicly available datasets, and add larger datasets adapted from CV and NLP domains, for a more holistic view. See details in §8.3.2.

Existing Benchmarks. There are some notable works that take effort to benchmark AD methods on tabular data, e.g., [51, 72, 76, 254, 281]. How does ADBench differ from them?

First, previous studies mainly focus on benchmarking the shallow unsupervised AD methods. Considering the rapid advancement of ensemble learning and deep learning methods, we argue that a comprehensive benchmark should also consider them. Second, most existing works only evaluate public benchmark datasets and/or some fully synthetic datasets; we organically incorporate both of them to unlock deeper insights. More importantly, existing benchmarks primarily focus on direct performance comparisons, while the settings may not be sufficiently complex to understand AD algorithm characteristics. We strive to address the above issues in ADBench, and illustrate the main differences between the proposed ADBench and existing AD benchmarks in Table 8.1.

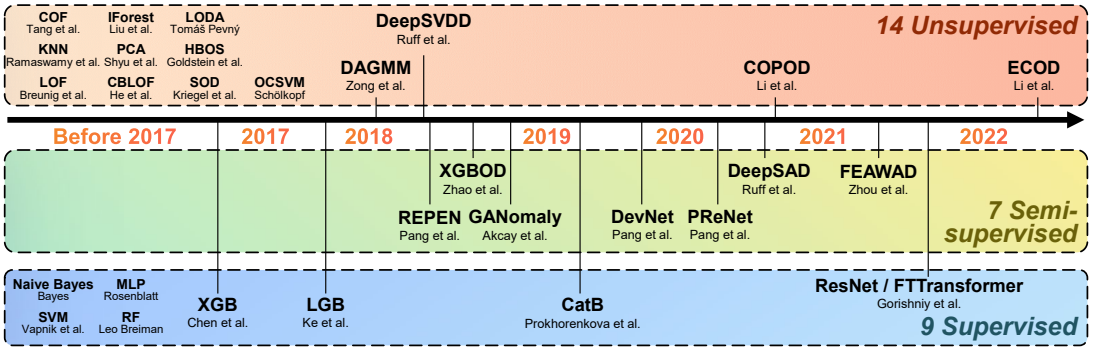


Figure 8.2: ADBench covers a wide range of AD algorithms. See §8.2.1 for more details.

Also, "anomaly detection" is an overloaded term; there are AD benchmarks for time-series [150, 154, 232], graph [182], CV [13, 53, 352] and NLP [242], but they are different from tabular AD in nature.

8.3 ADBench: AD Benchmark Driven by Research and Application Needs

8.3.1 Preliminaries and Problem Definition

We follow the definitions in §0 to categorize the algorithms included in ADBench. Irrespective of the types of underlying AD algorithms, the goal of ADBench is to understand AD algorithms' performance under the inductive setting. Collectively, we refer semi-supervised and supervised AD methods as "label-informed" methods. Refer to §8.4.1 for specific experiment settings.

8.3.2 The Largest AD Benchmark with 30 Algorithms and 57 Datasets

Algorithms. Compared to the previous benchmarks, we have a larger algorithm collection with (*i*) the latest unsupervised AD algorithms like DeepSVDD [253] and ECOD [175]; (*ii*) SOTA semi-supervised algorithms, including DeepSAD [255] and DevNet [230]; (*iii*) latest network architectures like ResNet [109] in computer vision (CV) and Transformer [296] in the natural language

processing (NLP) domain—we adapt ResNet and FTTransformer models [97] for tabular AD in the proposed ADBench; and (*iv*) ensemble learning methods like LightGBM [135], XGBoost [56], and CatBoost [237] that have shown effectiveness in AD tasks [294]. Fig. 8.2 shows the 30 algorithms (14 unsupervised, 7 semi-supervised, and 9 supervised algorithms) evaluated in ADBench, where we provide more information about them in §8.2.1.

Algorithm Implementation. Most unsupervised algorithms are readily available in our early work Python Outlier Detection (PyOD) [347], and some supervised methods are available in scikit-learn [235] and corresponding libraries. Supervised ResNet and FTTransformer tailored for tabular data have been open-sourced in their original paper [97]. We implement the semi-supervised methods and release them along with ADBench.

Public AD Datasets.

In ADBench, we gather more than 40 benchmark datasets [51, 76, 228, 245], for model evaluation, as shown in Table 8.2. These datasets cover many application domains, including healthcare (e.g., disease diagnosis), audio and language processing (e.g., speech recognition), image processing (e.g., object identification), finance (e.g., financial fraud detection), etc. For due diligence, we keep the datasets where the anomaly ratio is below 40%.

Newly-added Datasets in ADBench. Since most of these datasets are relatively small, we introduce 10 more complex datasets from CV and NLP domains with more samples and richer features in ADBench (highlighted in Table 8.2). Pretrained models are applied to extract data embedding from CV and NLP datasets to access more complex representations, which has been widely used in AD literature [64, 194, 255] and shown better results than using the raw features. For NLP datasets, we use BERT [67] pretrained on the BookCorpus and English Wikipedia to extract the embedding of the [CLS] token. For CV datasets, we use ResNet18 [109] pretrained on the ImageNet [66] to extract the embedding after the last average pooling layer. Following previous works [253, 255], we set one of the multi-classes as normal, downsample the remaining classes to 5% of the total instances

Table 8.2: Data description of the 57 datasets included in ADBench; 10 newly added datasets from CV and NLP domain are highlighted in blue at the bottom of the table.

Data	# Samples	# Features	# Anomaly	% Anomaly	Category	Reference
ALOI	49534	27	1508	3.04	Image	[76]
annthyroid	7200	6	534	7.42	Healthcare	[240]
backdoor	95329	196	2329	2.44	Network	[207]
breastw	683	9	239	34.99	Healthcare	[317]
campaign	41188	62	4640	11.27	Finance	[230]
cardio	1831	21	176	9.61	Healthcare	[27]
Cardiotocography	2114	21	466	22.04	Healthcare	[27]
celeba	202599	39	4547	2.24	Image	[230]
census	299285	500	18568	6.20	Sociology	[230]
cover	286048	10	2747	0.96	Botany	[40]
donors	619326	10	36710	5.93	Sociology	[230]
fault	1941	27	673	34.67	Physical	[76]
fraud	284807	29	492	0.17	Finance	[230]
glass	214	7	9	4.21	Forensic	[77]
Hepatitis	80	19	13	16.25	Healthcare	[68]
http	567498	3	2211	0.39	Web	[245]
InternetAds	1966	1555	368	18.72	Image	[51]
Ionosphere	351	33	126	35.90	Oryctognosy	[273]
landsat	6435	36	1333	20.71	Astronautics	[76]
letter	1600	32	100	6.25	Image	[85]
Lymphography	148	18	6	4.05	Healthcare	[52]
magic.gamma	19020	10	6688	35.16	Physical	[76]
mammography	11183	6	260	2.32	Healthcare	[319]
mnist	7603	100	700	9.21	Image	[159]
musk	3062	166	97	3.17	Chemistry	[69]
optdigits	5216	64	150	2.88	Image	[20]
PageBlocks	5393	10	510	9.46	Document	[192]
pendigits	6870	16	156	2.27	Image	[18]
Pima	768	8	268	34.90	Healthcare	[245]
satellite	6435	36	2036	31.64	Astronautics	[245]
satimage-2	5803	36	71	1.22	Astronautics	[245]
shuttle	49097	9	3511	7.15	Astronautics	[245]
skin	245057	3	50859	20.75	Image	[76]
smtp	95156	3	30	0.03	Web	[245]
SpamBase	4207	57	1679	39.91	Document	[51]
speech	3686	400	61	1.65	Linguistics	[47]
Stamps	340	9	31	9.12	Document	[51]
thyroid	3772	6	93	2.47	Healthcare	[241]
vertebral	240	6	30	12.50	Biology	[36]
vowels	1456	12	50	3.43	Linguistics	[147]
Waveform	3443	21	100	2.90	Physics	[186]
WBC	223	9	10	4.48	Healthcare	[193]
WDBC	367	30	10	2.72	Healthcare	[193]
Wilt	4819	5	257	5.33	Botany	[51]
wine	129	13	10	7.75	Chemistry	[6]
WPBC	198	33	47	23.74	Healthcare	[193]
yeast	1484	8	507	34.16	Biology	[114]
CIFAR10	5263	512	263	5.00	Image	[146]
FashionMNIST	6315	512	315	5.00	Image	[321]
MNIST-C	10000	512	500	5.00	Image	[208]
MVTec-AD		See [105].			Image	[33]
SVHN	5208	512	260	5.00	Image	[212]
Agnews	10000	768	500	5.00	NLP	[339]
Amazon	10000	768	500	5.00	NLP	[110]
Imdb	10000	768	500	5.00	NLP	[191]
Yelp	10000	768	500	5.00	NLP	[339]
20newsgroups		See [105].			NLP	[153]

as anomalies, and report the average results over all the respective classes. Including these originally non-tabular datasets helps to see whether tabular AD methods can work on CV/NLP data after necessary preprocessing. See [105] for more details on datasets.

8.3.3 Benchmark Angles in ADBench

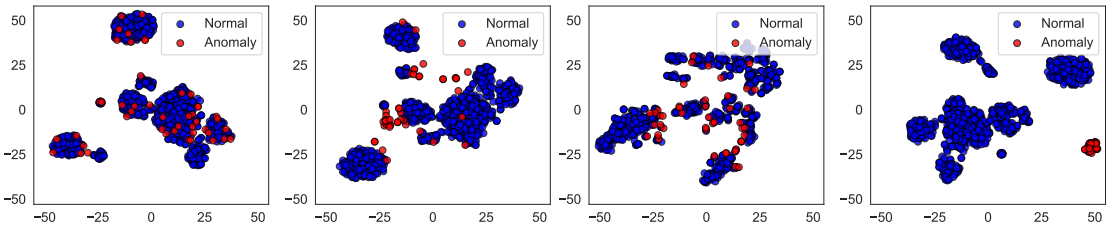
8.3.3.1 Angle I: Availability of Ground Truth Labels (Supervision)

Motivation. As shown in Table 8.1, existing benchmarks only focus on the unsupervised setting, i.e., none of the labeled anomalies is available. Despite, in addition to unlabeled samples, one may have access to a limited number of labeled anomalies in real-world applications, e.g., a few anomalies identified by domain experts or human-in-the-loop techniques like active learning [12, 14, 138, 336]. Notably, there is a group of semi-supervised AD algorithms [226, 227, 229, 230, 255, 287, 357] that have not been covered by existing benchmarks.

Our design: We first benchmark existing unsupervised anomaly detection methods, and then evaluate both semi-supervised and fully-supervised methods with varying levels of supervision following the settings in [226, 230, 357] to provide a fair comparison. For example, labeled anomalies $\gamma_l = 10\%$ means that 10% anomalies in the train set are known while other samples remain unlabeled. The complete experiment results of un-, semi-, and full-supervised algorithms are presented in §8.4.2.

8.3.3.2 Angle II: Types of Anomalies

Motivation. While extensive public datasets can be used for benchmarking, they often consist of a mixture of different types of anomalies, making it challenging to understand the pros and cons of AD algorithms regarding specific types of anomalies [96, 281]. In real-world applications, one may know specific types of anomalies of interest. To better understand the impact of anomaly types, we



(a) Local **(b)** Global **(c)** Dependency **(d)** Clustered

Figure 8.3: Illustration of four types of synthetic anomalies shown on Lymphography dataset.

create synthetic datasets based on public datasets by injecting specific types of anomalies to analyze the response of AD algorithms.

Our design: In ADBench, we create *realistic* synthetic datasets from benchmark datasets by injecting specific types of anomalies. Some existing works, such as PyOD [347], generate fully synthetic anomalies by assuming their data distribution, which fails to create complex anomalies. We follow and enrich the approach in [281] to generate "realistic" synthetic data; ours supports more types of anomaly generation. The core idea is to build a generative model (e.g., Gaussian mixture model GMM used in [281], Sparx [338], and ADBench) using the normal samples from a benchmark dataset and discard its original anomalies as we do not know their types. Then, We could generate normal samples and different types of anomalies based on their definitions by tweaking the generative model. The generation of normal samples is the same in all settings if not noted, and we provide the generation process of four types of anomalies below (also see our codebase for details).

Definition and Generation Process of Four Types of Common Anomalies Used in ADBench:

- **Local anomalies** refer to the anomalies that are deviant from their local neighborhoods [46]. We follow the GMM procedure [202, 281] to generate synthetic normal samples, and then scale the covariance matrix $\hat{\Sigma} = \alpha\hat{\Sigma}$ by a scaling parameter $\alpha = 5$ to generate local anomalies.

- **Global anomalies** are more different from the normal data [117], generated from a uniform

distribution Unif $\left(\alpha \cdot \min\left(\mathbf{X^k}\right), \alpha \cdot \max\left(\mathbf{X^k}\right)\right)$, where the boundaries are defined as the *min* and *max* of an input feature, e.g., k-th feature $\mathbf{X^k}$, and $\alpha = 1.1$ controls the outlyingness of anomalies.

- **Dependency anomalies** refer to the samples that do not follow the dependency structure which normal data follow [198], i.e., the input features of dependency anomalies are assumed to be independent of each other. Vine Copula [1] method is applied to model the dependency structure of original data, where the probability density function of generated anomalies is set to complete independence by removing the modeled dependency (see [198]). We use Kernel Density Estimation (KDE) [108] to estimate the probability density function of features and generate normal samples.

- **Clustered anomalies**, also known as group anomalies [161], exhibit similar characteristics [76, 179]. We scale the mean feature vector of normal samples by $\alpha = 5$, i.e., $\hat{\mu} = \alpha\hat{\mu}$, where α controls the distance between anomaly clusters and the normal, and use the scaled GMM to generate anomalies.

Fig. 8.3 shows 2-d t-SNE [291] visualization of the four types of synthetic outliers generated from Lymphography dataset, where they generally satisfy the expected characteristics. Local anomalies (Fig. 8.3a) are well overlapped with the normal samples. Global anomalies (Fig. 8.3b) are more deviated from the normal samples and on the edges of normal clusters. The other two types of anomalies are as expected, with no clear dependency structure in Fig. 8.3c and having anomaly cluster(s) in Fig. 8.3d. In ADBench, we analyze the algorithm performances under all four types of anomalies above (§8.4.3).

8.3.3.3 Angle III: Model Robustness with Noisy and Corrupted Data

Motivation. Model robustness has been an important aspect of anomaly detection and adversarial machine learning [50, 74, 79, 137, 320]. Meanwhile, the input data likely suffers from noise and corruption to some extent in real-world applications [76, 96, 101, 223]. However, this important view has not been well studied in existing benchmarks, and we try to understand this by evaluating AD algorithms under three noisy and corruption settings (see results in §8.4.4):

- **Duplicated Anomalies**. In many applications, certain anomalies likely repeat multiple times in the data for reasons such as recording errors [149]. The presence of duplicated anomalies is also called the "anomaly masking" [96, 101, 180], posing challenges to many AD algorithms [51], e.g., the density-based KNN [26, 243]. Besides, the change of anomaly frequency would also affect the behavior of detection methods [76]. Therefore, we simulate this setting by splitting the data into train and test set, then duplicating the anomalies (both features and labels) up to 6 times in both sets, and observing how AD algorithms change.

- **Irrelevant Features**. Tabular data may contain irrelevant features caused by measurement noise or inconsistent measuring units [54, 96], where these noisy dimensions could hide the characteristics of anomaly data and thus make the detection process more difficult [227, 254]. We add irrelevant features up to 50% of the total input features (i.e., d in the problem definition) by generating uniform noise features from Unif $\left(\min\left(\mathbf{X^k}\right), \max\left(\mathbf{X^k}\right)\right)$ of randomly selected k-th input feature $\mathbf{X^k}$ while the labels stay correct, and summarize the algorithm performance changes.

- **Annotation Errors**. While existing studies [230, 255] explored anomaly contamination in the unlabeled samples, we further discuss the more generalized impact of label contamination on the algorithm performance, where the label flips [214, 351] between the normal

samples and anomalies are considered (up to 50% of total labels). Note this setting does not affect unsupervised methods as they do not use any labels. Discussion of annotation errors is meaningful since manual annotation or some automatic labeling techniques are always noisy while being treated as perfect.

8.4 Experiments

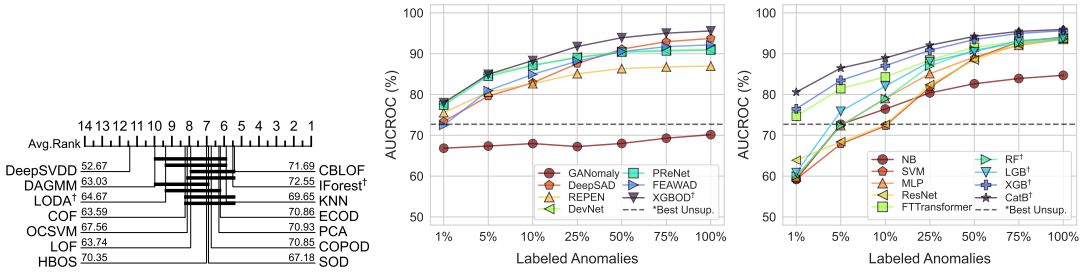
We conduct 98,436 experiments to answer **Q1** (§8.4.2): How do AD algorithms perform with varying levels of supervision? **Q2** (§8.4.3): How do AD algorithms respond to different types of anomalies? **Q3** (§8.4.4): How robust are AD algorithms with noisy and corrupted data? In each subsection, we first present the key results and analyses (please refer to the additional points in [105]), and then propose a few open questions and future research directions.

8.4.1 ADBench:Experiment Setting

Datasets, Train/test Data Split, and Independent Trials. As described in §8.3.2 and Table 8.2, ADBench includes 57 existing and freshly proposed datasets, which cover different fields including healthcare, security, and more. Although unsupervised AD algorithms are primarily designed for the transductive setting (i.e., outputting the anomaly scores on the input data only other than making predictions on the newcoming data), we adapt all the algorithms for the inductive setting to predict the newcoming data, which is helpful in applications and also common in popular AD library PyOD [347], TODS [151], and PyGOD [181]. Thus, we use 70% data for training and the remaining 30% as the test set. We use stratified sampling to keep the anomaly ratio consistent. We repeat each experiment 3 times and report the average. Detailed settings are described in [105].

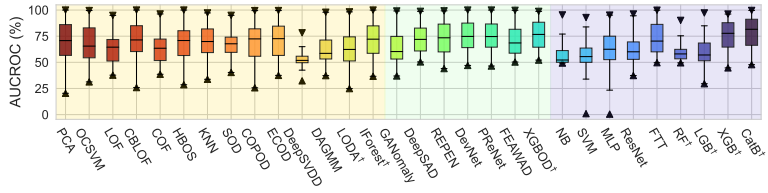
Hyperparameter Settings. For all the algorithms in ADBench, we use their default hyperparameter (HP) settings in the original paper for a fair comparison. Refer to [105] for more information.

Evaluation Metrics and Statistical Tests. We evaluate different AD methods by two widely used metrics: AUCROC (Area Under Receiver Operating Characteristic Curve) and AUCPR (Area Under Precision-Recall Curve) value*. Besides, the critical difference diagram (CD diagram) [65, 121] based on the Wilcoxon-Holm method is used for comparing groups of AD methods statistically ($p \leq 0.05$).

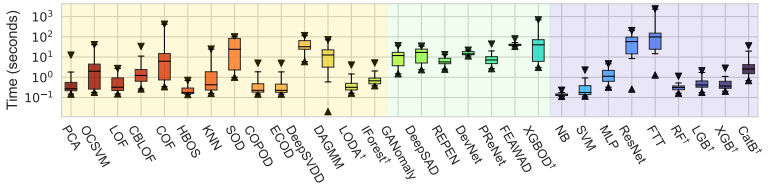


(a) Avg. rank (lower the better) and avg. AUCROC (on each line) of unsupervised methods; groups of algorithms not statistically different are connected horizontally.

(b) Avg. AUCROC (on 57 datasets) vs. % of labeled anomalies (x-axis); semi-supervised (left) and fully-supervised (right). Most label-informed algorithms outperform the best *unsupervised* algorithm CBLOF (denoted as the dashed line) with 10% labeled anomalies.



(c) Boxplot of AUCROC (@1% labeled anomalies) on 57 datasets; we denote un-, semi-, and fully supervised methods in light yellow, green, and purple.



(d) Boxplot of train time (see inf. time in [105]) on 57 datasets; we denote un-, semi-, and fully supervised methods in light yellow, green, and purple.

Figure 8.4: Average AD model performance across 57 benchmark datasets. (a) shows that no unsupervised algorithm statistically outperforms the rest. (b) shows that semi-supervised methods leverage the labels more efficiently than fully-supervised methods with a small labeled anomaly ratio. (c) and (d) present the boxplots of AUCROC and runtime. Ensemble methods are marked with "†".

*We present the results based on AUCROC and observe similar results for AUCPR; See [105] for all.

8.4.2 Overall Model Performance on Datasets with Varying Degrees of Supervision

As introduced in §8.3.3.1, we first present the results of unsupervised methods on 57 datasets in Fig. 8.4a, and then compare label-informed semi- and fully-supervised methods under varying degrees of supervision, i.e., different label ratios of γ_l (from 1% to 100% full labeled anomalies) in Fig. 8.4b.

None of the unsupervised methods is statistically better than the others, as shown in the critical difference diagram of Fig. 8.4a (where most algorithms are horizontally connected without statistical significance). We also note that some DL-based unsupervised methods like DeepSVDD and DAGMM are surprisingly worse than shallow methods. Without the guidance of label information, DL-based unsupervised algorithms are harder to train (due to more hyperparameters) and more difficult to tune hyperparameters, leading to unsatisfactory performance.

Semi-supervised methods outperform supervised methods when limited label information is available. For $\gamma_l \leq 5\%$, i.e., only less than 5% labeled anomalies are available during training, the detection performance of semi-supervised methods (median AUCROC= 75.56% for $\gamma_l = 1\%$ and AUCROC= 80.95% for $\gamma_l = 5\%$) are generally better than that of fully-supervised algorithms (median AUCROC= 60.84% for $\gamma_l = 1\%$ and AUCROC= 72.69% for $\gamma_l = 5\%$). For most semi-supervised methods, merely 1% labeled anomalies are sufficient to surpass the best unsupervised method (shown as the dashed line in Fig. 8.4b), while most supervised methods need 10% labeled anomalies to achieve so. We also show the improvement of algorithm performances about the increasing γ_l, and notice that with a large number of labeled anomalies, both semi-supervised and supervised methods have comparable performance. Putting these together, we verify the assumed advantage of semi-supervised methods in leveraging limited label information more efficiently.

Latest network architectures like Transformer and emerging ensemble methods yield

competitive performance in AD. Fig. 8.4b shows FTTransformer and ensemble methods like XGB(oost) and CatB(oost) provide satisfying detection performance among all the label-informed algorithms, even these methods are not specifically proposed for the anomaly detection tasks. For $\gamma_l = 1\%$, the AUCROC of FTTransformer and the median AUCROC of ensemble methods are 74.68% and 76.47%, respectively, outperforming the median AUCROC of all label-informed methods 72.91%. The great performance of tree-based ensembles (in tabular AD) is consistent with the findings in literature [43, 98, 294], which may be credited to their capacity to handle imbalanced AD datasets via aggregation. Future research may focus on understanding the cause and other merits of ensemble trees in tabular AD, e.g., better model efficiency.

Runtime Analysis. We present the train time in Fig. 8.4d. Runtime analysis finds that HBOS, COPOD, ECOD, and NB are the fastest as they treat each feature independently. In contrast, more complex representation learning methods like XGBOD, ResNet, and FTTansformer are computationally heavy. This should be factored in for algorithm selection.

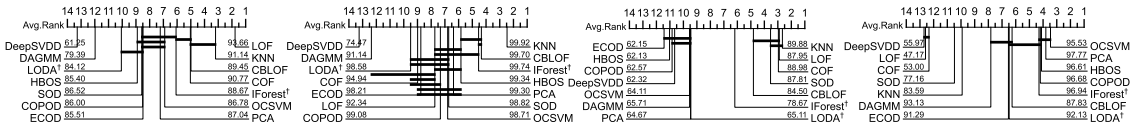
Future Direction 1: Unsupervised Algorithm Evaluation, Selection, and Design. For unsupervised AD, the results suggest that future algorithms should be evaluated on large testbeds like ADBench for statistical tests (such as via critical difference diagrams). Meanwhile, the no-free-lunch theorem [318] suggests there is no universal winner for all tasks, and more focus should be spent on understanding the suitability of each AD algorithm. Notably, algorithm selection and hyperparameter optimization are important in unsupervised AD, but limited works [30, 190, 348, 349, 342] have studied them. We may consider self-supervision [239, 263, 269, 322] and transfer learning [64] to improve tabular AD as well. Thus, we call for attention to large-scale evaluation, task-driven algorithm selection, and data augmentation/transfer for unsupervised AD.

Future Direction 2: Semi-supervised Learning. By observing the success of using limited labels in AD, we would call for more attention to semi-supervised AD methods which can leverage both the guidance from labels efficiently and the exploration of the unlabeled data. Regarding backbones,

the latest network architectures like Transformer and ensembling show their superiority in AD tasks.

8.4.3 Algorithm Performance under Different Types of Anomalies

Under four types of anomalies introduced in §8.3.3.2), we show the performances of unsupervised methods in Fig. 8.5, and then compare both semi- and fully-supervised methods in Fig. 8.6.

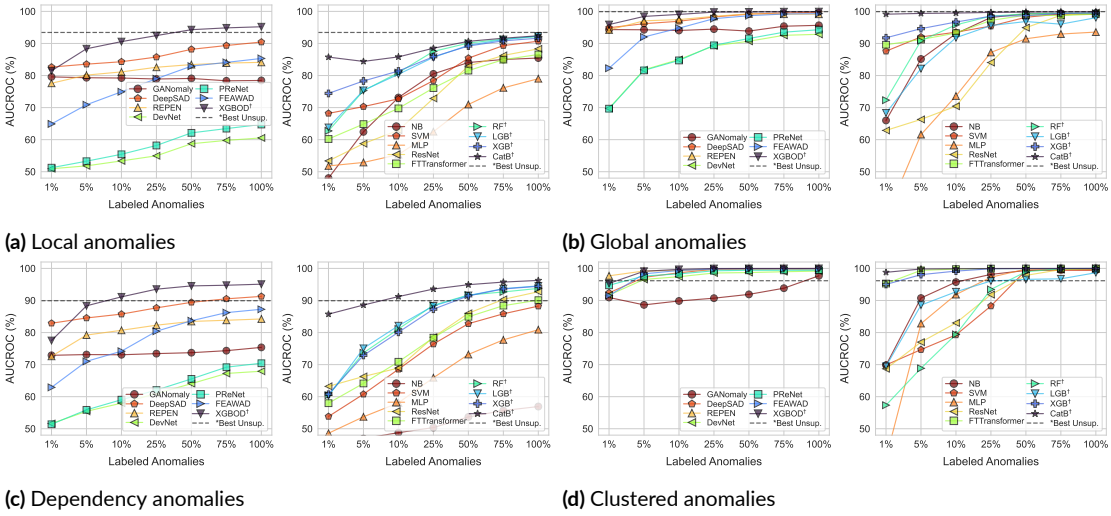


(a) Local anomalies **(b)** Global anomalies **(c)** Dependency anomalies **(d)** Clustered anomalies

Figure 8.5: Avg. rank (lower the better) of unsupervised methods on different types of anomalies. Groups of algorithms not significantly different are connected horizontally in the CD diagrams. The unsupervised methods perform well when their assumptions conform to the underlying anomaly type.

Performance of unsupervised algorithms highly depends on the alignment of its assumptions and the underlying anomaly type. As expected, *local* anomaly factor (LOF) is statistically better than other unsupervised methods for the local anomalies (Fig. 8.5a), and KNN, which uses k-th (*global*) nearest neighbor's distance as anomaly scores, is the statistically best detector for global anomalies (Fig. 8.5b). Again, there is no algorithm performing well on all types of anomalies; LOF achieves the best AUCROC on local anomalies (Fig. 8.5a) and the second best AUCROC rank on dependency anomalies (Fig. 8.5c), but performs poorly on clustered anomalies (Fig. 8.5d). Practitioners should select algorithms based on the characteristics of the underlying task, and consider the algorithm which may cover more high-interest anomaly types [161].

The "power" of prior knowledge on anomaly types may overweigh the usage of partial labels. For the local, global, and dependency anomalies, most label-informed methods perform worse than the best unsupervised methods of each type (corresponding to LOF, KNN, and KNN). For example, the detection performance of XGBOD for the local anomalies is inferior to the best unsu-



(a) Local anomalies

(b) Global anomalies

(c) Dependency anomalies

(d) Clustered anomalies

Figure 8.6: Semi- (left of each subfigure) and supervised (right) algorithms' performance on different types of anomalies with varying levels of labeled anomalies. Surprisingly, these label-informed algorithms are *inferior* to the best unsupervised method except for the clustered anomalies.

pervised method LOF when $\gamma_l \leq 50\%$, while other methods perform worse than LOF in all cases (See Fig. 8.6a). Why could not label-informed algorithms beat unsupervised methods in this setting? We believe that partially labeled anomalies cannot well capture all characteristics of specific types of anomalies, and learning such decision boundaries is challenging. For instance, different local anomalies often exhibit various behaviors, as shown in Fig. 8.3a, which may be easier to identify by a generic definition of "locality" in unsupervised methods other than specific labels. Thus, incomplete label information may bias the learning process of these label-informed methods, which explains their relatively inferior performances compared to the best unsupervised methods. This conclusion is further verified by the results of clustered anomalies (See Fig. 8.6d), where label-informed (especially semi-supervised) methods outperform the best unsupervised method OCSVM, as few labeled anomalies can already represent similar behaviors in the clustered anomalies (Fig. 8.3d).

Future Direction 3: Leveraging Anomaly Types as Valuable Prior Knowledge. The above results emphasize the importance of knowing anomaly types in achieving high detection performance even without labels, and call for attention to designing anomaly-type-aware detection algorithms.

In an ideal world, one may combine multiple AD algorithms based on the composition of anomaly types, via frameworks like dynamic model selection and combination [346]. To our knowledge, the latest advancement in this end [125] provides an equivalence criterion for measuring to what degree two anomaly detection algorithms detect the same kind of anomalies. Furthermore, future research may also consider designing semi-supervised AD methods capable of detecting different types of unknown anomalies while effectively improving performance by the partially available labeled data. Another interesting direction is to train an offline AD model using synthetically generated anomalies and then adapt it for online prediction on real-world datasets with likely similar anomaly types. Unsupervised domain adaption and transfer learning for AD [64, 330] may serve as useful references.

8.4.4 Algorithm Robustness under Noisy and Corrupted Data

In this section, we investigate the algorithm robustness (i.e., Δperformance; see absolute performance plot in [105] of different AD algorithms under noisy and data corruption described in §8.3.3.3. The default γ_l is set to 100% since we only care about the relative change of model performance. Fig. 8.7 demonstrates the results.

Unsupervised methods are more susceptible to duplicated anomalies. As shown in Fig. 8.7a, almost all unsupervised methods are severely impacted by duplicated anomalies. Their AUCROC deteriorates proportionally with the increase in duplication. When anomalies are duplicated by 6 times, the median ΔAUCROC of unsupervised methods is -16.43%, compared to that of semi-supervised methods -0.05% (Fig. 8.7b) and supervised methods 0.13% (Fig. 8.7c). One explanation is that unsupervised methods often assume the underlying data is imbalanced with only a smaller percentage of anomalies—they rely on this assumption to detect anomalies. With more duplicated anomalies, the underlying data becomes more balanced, and the minority assumption of anomalies is violated, causing the degradation of unsupervised methods. Differently, more balanced

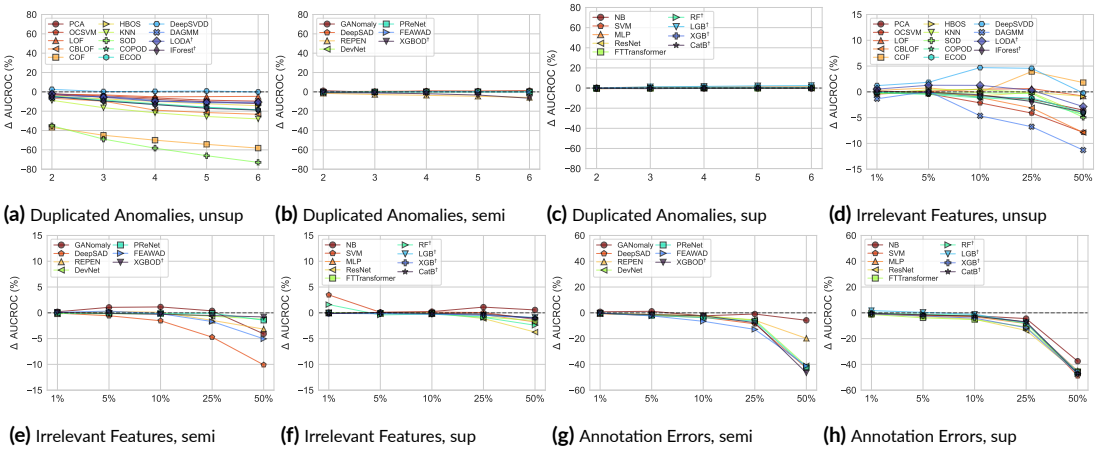


Figure 8.7: Algorithm performance change under noisy and corrupted data (i.e., duplicated anomalies for (a)-(c), irrelevant features for (d)-(f), and annotation errors for (g) and (h)). X-axis denotes either the duplicated times or the noise ratio. Y-axis denotes the % of performance change, and its range remains consistent across different algorithms. The results reveal unsupervised methods' susceptibility to duplicated anomalies and the usage of label information in defending irrelevant features. Un-, semi-, and fully-supervised methods are denoted as *unsup*, *semi*, and *sup*, respectively.

datasets do not affect the performance of semi- and fully-supervised methods remarkably, with the help of labels.

Irrelevant features cause little impact on supervised methods due to feature selection. Compared to the unsupervised and most semi-supervised methods, the training process of supervised methods is fully guided by the data labels ($\mathbf{y}$), therefore performing robustly to the irrelevant features (i.e., corrupted $\mathbf{X}$) due to the direct (or indirect) feature selection process. For instance, ensemble trees like XGBoost can filter irrelevant features. As shown in Fig. 8.7f, even the worst performing supervised algorithm (say ResNet) in this setting yields $\leq$ 5% degradation when 50% of the input features are corrupted by the uniform noises, while the un- and semi-supervised methods could face up to 10% degradation. Besides, the robust performances of supervised methods (and some semi-supervised methods like DevNet) indicate that the label information can be beneficial for feature selection. Also, Fig. 8.7f shows that minor irrelevant features (e.g., 1%) help supervised methods as regularization to generalize better.

Both semi- and fully-supervised methods show great resilience to minor annotation errors.

Although the detection performance of these methods is significantly downgraded when the annotation errors are severe (as shown in Fig. 8.7g and 8.7h), their degradation with regard to minor annotation errors is acceptable. The median ΔAUCROC of semi- and fully-supervised methods for 5% annotation errors is −1.52% and −1.91%, respectively. That being said, label-informed methods are still acceptable in practice as the annotation error should be relatively small [167, 326].

Future Direction 4: Noise-resilient AD Algorithms. Our results indicate there is an improvement space for robust unsupervised AD algorithms. One immediate remedy is to incorporate unsupervised feature selection [57, 224, 225] to combat irrelevant features. Moreover, label information could serve as effective guidance for model training against data noise, and it helps semi- and fully-supervised methods to be more robust. Given the difficulty of acquiring full labels, we suggest using semi-supervised methods as the backbone for designing more robust AD algorithms. Also, recent works on leveraging multiple sets of noisy labels collectively for learning AD models are also relevant [350].

8.5 Discussions

In this chapter, we introduce ADBench, the most comprehensive tabular anomaly detection benchmark with 30 algorithms and 57 benchmark datasets. Based on the analyses of multiple comparison angles, we unlock insights into the role of supervision, the importance of prior knowledge of anomaly types, and the principles of designing robust detection algorithms. On top of them, we summarize a few promising future research directions for anomaly detection, along with the fully released benchmark suite for evaluating new algorithms.

ADBench can extend to understand the algorithm performance with (*i*) mixed types of anomalies; (*ii*) different levels of (intrinsic) anomaly ratio; and (*iii*) more data modalities. Also, future benchmarks can consider the latest algorithms [54, 179, 269], and curate datasets from emerging

fields like drug discovery [118], interpretability and explainability [222, 324], and bias and fairness [63, 116, 222, 267, 279, 337].

9

ADMoE: Anomaly Detection with Multiple-set of Noisy Labels

This chapter is primarily based on:

Yue Zhao, Guoqing Zheng, Subhabrata Mukherjee, Robert McCann, and Ahmed Awadallah. "ADMoE: Anomaly Detection with Mixture-of-experts from Noisy Labels." *AAAI Conference on Artificial Intelligence (AAAI)*, 2023.

9.1 Highlight

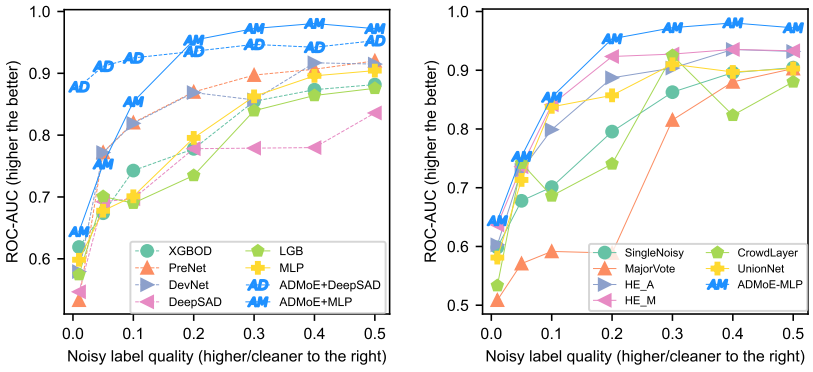
Although there are numerous anomaly detection (AD) algorithms [7, 228, 348, 182], existing AD methods assume the availability of (partial) labels that are *clean* (i.e. without noise), and cannot learn from weak/noisy labels*.

Simply treating noisy labels as (pseudo) clean labels leads to biased and degraded models [278]. Over the years, researchers have developed algorithms for classification and regression tasks to learn from noisy sources [250, 100, 308], which has shown great success. However, these methods are not tailored for AD with extreme data imbalance, and existing AD methods cannot learn from (multiple) noisy sources.

Why is it important to leverage noisy labels in AD applications? Taking malware detection as an example, it is impossible to get a large number of clean labels due to the data sensitivity and the cost of annotation. However, often there exists a large number of weak/noisy historical security rules designed for detecting malware from different perspectives, e.g., unauthorized network access and suspicious file movement, which have not been used in AD yet. Though not as perfect as human annotations, they are valuable as they encode prior knowledge from past detection experiences. Also, although each noisy source may be insufficient for difficult AD tasks, learning them jointly may build competitive models as they tend to *complement* each other.

In this chapter, we propose ADMoE, (to our knowledge) the *first weakly-supervised approach* for enabling anomaly detection algorithms to learn from *multiple sets of noisy labels*. In a nutshell, ADMoE enhances existing neural-network-based AD algorithms by Mixture-of-experts (MoE) network(s) [122, 266], which has a learnable gating function to activate different sub-networks (experts) based on the *incoming samples* and *their noisy labels*. In this way, the proposed ADMoE can jointly learn from multiple sets of noisy labels with the majority of parameters shared, while

*We use the terms *noisy* and *weak* interchangeably.



(a) Comparison with leading AD methods **(b)** Comp. w/ noisy learning methods

Figure 9.1: Performance (ROC-AUC) comparison on `Yelp` (see results on all datasets in §9.4.2 and 9.4.3), where ADMoE outperforms two groups of baselines: (a) SOTA AD methods; (b) leading classification methods for learning from multiple noisy sources. ADMoE enhanced DeepSAD and MLP are denoted as ***AD*** and ***AM***.

providing specialization and scalability via experts. Unlike existing noisy label learning approaches, ADMoE does not require explicit mapping from noisy labels to network parameters, providing better scalability and flexibility. To encourage ADMoE to develop specialization based on the noisy sources, we use noisy labels as (part of the) input features with learnable embeddings to make the gating function aware of them.

Key Results. Fig. 9.1 shows that a multiple layer perception (MLP) [251] enhanced by ADMoE can largely outperform both (9.1a) leading AD algorithms as well as (9.1b) noisy-label learning methods for classification. Note ADMoE is not strictly another detection algorithm, but *a general framework* to empower any neural-based AD methods to leverage multiple sets of weak labels. §9.4 shows extensive results on more datasets, and the improvement in enhancing more complex DeepSAD [255] with ADMoE.

In summary, the key contributions of this chapter include:

- **Problem formulation, baselines, and datasets**. We formally define the crucial problem of using *multiple sets of noisy labels for AD* (MNLAD), and release the first batch of baselines

and datasets for future research[†].

- **The first AD framework for learning from multiple noisy sources**. The proposed ADMoE is a novel method with Mixture-of-experts (MoE) architecture to achieve specialized and scalable learning for MNLAD.
- **Model-agnostic design**. ADMoE enhances any neural-network-based AD methods, and we show its effectiveness on MLP and state-of-the-art (SOTA) DeepSAD.
- **Effectiveness and real-world deployment**. We demonstrate ADMoE's SOTA performance on seven benchmark datasets and a proprietary enterprise security application, in comparison with two groups of leading baselines (13 in total). It brings on average 14% and up to 34% improvement over not using it, with the equivalent number of learnable parameters and FLOPs as baselines.

9.2 Related Work

9.2.1 Weakly-supervised Anomaly Detection

There exists some literature on weakly-supervised AD, and most of them fall under the "incomplete supervision" category. These semi-supervised methods assume access to a small set of clean labels and it is unclear how to extend them for multi-set noisy labels. Representative work includes XGBOD [344], DeepSAD [255], DevNet [230], PreNet [229]; see [105] for more details. We use these leading methods (that work with one set of labels) as baselines in §9.4.2, and demonstrate ADMoE's performance gain in leveraging multiple sets of noisy labels. A more recent and comprehensive survey can be found in [129].

[†]See code and appendix: https://github.com/microsoft/admoe

9.2.2 Learning from Single Set of Noisy Labels

There have been rich literature on learning from a single set of noisy labels, including learning a label corruption/transition matrix [234], correcting labels via meta-learning [351], and building robust training mechanisms like co-teaching [104], co-teaching+ [334], and JoCoR [307]. More details can be found in a recent survey [103]. These algorithms are primarily for a single set of noisy labels, and are not designed for AD tasks.

9.2.3 Learning from Multiple Noisy Sources

Table 9.1: Baselines and ADMoE for comparison with categorization by (first row) whether it uses multiple sets of weak labels, (second row) whether it only trains a single model, (third row) whether the training process is end-to-end and (the last row) whether it is scalable with regard to many sets of weak labels. ADMoE is an end-to-end, scalable paradigm.

Category	**SingleNoisy**	**LabelVote**	**HE_A**	**HE_M**	**CrowdLayer**	**UnionNet**	**ADMoE**
Multi-source	✗	✓	✓	✓	✓	✓	✓
Single-model	✓	✓	✗	✗	✓	✓	✓
End-to-end	✓	✗	✗	✗	✓	✓	✓
Scalability	High	High	Low	Low	Med	Med	High

MNLAD falls under *weakly supervised ML* [358], where it deals with *multiple* sets of *inaccurate/noisy* labels. Naturally, one can aggregate noisy labels to generate a "corrected" label set [351], while it may be challenging for AD with extreme data imbalance. Other than label correction, one may take ensembling to train multiple independent AD models for combination, e.g., one model per set of noisy labels, which however faces scalability issues while dealing with many sets of labels. What is worse, independently trained models fail to explore the interaction among noisy labels. Differently, end-to-end noisy label learning methods, including Crowd Layer [250], DoctorNet [100], and UnionNet [308], can directly learn from multiple sets of noisy labels and map each set of noisy labels to part of the network (e.g., transition matrix), encouraging the model to learn knowledge from all noisy labels collectively. Although they yield great performance in crowd-sourcing scenarios

with a small number of annotators, they do not scale in MNLAD with many sets of "cheap" noisy labels due to this explicit one-to-one mapping. Also, each set of noisy AD labels may be only good at certain anomalies as a biased annotator. Consequently, explicit one-to-one mapping (e.g., transition matrix) from a single set of labels to a network in existing classification works is not ideal for MNLAD. ADMoE lifts this constraint to allow many-to-many mapping, improving model scalability and robustness for MNLAD.

We summarize the methods for learning from multiple sources of noisy labels in Table 9.1 categorized as: **(1) SingleNoisy** trains an AD model using only one set of weak labels, which sets the lower bound of all baselines. **(2) LabelVote** trains an AD model based on the consensus of weak labels via majority vote. **(3) HyperEnsemble** [309] trains an individual model for each set of noisy labels (i.e., k models for k sets of labels), and combines their anomaly scores by averaging (i.e., **HE_A**) and maximizing (i.e., **HE_M**). **(4) CrowdLayer** [250] tries to reconstruct the input weak labels during training. **(5) UnionNet** [308] learns a transition matrix for all weak labels together. Note that they are primarily for classification and not tailored for anomaly detection. Nonetheless, we adapt the methods in Table 9.1 as baselines. In §9.4.3, we show ADMoE outperforms all these methods.

9.3 AD from Multiple Sets of Noisy Labels

In §9.3.1, we formally present the problem of AD with multiple sets of noisy labels, followed by the discussion on why multiple noisy sources help AD in §9.3.2. Motivated by above, we describe the proposed ADMoE framework in §9.3.3.

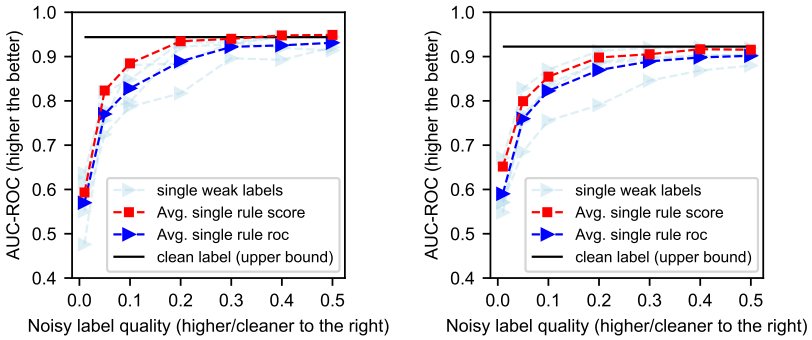
9.3.1 Problem Statement

We consider the problem of anomaly detection with multiple sets of weak/noisy labels. We refer to this problem as MNLAD, an acronym for using multiple sets of noisy labels for anomaly detection.

We present the problem definition here.

Problem 3 (MNLAD) Given *an anomaly detection task with input feature* $\mathbf{X} \in \mathbb{R}^{n \times d}$ *(e.g., n samples and d features) and t sets of noisy/weak labels* $\mathcal{Y}_w = \{\mathbf{y}_{w,1}, \ldots, \mathbf{y}_{w,t}\}$ *(each in* $\mathbb{R}^n$*), build a detection model M to leverage all information to achieve the best performance.*

Existing AD methods [7, 228, 105] can (at best) treat one set of noisy labels from $\mathcal{Y}^w$ as (pseudo) clean labels to train a model. None of them leverages multiple sets of noisy labels $\mathcal{Y}^w$ collectively.



(a) Result analysis on PreNet **(b)** Result analysis on XGBOD

Figure 9.2: Benefit of leveraging multiple noisy sources on `Yelp`: even simply averaging individual models' outputs (shown in red) is better than training each weak source independently (shown in blue).

9.3.2 Why and How Do Multiple Sets of Weak Labels Help in Anomaly Detection?

Benefits of joint learning in MNLAD. AD benefits from model combination and ensemble learning from diverse base models [8, 346, 71], and the improvement is expected when base models make *complementary* errors [359, 8]. Multiple sets of noisy labels are natural sources for ensembling with built-in diversity, as they reflect distinct detection aspects and historical knowledge. Fig. 9.2 shows that averaging the outputs from multiple AD models (each trained on one set of noisy labels) leads to better results than training each model independently; this observation holds true for both deep

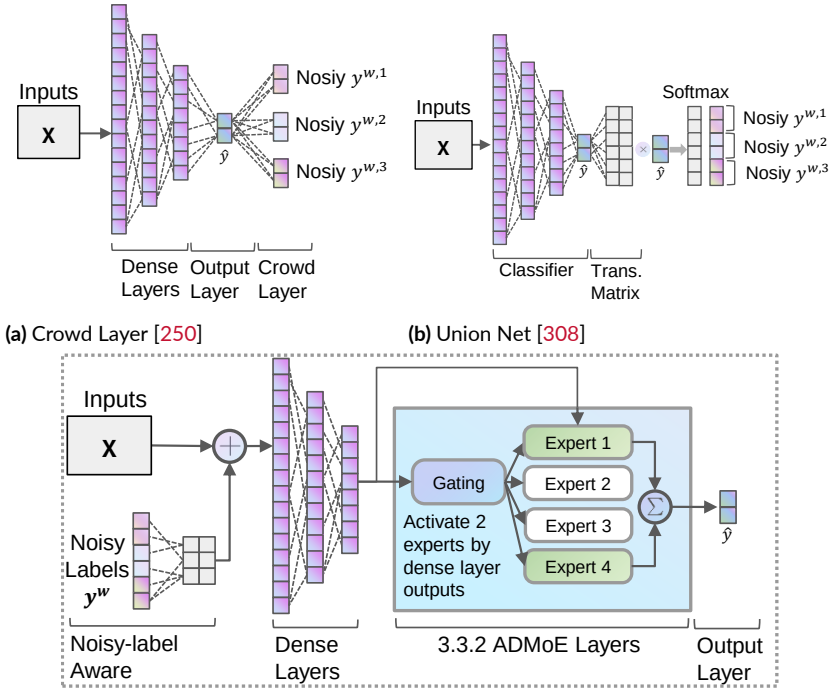


Figure 9.3: A toy example of a 3-layer MLP for AD; 3 sets of noisy labels $\mathbf{y}^{\mathbf{w}} = \{\mathbf{y}^{w,1}, \mathbf{y}^{w,2}, \mathbf{y}^{w,3}\}$ are assumed. Existing methods (*a* and *b*) learn to recover noisy labels explicitly, while ADMoE (§9.3.3.2) (*c*) uses an MoE architecture with noisy-label-aware expert activation to learn specialization from noisy sources without explicit label mapping. In the example, we add an ADMoE layer between the dense layers and the output layer; only the top two experts are activated for input samples.

(neural) AD models (e.g., PreNet in Fig. 9.2a) and shallow models (e.g., XGBOD in Fig. 9.2b). This example justifies *the benefit of learning from multiple sets of noisy labels*; even simple averaging in MNLAD can already "touch" the performance upper bound (i.e., training a model using all clean labels).

9.3.3 ADMoE: Specialized and Scalable Anomaly Detection with Multiple Sets of Noisy Labels

Motivation. After reviewing gaps and opportunities in existing works, we argue the ideal design for MNLAD should fulfill the following requirements: (*i*) encourage *specialization* from different noisy

sources but also explore their *similarity* (*ii*) *scalability* to handle an increasing number of noisy sources and (*iii*) *generality* to apply to various AD methods.

Overview of ADMoE. In this chapter, (for the first time) we adapt Mixture-of-experts (MoE) architecture/layer [122, 266] (see preliminary in §9.3.3.1) to AD algorithm design for MNLAD. Specifically, we propose model-agnostic ADMoE[‡] to enhance any neural network-based AD method via: (*i*) mixture-of-experts (MoE) layers to do specialized and scalable learning from noisy labels and (*ii*) noisy-label aware expert activation by using noisy labels as (part of the) input with learnable embedding to facilitate specialization. Refer to Fig. 9.3c for an illustration of applying ADMoE to a simple MLP, where we add an ADMoE layer between the dense layers and output layer for MNLAD, and use noisy labels directly as input with learnable embedding to help ADMoE to better specialize. Other neural-network-based AD methods can follow the same procedure to be enhanced by ADMoE (i.e., inserting ADMoE layers before the output layer and using noisy labels as input). In the following subsections, we give a short background of MoE and then present the design of ADMoE.

9.3.3.1 Preliminary on Mixture-of-experts (MoE) Architecture.

The original MoE [122] is designed as a dynamic learning paradigm to allow different parts (i.e., experts) of a network to specialize for different samples. More recent (sparsely-gated) MoE [266] has been shown to improve model scalability for natural language processing (NLP) tasks, where models can have billions of parameters [73]. The key difference between the original MoE and the sparse MoE is the latter builds more experts (sub-networks) for a large model but only activates a few of them for scalability and efficiency.

Basically, MoE splits specific layer(s) of a (large) neural network into m small "experts" (i.e., sub-networks). It uses top-k gating to activate k experts ($k < m$ or $k \ll m$ for sparse MoE) for each

[‡]Throughout the paper, we slightly abuse the term ADMoE to refer to both our overall framework and the proposed MoE layer.

input sample computation as opposed to using the entire network as standard dense models. More specifically, MoE uses a differentiable gating function $G(\cdot)$ to calculate the activation weights of each expert. It then aggregates the weighted outputs of the top-k experts with the highest activation weights as the MoE layer output. For example, the expert weights $\beta^j \in \mathbb{R}^m$ of the j-th sample is a function of input data by the gating $G(\cdot)$, i.e., $\beta^j = G(\mathbf{X}^j)$.

Till now, MoE has been widely used in multi-task learning [354], video captioning [304], and multilingual neural machine translation (NMT) [60] tasks. Taking NMT as an example, prior work [60] shows that MoE can help learn a model from diverse sources (e.g., different languages), where they share the most common parameters but also specialize for individual source via "experts" (sub-networks). We combine the strengths of original and sparse MoE for our ADMoE.

9.3.3.2 Capitalizing Mixture-of-experts (MoE) Architecture for MNLAD.

It is easy to see the ***connection*** between MoE's applications (e.g., multilingual machine translation) and MNLAD—in both cases MoE can help ***learn from diverse sources*** (e.g., multiple sets of noisy labels in MNLAD) to capture the *similarity via parameter sharing* while encouraging *specialization via expert learning*.

By recognizing this connection, we (for the first time), introduce MoE architecture for weakly supervised AD with noisy labels (i.e., MNLAD), called ADMoE. Similar to other works that use MoE, proposed ADMoE keeps (does not alter) an (AD) algorithm's original layers *before* the output layer (e.g., dense layers in an MLP) to explore the agreement among noisy sources via shared parameters, while inserting an MoE layer before the output layer to learn from each noisy source with specialization (via the experts/sub-networks). In an ideal world, each expert is good at handling samples from different noisy sources and updated only with more accurate sets of noisy labels. As the toy example shown in Fig. 9.3c to apply ADMoE on an MLP for AD, we insert an ADMoE layer between the dense layers and the output layer: where the ADMoE layer contains four experts, and the gating

activates only the top two for each input example.

For the j-th sample, the MoE layer's output O^j is shown in Eq. (9.1) as a weighted sum of all activated experts' outputs, where $E_i(\cdot)$ denotes the i-th expert network, and h^j is the output from the dense layer (as the input to ADMoE). β_i^j is the weight of the i-th expert assigned by the gating function $G(\cdot)$, and we describe its calculation in Eq. (9.2). Although Eq. (9.1) enumerates all m experts for aggregation, only the top-k experts with non-negative weights $\beta_i^j > 0$ are used.

$$O^j = \sum_{i=1}^{m} \beta_i^j E_i(h^j) \tag{9.1}$$

Improving ADMoE with Noisy-Label Aware Expert Activation. The original MoE calculates the activation weights with only the raw input feature $\mathbf{X}$ (see §9.3.3.1), which can be improved in MNLAD with the presence of noisy labels. To such end, we explicitly make the gating function $G(\cdot)$ aware of noisy labels $\mathcal{Y}_w$ while calculating the weights for (activating) experts. Intuitively, t sets of noisy labels can be expressed as t-element binary vectors (0 means normalcy and 1 means abnormality). However, it is hard for neural networks to directly learn binary inputs [48]. Thus, we propose to learn a $\mathbb{R}^{t\times e}$ continuous embedding $Emb(\cdot)$ for noisy labels (e is embedding dimension), and use both the raw input features $\mathbf{X}$ in d dims and the average of the embedding in e dims as the input of the gating. As such, the expert weights for the j-th sample β^j can be calculated with Eq. (9.2), and plugged back into Eq. (9.1) for ADMoE output.

$$\beta^j = G\left(\mathbf{X}^j, Emb(\mathcal{Y}_w^j)\right) \tag{9.2}$$

Loss Function. ADMoE is strictly an enhancement framework for MNLAD other than a new detection algorithm, and thus we do not need to design a new loss function but just enrich the original loss $\mathcal{L}_o$ of the underlying AD algorithm (e.g., cross-entropy for a simple MLP). While calculating the loss in each batch of data, we *randomly sample one set* of noisy labels treated as the (pseudo) clean labels or *combine the loss of all t sets* of noisy labels by treating them as (pseudo) clean labels. To

encourage all experts to be evenly activated and not collapse on a few of them [248], we include an additional loss term on gating (i.e, $\mathcal{L}_g$). Putting these together, we show the loss for the j-th training sample in Eq. (9.3), where the first term encourages equal activation of experts with α as the load balancing factor, and the second part is the loss of the j-th sample, which depends on the MoE layer output O^j in Eq. (9.1) and the corresponding noisy labels $\mathcal{Y}_w^j$ (either one set or all in loss calculation).

$$\mathcal{L}^j = \alpha \mathcal{L}_g + \mathcal{L}_o \left(O^j, \mathcal{Y}_w^j \right) \tag{9.3}$$

Remark on the Number of ADMoE Layers and Sparsity. Similar to the usage in NLP, one may use multiple ADMoE layers in an AD model (e.g., every other layer), especially for the AD models with complex structures like transformers [170]. In this chapter, we only show inserting the ADMoE layer before the output layer (see the illustration in Fig. 9.3c) of MLP, while future work may explore more extensive use of ADMoE layers in complex AD models. As AD models are way smaller [228], we do not enforce gating sparsity as in NLP models [266]. See ablation studies on the total number of experts and the number of activated experts in §9.4.4.3.

9.3.3.3 Advantages and Properties of ADMoE

Implicit Mapping of Noisy Labels to Experts with Better Scalability. The proposed ADMoE (Fig. 9.3c) is more scalable than existing noisy-label learning methods for classification, e.g., CrowdLayer (Fig. 9.3a) and UnionNet (Fig. 9.3b). As discussed in §9.2.3, these methods explicitly map each set of noisy labels to part of the network. With more sets of noisy labels, the network parameters designated for (recovering or mapping) noisy labels increase proportionately. In contrast, ADMoE does not enforce explicit one-to-one mapping from noisy labels to experts. Thus, its many-to-many mapping becomes more scalable with many noisy sources.

Extension with Clean Labels. Our design also allows for easy integration of clean labels. In many

AD tasks, a small set of clean labels are feasible, where ADMoE can easily use them. No update is needed for the network design, but simply treating the clean label as "another set of noisy labels" with higher weights. See §9.4.4.4 for experiment results.

9.4 Experiments

We design experiments to answer the following questions:

1. How do ADMoE-enhanced methods compare to SOTA AD methods that learn from a single set of labels? (§9.4.2)
2. How does ADMoE compare to leading (multiple sets of) noisy label learning methods for classification? (§9.4.3)
3. How does ADMoE perform under different settings, e.g., varying num. of experts and clean label ratios? (§9.4.4)

9.4.1 Experiment Setting

Table 9.2: Data description of the eight datasets used in this study: the top seven datasets are adapted from AD repo., e.g., DAMI [51] and ADBench [105], and security* is a proprietary enterprise-security dataset.

Data	# Samples	# Features	# Anomaly	% Anomaly	Category
ag news	10000	768	500	5.00	NLP
aloi	49534	27	1508	3.04	Image
mnist	7603	100	700	9.21	Image
spambase	4207	57	1679	39.91	Doc
svhn	5208	512	260	5.00	Image
Imdb	10000	768	500	5.00	NLP
Yelp	10000	768	500	5.00	NLP
security*	5525	21	378	6.84	Security

Benchmark Datasets. As shown in Table 9.2, we evaluate ADMoE on seven public datasets adapted from AD repositories [51, 105] and a proprietary enterprise-security dataset (with $t = 3$ sets of noisy labels). Note that these public datasets do not have existing noisy labels for AD, so we simulate $t = 4$ sets of noisy labels per dataset via two methods:

1. **Label Flipping** [351] generates noisy labels by uniformly swapping anomaly and normal classes at a designated noise rate.

2. **Inaccurate Output** uses varying percentages of ground truth labels to train t diverse classifiers, and considers their (inaccurate) predictions as noisy labels. With more ground truth labels to train a classifier, its prediction (e.g., noisy labels) will be more accurate—we control the noise levels by the availability of ground truth labels.

In this chapter, we use the noisy labels simulated by **Inaccurate Output** since that is more realistic and closer to real-world applications (e.g., noise is not random), while we also release the datasets by **Label Flipping** for broader usage.

We use varying percentages of ground truth labels to train t diverse classifiers (called noisy label generators), and consider their (inaccurate) predictions as noisy labels. Naturally, with more available ground truth labels to train a classifier, its prediction (e.g., noisy labels) will be more accurate—we therefore control the noise levels by the availability of ground truth labels.

Following this approach, we generate the noisy labels for the benchmark datasets at varying percentages of ground truth labels (namely, $\{0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5\}$) to train classifiers (a simple feed-forward MLP [251], decision tree [45], and ensemble methods Random Forest [44] and LightGBM [135]). Please see our code for more details.

Table 9.3 summarizes avg. ROC-AUC of the generated noisy labels while using varying percentages of ground truth labels to train noisy label generators. Of course, with more ground truth labels,

the generated noisy labels are also more accurate. Note that `security*` comes with actual noisy labels, and thus ROC-AUC does not vary.

Table 9.3: Avg. ROC-AUC of noisy labels at different noisy label qualities (higher the better). Note that security*'s noisy labels are not simulated and thus do not vary.

Clean Perc.	agnews	aloi	imdb	mnist	spamspace	svhn	yelp	security*
0.01	0.5384	0.5154	0.5085	0.7260	0.7862	0.5164	0.5192	0.6862
0.05	0.5663	0.5330	0.5304	0.7887	0.8929	0.5773	0.5455	0.6862
0.1	0.6210	0.5540	0.5628	0.8592	0.9047	0.6132	0.5754	0.6862
0.2	0.6641	0.5957	0.6025	0.9044	0.9182	0.6496	0.6271	0.6862
0.3	0.7065	0.6303	0.6356	0.9301	0.9301	0.6827	0.6621	0.6862
0.4	0.7462	0.6503	0.6758	0.9308	0.9401	0.7224	0.7077	0.6862
0.5	0.7662	0.6701	0.7118	0.9427	0.9482	0.7531	0.7212	0.6862

Two Groups of Baselines are described below:

1. *Leading AD methods that can only handle a single set of labels* to show ADMoE's benefit of leveraging multiple sets of noisy labels. These include *SOTA AD methods*: **(1) XGBOD** [344] **(2) PreNet** [229] **(3) DevNet** [230] **(4) DeepSAD** [255], and *popular classification methods*: **(5) MLP (6) XGBoost** [56] **(7) LightGBM** [135]. See detailed descriptions in [105].

2. *Leading methods (for classification) that handle multiple sets of noisy labels* (we adapt them for MNLAD; see §9.2.3 and Table 9.1 for details): **(1) SingleNoisy (2) LabelVote (3) HyperEnsemble** [309] that averages t models' scores (referred as **HP_A**) or takes their max (referred as **HP_M**) **(4) CrowdLayer** [250] and latest **(5) UnionNet** [308].

Backbone AD Algorithms, Model Capacity, and Hyperparameters. We show the generality of ADMoE to enhance (*i*) simple MLP and (*ii*) SOTA DeepSAD [255]. To ensure a fair comparison, we ensure all methods have the equivalent number of trainable parameters and FLOPs. All experiments are run on an Intel i7-9700 @3.00 GH, 64GB RAM, 8-core workstation with an NVIDIA Tesla V100 GPU.

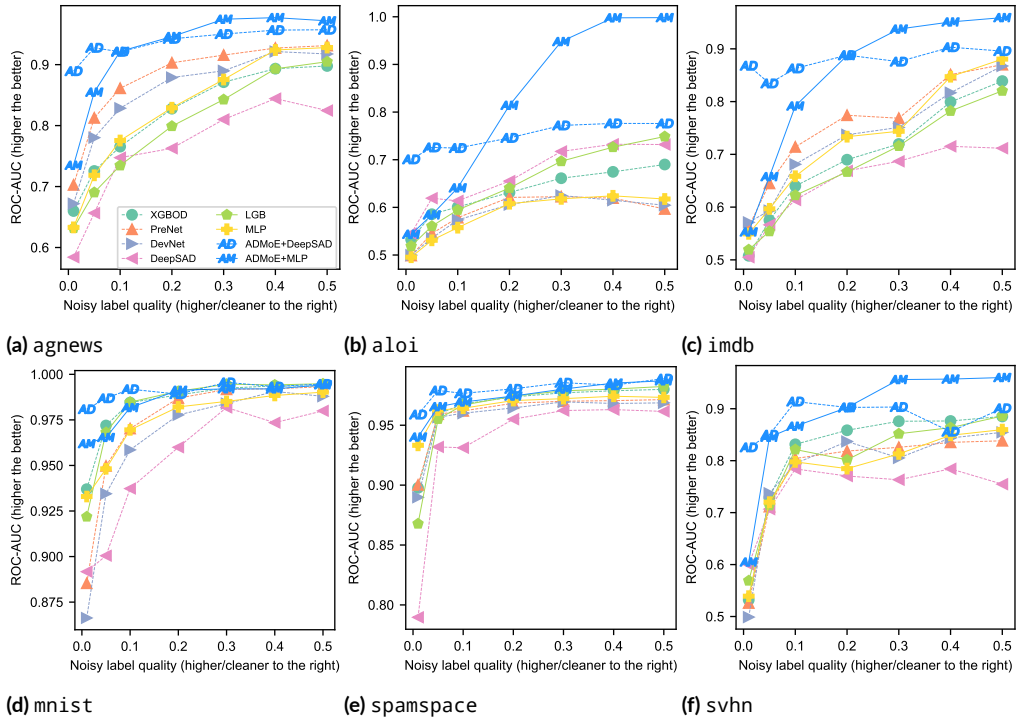


Figure 9.4: ROC-AUC comparison at different noisy label quality of ADMoE enhanced DeepSAD and MLP (denoted as ***AD*** and ***AM***) with leading AD methods that can only leverage one set of noisy labels. We already show `Yelp`'s result in Fig. 9.1a. Notably, ADMoE enabled ***AD*** and ***AM*** has significant performance improvement especially when the label quality is very low (to the left of the x-axis).

Table 9.4: Performance comparison between ADMoE-enhanced AD methods and leading AD methods (that can only use one set of labels) at noisy level 0.2. The best performance is highlighted in bold per dataset (row). ADMoE-based methods (ADMoE-MLP and ADMoE-DeepSAD denote ADMoE-enhanced MLP and DeepSAD) outperform all baselines. ADMoE brings on average 13.83% and up to 33.96% improvement over the original MLP, and on average 14.44% and up to 32.79% improvement over DeepSAD. Note that all the neural-network models use the equivalent numbers of parameters and training FLOPs.

Dataset	XGBOD	PreNet	DevNet	LGB	MLP	ADMoE-MLP	Δ Perf.	DeepSAD	ADMoE-DeepSAD	Δ Perf.
agnews	0.8277	0.903	0.8789	0.7991	0.8293	**0.946**	+14.07%	0.7627	0.9423	+23.55%
aloi	0.6312	0.6209	0.6057	0.6406	0.6078	**0.8142**	+33.96%	0.6552	0.7458	+13.83%
imdb	0.6898	0.7741	0.7372	0.6667	0.7335	0.8872	+20.95%	0.6688	**0.8881**	+32.79%
mnist	0.9887	0.9869	0.9777	0.9907	0.9821	**0.9913**	+0.94%	0.96	0.9891	+3.03%
spamspace	0.9736	0.9683	0.9645	0.9743	0.9706	0.9744	+0.39%	0.955	**0.9804**	+2.66%
svhn	0.8585	0.8185	0.837	0.8016	0.7847	0.9022	+14.97%	0.7705	**0.9025**	+17.13%
yelp	0.7778	0.8699	0.8689	0.7348	0.7955	**0.9535**	+19.86%	0.7781	0.9355	+20.23%
security*	0.7479	0.7415	0.7498	0.7335	0.7363	0.7767	+5.49%	0.7928	**0.8108**	+2.27%
Average	0.8119	0.8353	0.8274	0.7926	0.8049	**0.9056**	+13.83%	0.7929	0.8993	+14.44%

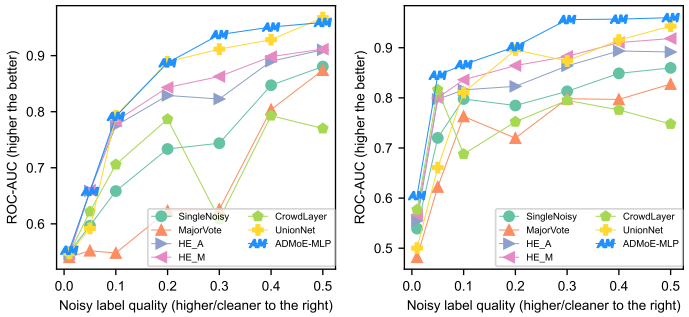
For all baselines used in this study, we use the same set of key hyperparameters for a fair comparison. We also use algorithms' default hyperparameter (HP) settings in the original paper for unique hyperparameters. More specifically, we use the same: (*i*) learning rate=0.001; (*ii*) batch size$=$ 256; and (*iii*) model size (number of trainable parameters $\approx$ 18, 000). We use the small validation set (5%) to choose the epoch with the highest validation performance.

Evaluation. For methods with built-in randomness, we run four independent trials and take the average, with a fixed dataset split (70% train, 25% for test, 5% for validation). Following AD research tradition [7, 346, 151, 105], we report ROC-AUC as the primary metric, while also showing additional results of average precision (AP) in [350].

9.4.2 Comparison Between ADMoE Methods and Leading AD Methods (Q1)

Fig. 9.1a and Fig. 9.4 show that **ADMoE enables MLP and DeepSAD to use multiple sets of noisy labels, which outperform leading AD methods that can use only one set of labels**, at varying noisy label qualities (x-axis). We further use Table 9.4 to compare them at noisy label quality 0.2 to understand the specific gain of ADMoE using multiple sets of noisy labels. The third block of the table shows that ADMoE brings on average 13.83%, and up to 33.96% (`aloi`; 3-rd row) improvements to a simple MLP. Additionally, the fourth block of the table further demonstrates that ADMoE enhances DeepSAD for using multiple noisy labels, with on average 14.44%, and up to 32.79% (`imdb`; 4-th row) gains. These results demonstrate the benefit of using ADMoE to enable AD algorithms to learn from multiple noisy sources.

ADMoE shows larger improvement when labels are more noisy (to the left of x-axis of Fig. 9.4). We observe that ADMoE-DeepSAD brings up to 60% of ROC-AUC improvement over the best-performing AD algorithm at noisy label quality 0.01 (see Fig. 9.4c for `imdb`). This observation is meaningful as we mostly need an approach to improve detection quality when the labels are extremely noisy, where ADMoE yields more improvement. Fig. 9.4 also demonstrates when the labels



(a) imdb **(b)** svhn

Figure 9.5: ROC-AUC comparison at different noisy label quality of ADMoE enhanced MLP (*AM*) with leading multi-set noisy-label learning methods. We already show Yelp's result in Fig. 9.1b. ADMoE outperforms most baselines.

are less noisy (to the right of the x-axis), the performance gaps among methods are smaller since the diversity among noisy sources is also reduced—using any set of noisy labels is sufficient to train a good AD model. Also, note that ADMoE-DeepSAD shows better performance than ADMoE-MLP with more noisy labels. This results from the additional robustness of the underlying semi-supervised method DeepSAD with access to unlabeled data.

9.4.3 Comparison Between ADMoE and Noisy Label Learning Methods (Q2)

Table 9.5: Performance comparison between ADMoE and leading noisy-label learning methods (in Table 9.1) on an MLP at noisy label quality 0.05. The best performance is highlighted in bold per dataset (row). ADMoE mostly outperforms baselines, with on avg. 9.4% and up to 19% improvement over SingleNoisy which trains w/ a set of noisy labels.

Dataset	Single Noisy	Major Vote	HE_A	HE_M	CrowdLayer	UnionNet	ADMoE
agnews	0.7185	0.5729	0.7824	0.8101	0.8543	0.7412	**0.8549**
aloi	0.531	0.479	0.5195	0.5579	0.5495	0.5773	**0.5842**
imdb	0.5967	0.5521	0.6587	**0.6599**	0.6219	0.5918	0.6577
mnist	0.9482	0.9467	0.9598	0.9563	0.9644	0.6767	**0.9655**
spamspace	0.9626	**0.9655**	0.9584	0.9586	0.7654	0.9567	**0.9655**
svhn	0.7201	0.6222	0.7971	0.7998	0.8161	0.6608	**0.8451**
yelp	0.6777	0.5708	0.7298	0.7358	0.7418	0.7133	**0.7533**
security*	0.7363	0.7653	0.7526	0.7484	0.7374	0.7435	**0.7767**
Average	0.7363	0.6843	0.7722	0.7783	0.7563	0.7014	**0.8003**

ADMoE also shows an edge over SOTA *classification* methods that learn from multiple sets

Table 9.6: Perf. breakdown of each expert and a comparison model (w/ the same capacity as each expert but trained independently; last col.) on subsamples activated for each expert by MoE on `Yelp`. We highlight the best model per row in **bold**. The specialized expert performs the best in their assigned subsamples by gating, i.e., the diagonal is all in bold.

Activated Expert	Expert 1	Expert 2	Expert 3	Expert 4	Comp.
Subsamples for Expert 1	**0.8554**	0.8488	0.8458	0.8440	0.7741
Subsamples for Expert 2	0.8995	**0.9043**	0.8913	0.8943	0.7976
Subsamples for Expert 3	0.8633	0.8643	**0.8729**	0.8665	0.8049
Subsamples for Expert 4	0.7903	0.7888	0.7775	**0.8066**	0.7134

of noisy labels[§], as shown in Fig. 9.1b and 9.5. We also analyze the comparison at noisy label quality 0.05 in Table 9.5. where ADMoE ranks the best in 7 out of 8 datasets. In addition to better detection accuracy, ADMoE only builds a single model, and is thus faster than **HE_E** and **HE_M** which requires building t independent models to aggregate predictions. We credit ADMoE's performance over SOTA algorithms including CrowdLayer and UnionNet to its implicit mapping of noisy labels to experts (§9.3.3.3).

9.4.4 Ablation Studies and Other Analysis (Q3)

9.4.4.1 Case Study: How does ADMoE Help?

Although MoE is effective in various NLP tasks, it is often challenging to contextualize how each expert responds to specific groups of samples due to the complexity of NLP models [266, 354, 360]. Given that AD models are much smaller and we use only one MoE layer before the output layer, we present an interesting case study (MLP with 4 experts where top 1 gets activated) in Table 9.6. We find each expert achieves the highest ROC-AUC on the subsamples where it gets activated by gating (see the diagonal), and they are significantly better than training an individual model with the same architecture (see the last column). Thus, each expert does develop a specialization in the subsamples they are "responsible" for.

[§]We adapt these classification methods (Table 9.1) for MNLAD.

9.4.4.2 Ablation on Using MoE and Noisy Labels as Inputs in ADMoE.

Additionally, we analyze the effect of using (*i*) ADMoE layer and (*ii*) noisy labels as input features in Fig. 9.6. First, ADMoE performs the best while using these two techniques jointly in most cases, and significantly better than not using them. Second, ADMoE helps the most when labels are noisier (to the left of the x-axis) with an avg. of 2% improvement over only using noisy labels as input. As expected, its impact is reduced with less noisy labels (i.e., closer to the ground truth): in that case, noisy labels are more similar to each other and specialization with ADMoE is therefore limited.

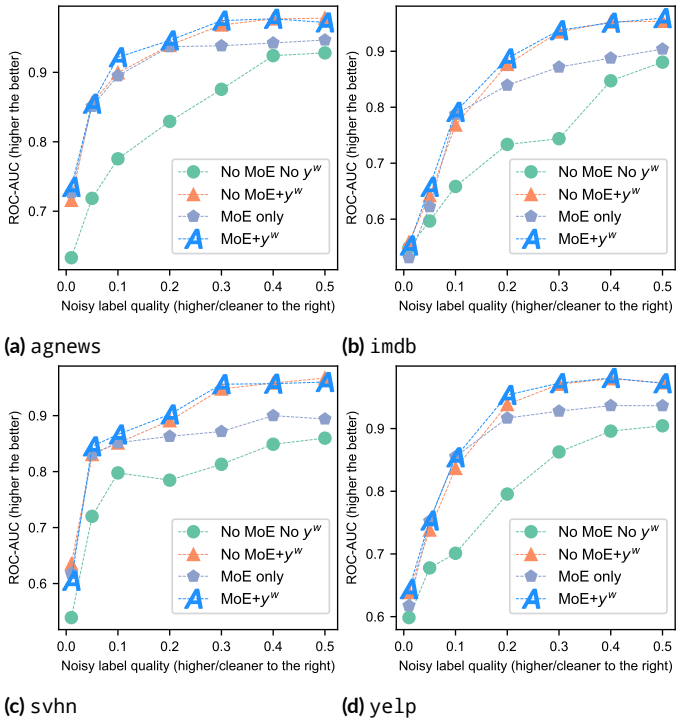


Figure 9.6: Ablation studies on (*i*) the use of ADMoE layer and (*ii*) the noisy labels $\mathbf{y}_w$ as input. ADMoE (A) using both techniques shows the best results at (nearly) all settings.

9.4.4.3 Effect of Number of Experts (m) and top-k Gating.

We vary the number of activated (x-axis) and total (y-axis) experts in ADMoE, and compare their detection accuracy in Fig. 9.7. The results suggest that the best choice is data-dependent, and we use $m = 4$ and $k = 2$ for all the datasets in the experiments.

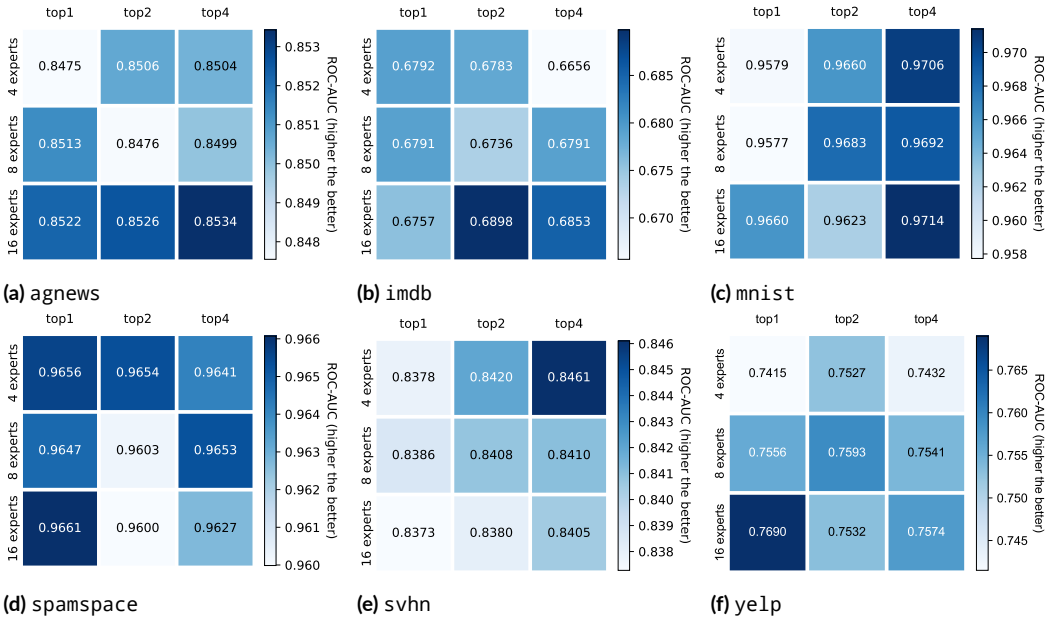


Figure 9.7: Ablation studies on key hyperparameters in ADMoE: (x-axis) the number of experts and (y-axis) top-k experts to activate. We show the results at noisy level 0.05, and find the best setting is data-dependent.

9.4.4.4 Performance on Varying Number of Clean Labels.

As introduced in §9.3.3.3, one merit of ADMoE is the easy integration of clean labels when available. Fig. 9.8 shows that ADMoE can leverage the available clean labels to achieve higher performance. Specifically, ADMoE with only 8% clean labels can achieve similar or better results than that of using all available clean labels on `spamspace` and `svhn`.

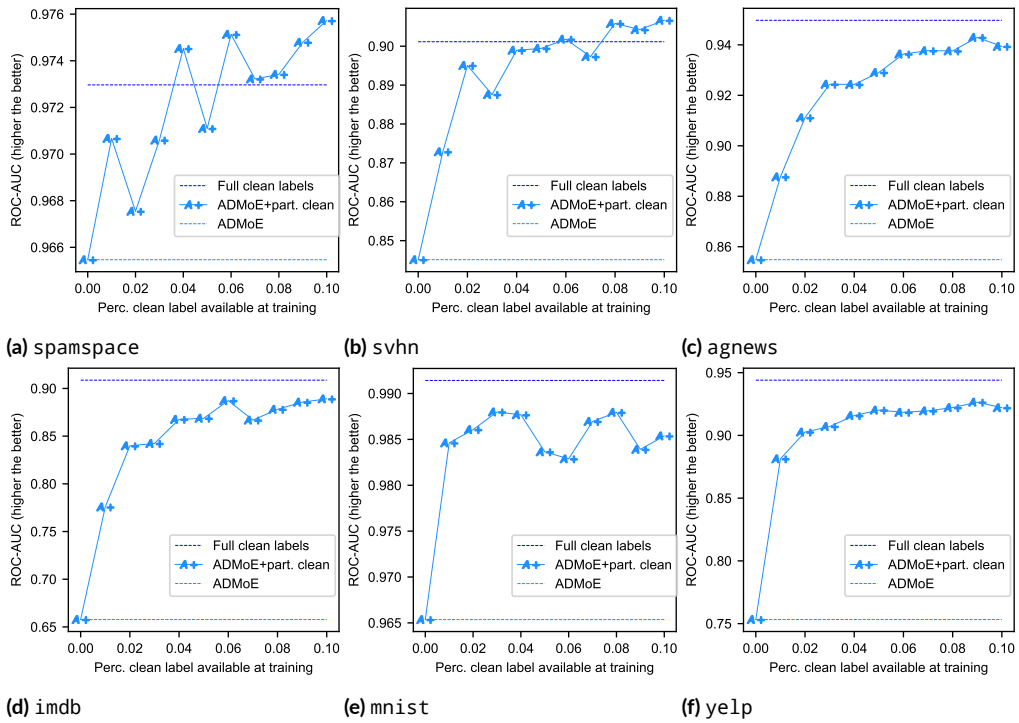


Figure 9.8: Analysis of integrating varying percentages (from 1% to 10%) of *additional* clean labels in ADMoE. The results show that ADMoE efficiently leverages the clean labels with increasing performance.

9.5 Discussions

We propose ADMoE, a model-agnostic learning framework, to enable anomaly detection algorithms to learn from multiple sets of noisy labels. Leveraging Mixture-of-experts (MoE) architecture from the NLP domain, ADMoE is scalable in building specialization based on the diversity among noisy sources. Extensive experiments show that ADMoE outperforms a wide range of leading baselines, bringing on average 14% improvement over not using it. Future work can explore its usage with complex detection algorithms.

Part IV

Concluding Remarks

10

Summary and Future Directions

10.1 Summary of Existing Work

In Chapter 1, we discuss internal model evaluation strategies for selecting a model for unsupervised outlier detection. The strategies rely solely on input data and outlier scores. The chapter surveys existing unsupervised outlier model selection (UOMS) methods, adapts deep representation learning techniques, and designs new methods. The effectiveness of these strategies is systematically evalu-

ated on a large testbed of real-world outlier detection datasets. The findings suggest that existing strategies designed for UOMS are ill-suited, and consensus-based methods are more competitive but not different from the best detector in the pool. The chapter contributes to a unified comparison of internal model evaluation strategies for UOMS and identifies new techniques.

Chapter 2 proposes MetaOD, an approach to UOMS based on meta-learning. MetaOD leverages the prior performances of a large collection of existing detection models on historical outlier detection benchmark datasets to estimate a candidate model's performance on a new task with no labels. Interestingly, we establish a connection between the UOMS problem and the cold-start problem in collaborative filtering, where the new task in UOMS is akin to a new user in CF with no available evaluations, and the model space is analogous to the item-set. Based on this connection, we redesigned existing collaborative filtering methods for UOMS.

Notably, we design novel meta-features to capture the outlying characteristics in a dataset, and show through extensive experiments on two benchmark testbeds that selecting a model by MetaOD significantly outperforms always using a popular model like iForest and other possible meta-learning approaches tailored for UOMS. Moreover, MetaOD incurs negligible run-time overhead at test time.

Chapter 3 proposes a novel approach called ELECT, an iterative meta-learning approach that selects a high-performance model for a new task by capitalizing on historical OD tasks without using any labels. Unlike prior work, ELECT uses performance-driven task similarity instead of hand-crafted meta-features to quantify task similarity. ELECT learns to quantify performance based on internal model performance measures and carefully decides which model to train on the new task iteratively, reducing computation time. Extensive experiments show that ELECT is significantly better than popular models and all meta-feature baselines, including MetaOD.

Chapter 4 discusses the challenge of optimizing hyperparameters for unsupervised outlier detection algorithms without labels. The proposed approach, called HPOD, leverages meta-learning to enable

efficient unsupervised OD hyperparameter optimization. HPOD uses a meta-database with historical performances of a large collection of hyperparameters on existing OD benchmark datasets and trains a proxy performance evaluator to evaluate hyperparameters on a new dataset without labels. HPOD is strictly an HP optimization method, not a new detection algorithm. The results show that HPOD outperforms baselines on various datasets with different types of hyperparameters.

Chapter 5 introduces PyOD—a valuable tool for practitioners and researchers in fields that process large amounts of unlabelled data and require a means to reliably perform pattern recognition, such as fraud detection in finance, fault diagnosis in mechanics, and pathology detection in medical imaging.

PyOD fills a gap in the Python ecosystem by providing a dedicated toolkit for outlier detection that includes more than 40 algorithms, combination methods for merging the results of multiple detectors and outlier ensembles, a unified API, detailed documentation, interactive examples, unit testing with cross-platform continuous integration, code coverage and code maintainability checks, optimization instruments for scalable outlier detection, and compatibility with major operating systems.

Chapter 6 presents a comprehensive system for accelerating heterogeneous outlier detection (OD) models called SUOD. The system includes three independent and complementary modules focusing on data level, model level, and execution level. The data level module uses random projection for high-dimensional data compression, the model level module employs pseudo-supervised approximation for fast prediction on new-coming samples, and the execution level module balances parallel scheduling for distributed task scheduling. We conduct extensive experiments to show the effectiveness of the acceleration modules independently and the entire framework as a whole, along with a real-world case on fraud detection.

Chapter 7 discusses TOD, the first tensor-based system for efficient and scalable outlier detection on distributed multi-GPU machines. The system decomposes complex OD applications into a small

collection of basic tensor algebra operators, allowing the system to accelerate OD computations by leveraging recent advances in deep learning infrastructure in both hardware and software. TOD leverages both the software and hardware optimizations in modern DNN frameworks to enable efficient and scalable OD computations on distributed multi-GPU clusters. The system addresses three major obstacles, including representing OD computations using tensor operations, quantizing OD computations, and enabling scalable OD computations.

Chapter 8 describes the ADBench benchmark, which evaluates the performance of 30 anomaly detection algorithms on 57 datasets with varying levels of supervision, different types of anomalies, and noisy and corrupted data. ADBench is designed to address the limitations of previous benchmark works, such as the primary focus on unsupervised methods, limited analysis of algorithm performance concerning anomaly types, and the absence of statistical tests for algorithm comparison. We identify insights into the role of supervision and anomaly types, as well as future research directions, which we further elaborate in the future direction section (§10.2).

Chapter 9 proposes a method called ADMoE, which is a framework for anomaly detection algorithms to learn from weak or noisy labels instead of clean labels. ADMoE leverages the Mixture-of-Experts (MoE) architecture to encourage specialized and scalable learning from multiple noisy sources. The framework captures similarities among noisy labels by sharing most model parameters, while encouraging specialization by building expert sub-networks. ADMoE uses noisy labels as input features to facilitate expert learning. The results show that ADMoE brings up to a 34

10.2 Future Directions

10.2.1 Multi-objective Optimization for OD Automation

This thesis presents a few automated paradigms for OD in Chapter 1-4, while the golden rule for selection and optimization is the model/algorithm accuracy. However, it is crucial to consider other

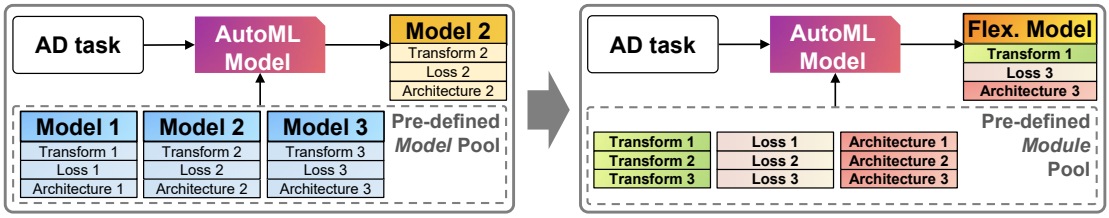


Figure 10.1: Future research (right) can make current AD model selection (left) more flexible

factors when designing such an automation framework, such as time consumption and memory usage.

Of course, accuracy is important to ensure that the system correctly identifies true outliers while minimizing false positives. However, time consumption and memory usage are equally important to ensure that the system can handle large datasets and operate efficiently in real-time scenarios. Therefore, by considering these factors all together via ***joint optimization***, we can create an automated OD paradigm that is accurate, efficient, and scalable to handle large datasets, which is crucial for practical data analysis and ML.

10.2.2 Pipeline Optimization for OD Automation

Current automated AD merely chooses a model from a pre-defined model pool (such as in Chapter 1-3), other than finding an AD model that actually tailors for the underlying task. If we consider all design choices/modules of an AD model, including data transformation, loss function, network architectures, etc., as tunable modules, then we gain the flexibility of intelligently selecting each module, resulting in more customized solutions with better performance.

Fig. 10.1 compares the difference between existing approaches (on the left), such as MetaOD and ELECT, to the proposed future plan (on the right). Note the proposed approach is more flexible to dynamically build a new AD model by selecting underlying modules, while the existing methods can only select from the static pool of models.

10.2.3 Automated Heterogeneous Device Selection for OD

As we have discussed in Chapter 7, there has been a significant advancement of hardware in the last decade, where GPUs are a good example. More comprehensively, we have access to CPUs (Central Processing Units), GPUs (Graphics Processing Units), TPUs (Tensor Processing Units), FPGAs (Field-Programmable Gate Arrays), and ASICs (Application-Specific Integrated Circuits) for different sorts of ML applications nowadays. Ultimately, the choice of hardware will depend on the specific needs of the machine learning task at hand, including the dataset's size, the algorithm's complexity, and the available budget.

With this in mind, one exciting direction for OD is to automate the selection of hardware, especially under the setting of heterogeneous devices. In short, heterogeneous devices are used in ML because different types of algorithms may require different types of hardware to achieve optimal performance. For example, a deep learning algorithm may be best suited to run on a GPU, while a reinforcement learning algorithm may be better suited to run on a CPU. The same situation applies to OD as well.

By combining multiple types of hardware into a single system, heterogeneous devices can offer faster performance and more efficient use of computing resources than using a single type of hardware. Additionally, heterogeneous devices can be customized to meet the specific needs of a particular ML task.

How to automate the heterogeneous device selection is an interesting future direction as we could leverage all possible devices for joint performance optimization for specialized OD applications. One approach is to leverage meta-learning to train a large corpus of hardware combinations and reuse the same configuration for a new task by similarity to historical tasks. This may be viewed as an extension of Chapter 2 and 3 onto hardware selection.

10.2.4 Weakly Supervised Outlier Detection

As shown in Chapter 8, semi-supervised OD methods yield significantly better performance than unsupervised methods, and may become the new frontier of OD research*.

More concretely, these semi-supervised methods address a specific setting called **OD with incomplete supervision**, where only part of the input data is labeled and fully-supervised models do not apply. This setting is common due to the high data labeling cost [105, 228]. For instance, accessing the complete label set of credit card fraud requires annotating all the transactions manually, which is time-consuming and costly. This problem can be formally defined as follow.

Problem 4 (OD with incomplete supervision) Given *an anomaly detection task with input data* $\mathbf{X} \in \mathbb{R}^{n \times d}$ *(e.g., n samples and d features) and partial labels* $\mathbf{y}_p \in \mathbb{R}^p$ *where* $p < n$*, train a detection model M with* $\{\mathbf{X}, \mathbf{y}_p\}$*. Output the anomaly scores on the test data* $\mathbf{X}_{test} \in \mathbb{R}^{m \times d}$*, i.e.,* $\mathbf{O}_{test} := M(\mathbf{X}_{test}) \in \mathbb{R}^{m \times 1}$.

In addition to representative weakly-supervised methods benchmarked in Chapter 8, there are a few promising future directions for OD with incomplete supervision worth noting.

Efficient label acquirement. One immediate relief is to acquire more labels efficiently via active learning and human-in-the-loop. However, it remains underexplored in choosing valuable samples (i.e., anomalies) for annotations. In addition to the above methods discussed in this survey, one promising approach is to use learning-to-rank that dynamically adjusts the sample order for annotators so that anomalies are assigned a higher probability to be labeled [152].

Few-shot/meta learning. Meta-learning can be applied to transfer supervision signals from similar datasets with more supervision [70], which makes it possible to incorporate more information instead of focusing on a single dataset.

*Part of this section is adapted from [129]

Balancing the weights of non-labeled and labeled data. As shown in Eq.10.1, most of the above representation learning methods' loss needs setting the balance factor α between the non-labeled and labeled data, which is an important hyperparameter here. The current approach primarily sets aside a hold-out data for setting it, while this may not be ideal for already insufficient labeled anomalies. One promising solution is to use automated ML to set it, which is discussed below.

$$\mathcal{L} = \underbrace{\mathcal{L}_u(\mathbf{X})}_{\text{non-labeled}} + \alpha \underbrace{\mathcal{L}_l(\mathbf{X}, \mathbf{y}_p)}_{\text{labeled}} \tag{10.1}$$

References

[1] Aas, K., Czado, C., Frigessi, A., & Bakken, H. (2009). Pair-copula constructions of multiple dependence. *Insurance: Mathematics and economics*, 44(2), 182–198.

[2] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P. A., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., & Zheng, X. (2016). Tensorflow: A system for large-scale machine learning. In K. Keeton & T. Roscoe (Eds.), *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016* (pp. 265–283).: USENIX Association.

[3] Abdulrahman, S. M., Brazdil, P., van Rijn, J. N., & Vanschoren, J. (2018). Speeding up algorithm selection using average ranking and active testing by introducing runtime. *Mach. Learn.*, 107(1), 79–108.

[4] Achlioptas, D. (2001). Database-friendly random projections. In P. Buneman (Ed.), *Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 21-23, 2001, Santa Barbara, California, USA*: ACM.

[5] Achtert, E., Kriegel, H., Reichert, L., Schubert, E., Wojdanowski, R., & Zimek, A. (2010). Visual evaluation of outlier detection models. In H. Kitagawa, Y. Ishikawa, Q. Li, & C. Watanabe (Eds.), *Database Systems for Advanced Applications, 15th International Conference, DASFAA 2010, Tsukuba, Japan, April 1-4, 2010, Proceedings, Part II*, volume 5982 of *Lecture Notes in Computer Science* (pp. 396–399).: Springer.

[6] Aeberhard, S., Coomans, D., & de Vel, O. (1992). The classification performance of rda. *Dept. of Computer Science and Dept. of Mathematics and Statistics, James Cook University of North Queensland, Tech. Rep*, (pp. 92–01).

[7] Aggarwal, C. C. (2013). *Outlier Analysis*. Springer.

[8] Aggarwal, C. C. (2018). *Neural Networks and Deep Learning - A Textbook*. Springer.

[9] Aggarwal, C. C. & Sathe, S. (2015). Theoretical foundations and algorithms for outlier ensembles. *SIGKDD Explor.*, 17(1), 24–47.

[10] Aggarwal, C. C., Zhao, Y., & Philip, S. Y. (2011). Outlier detection in graph streams. In *2011 IEEE 27th international conference on data engineering* (pp. 399–409).: IEEE IEEE.

[11] Ahmed, M., Mahmood, A. N., & Islam, M. R. (2016). A survey of anomaly detection techniques in financial domain. *Future Generation Computer Systems*, 55, 278–288.

[12] Aissa, N. B. & Guerroumi, M. (2016). Semi-supervised statistical approach for network anomaly detection. *Procedia Computer Science*, 83, 1090–1095.

[13] Akcay, S., Ameln, D., Vaidya, A., Lakshmanan, B., Ahuja, N., & Genc, U. (2022). Anomalib: A deep learning library for anomaly detection. *ArXiv preprint*, abs/2202.08341.

[14] Akcay, S., Atapour-Abarghouei, A., & Breckon, T. P. (2018). Ganomaly: Semi-supervised anomaly detection via adversarial training. In *ACCV* (pp. 622–637).

[15] Akçay, S., Atapour-Abarghouei, A., & Breckon, T. P. (2019). Skip-ganomaly: Skip connected and adversarially trained encoder-decoder anomaly detection. In *IJCNN* (pp. 1–8).: IEEE.

[16] Akoglu, L. (2014). Quantifying political polarity based on bipartite opinion networks. In *ICWSM*: The AAAI Press.

[17] Aldrich, N., Crowder, J., & Benson, B. (2014). How much does medicare lose due to fraud and improper payments each year. *The Sentinel*.

[18] Alimoglu, F. & Alpaydin, E. (1996). Methods of combining multiple classifiers based on different representations for pen-based handwritten digit recognition. In *TAINN*: Citeseer.

[19] Almardeny, Y., Boujnah, N., & Cleary, F. (2020). A novel outlier detection method for multivariate data. *IEEE Transactions on Knowledge and Data Engineering*.

[20] Alpaydin, E. & Kaynak, C. (1998). Cascading classifiers. *Kybernetika*, 34(4), 369–374.

[21] Alshawabkeh, M., Jang, B., & Kaeli, D. R. (2010). Accelerating the local outlier factor algorithm on a GPU for intrusion detection systems. In D. R. Kaeli & M. Leeser (Eds.), *Proceedings of 3rd Workshop on General Purpose Processing on Graphics Processing Units, GPGPU 2010, Pittsburgh, Pennsylvania, USA, March 14, 2010*, volume 425 of *ACM International Conference Proceeding Series* (pp. 104–110).: ACM.

[22] Amazon (2020). The total cost of ownership (tco) of amazon sagemaker. Accessed: 2-21-2021.

[23] André, F., Kermarrec, A.-M., & Le Scouarnec, N. (2016). Cache locality is not enough: High-performance nearest neighbor search with product quantization fast scan. In *VLDB*, volume 9 (pp.12).: VLDB Endowment.

[24] Angiulli, F., Basta, S., Lodi, S., & Sartori, C. (2010). A distributed approach to detect outliers in very large data sets. In *Euro-Par 2010 - Parallel Processing* (pp. 329–340). Berlin, Heidelberg: Springer Berlin Heidelberg.

[25] Angiulli, F., Basta, S., Lodi, S., & Sartori, C. (2016). GPU strategies for distance-based outlier detection. *IEEE Trans. Parallel Distributed Syst.*, 27(11), 3256–3268.

[26] Angiulli, F. & Pizzuti, C. (2002). Fast outlier detection in high dimensional spaces. In *ECML/PKDD* (pp. 15–27).: Springer.

[27] Ayres-de Campos, D., Bernardes, J., Garrido, A., Marques-de Sa, J., & Pereira-Leite, L. (2000). Sisporto 2.0: a program for automated analysis of cardiotocograms. *Journal of Maternal-Fetal Medicine*.

[28] Azmandian, F., Yilmazer, A., Dy, J. G., Aslam, J. A., & Kaeli, D. R. (2012). Gpu-accelerated feature selection for outlier detection using the local kernel density ratio. In M. J. Zaki, A. Siebes, J. X. Yu, B. Goethals, G. I. Webb, & X. Wu (Eds.), *ICDM* (pp. 51–60).: IEEE Computer Society.

[29] Bagdoyan, S. J. (2018). Medicare: Actions needed to better manage fraud risks. *https://www.gao.gov/assets/700/693156.pdf*.

[30] Bahri, M., Salutari, F., Putina, A., & Sozio, M. (2022). Automl: state of the art with a focus on anomaly detection, challenges, and research directions. *IJDSA*, (pp. 1–14).

[31] Baur, C., Wiestler, B., Albarqouni, S., & Navab, N. (2018). Deep autoencoding models for unsupervised anomaly segmentation in brain mr images. *arXiv preprint arXiv:1804.04488*.

[32] Bergman, L. & Hoshen, Y. (2020). Classification-based anomaly detection for general data. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*: OpenReview.net.

[33] Bergmann, P., Fauser, M., Sattlegger, D., & Steger, C. (2019). Mvtec AD - A comprehensive real-world dataset for unsupervised anomaly detection. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019* (pp. 9592–9600).: Computer Vision Foundation / IEEE.

[34] Bergstra, J. & Bengio, Y. (2012). Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13, 281–305.

[35] Bergstra, J., Komer, B., Eliasmith, C., Yamins, D., & Cox, D. D. (2015). Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, 8(1).

[36] Berthonnaud, E., Dimnet, J., Roussouly, P., & Labelle, H. (2005). Analysis of the sagittal balance of the spine and pelvis using shape and orientation parameters. *Clinical Spine Surgery*, 18(1), 40–47.

[37] Bhaduri, K., Matthews, B. L., & Giannella, C. R. (2011). Algorithms for speeding up distance-based outlier detection. In *KDD* (pp. 859–867).: ACM.

[38] Bharadhwaj, H. (2019). Meta-learning for user cold-start recommendation. In *IJCNN* (pp. 1–8).: IEEE.

[39] Bilalli, B., Abelló Gamazo, A., & Aluja Banet, T. (2017). On the predictive power of meta-features in OpenML. *International Journal of Applied Mathematics and Computer Science*, 27(4), 697–712.

[40] Blackard, J. A. & Dean, D. J. (1999). Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and electronics in agriculture*, 24(3), 131–151.

[41] Blalock, D. W. & Guttag, J. V. (2017). Bolt: Accelerated data mining with fast vector compression. In *KDD* (pp. 727–735).: ACM.

[42] Boehm, M., Reinwald, B., Hutchison, D., Sen, P., Evfimievski, A. V., & Pansare, N. (2018). On optimizing operator fusion plans for large-scale machine learning in systemml. *Proc. VLDB Endow.*, 11(12), 1755–1768.

[43] Borisov, V., Leemann, T., Seßler, K., Haug, J., Pawelczyk, M., & Kasneci, G. (2021). Deep neural networks and tabular data: A survey. *ArXiv preprint*, abs/2110.01889.

[44] Breiman, L. (2001). Random forests. *Machine learning*, 45(1), 5–32.

[45] Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (2017). *Classification and regression trees*. Routledge.

[46] Breunig, M. M., Kriegel, H.-P., Ng, R. T., & Sander, J. (2000). Lof: identifying density-based local outliers. In *SIGMOD* (pp. 93–104).

[47] Brümmer, N., Cumani, S., Glembek, O., Karafiát, M., Matějka, P., Pešán, J., Plchot, O., Soufifar, M., Villiers, E. d., & Černockỳ, J. H. (2012). Description and analysis of the brno276 system for lre2011. In *Odyssey 2012-the speaker and language recognition workshop*.

[48] Buckman, J., Roy, A., Raffel, C., & Goodfellow, I. (2018). Thermometer encoding: One hot way to resist adversarial examples. In *International Conference on Learning Representations*.

[49] Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., et al. (2013). Api design for machine learning software: experiences from the scikit-learn project. *arXiv preprint arXiv:1309.0238*.

[50] Cai, H., Liu, J., & Yin, W. (2021). Learned robust pca: A scalable deep unfolding approach for high-dimensional outlier detection. *NeurIPS*, 34.

[51] Campos, G. O., Zimek, A., Sander, J., Campello, R. J., Micenková, B., Schubert, E., Assent, I., & Houle, M. E. (2016). On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study. *Data mining and knowledge discovery*, 30(4), 891–927.

[52] Cestnik, B., Kononenko, I., & Bratko, I. (1987). A knowledge-elicitation tool for sophisticated users. In *European Conference on European Working Session on Learning EWSL*, volume 87.

[53] Chan, R., Lis, K., Uhlemeyer, S., Blum, H., Honari, S., Siegwart, R., Fua, P., Salzmann, M., & Rottmann, M. (2021). Segmentmeifyoucan: A benchmark for anomaly segmentation. In *NeurIPS*.

[54] Chang, C.-H., Yoon, J., Arik, S., Udell, M., & Pfister, T. (2022). Data-efficient and interpretable tabular anomaly detection. *ArXiv preprint*, abs/2203.02034.

[55] Chen, J., Sathe, S., Aggarwal, C. C., & Turaga, D. S. (2017). Outlier detection with autoencoder ensembles. In N. V. Chawla & W. Wang (Eds.), *SDM* (pp. 90–98).: SIAM.

[56] Chen, T. & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In B. Krishnapuram, M. Shah, A. J. Smola, C. C. Aggarwal, D. Shen, & R. Rastogi (Eds.), *KDD* (pp. 785–794).: ACM.

[57] Cheng, L., Wang, Y., Liu, X., & Li, B. (2020). Outlier detection ensemble with embedded feature selection. In *AAAI* (pp. 3503–3512).: AAAI Press.

[58] Clei, C. L., Pushak, Y., Zogaj, F., Kareshk, M. O., Zohrevand, Z., Harlow, R., Moghadam, H. F., Hong, S., & Chafi, H. (2022). N-1 experts: Unsupervised anomaly detection model selection. In *Latebreaking Workshop on AutoML Conference*.

[59] Constantinou, V. (2018). Pynomaly: Anomaly detection using local outlier probabilities (LoOP). *The Journal of Open Source Software (JOSS)*, 3, 845.

[60] Dai, D., Dong, L., Ma, S., Zheng, B., Sui, Z., Chang, B., & Wei, F. (2022). Stablemoe: Stable routing strategy for mixture of experts. In *Proceedings of ACL* (pp. 7085–7095).

[61] Dai, S., Venkatesan, R., Ren, H., Zimmer, B., Dally, W. J., & Khailany, B. (2021). Vs-quant: Per-vector scaled quantization for accurate low-precision neural network inference. *CoRR*.

[62] Das, S., Wong, W.-K., Dietterich, T. G., Fern, A., & Emmott, A. (2016). Incorporating expert feedback into active anomaly discovery. In *ICDM* (pp. 853–858).: IEEE Computer Society.

[63] Davidson, I. & Ravi, S. S. (2020). A framework for determining the fairness of outlier detection. In *ECAI 2020* (pp. 2465–2472). IOS Press.

[64] Deecke, L., Ruff, L., Vandermeulen, R. A., & Bilen, H. (2021). Transfer-based semantic anomaly detection. In M. Meila & T. Zhang (Eds.), *ICML*, volume 139 of *PMLR* (pp. 2546–2558).: PMLR.

[65] Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *The JMLR*, 7, 1–30.

[66] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition* (pp. 248–255).: IEEE.

[67] Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* (pp. 4171–4186). Minneapolis, Minnesota: Association for Computational Linguistics.

[68] Diaconis, P. & Efron, B. (1983). Computer-intensive methods in statistics. *Scientific American*, 248(5).

[69] Dietterich, T., Jain, A., Lathrop, R., & Lozano-Perez, T. (1993). A comparison of dynamic reposing and tangent distance for drug activity prediction. *NeurIPS*, 6.

[70] Ding, K., Zhou, Q., Tong, H., & Liu, H. (2021). Few-shot network anomaly detection via cross-network meta-learning. In *WWW*.

[71] Ding, X., Zhao, L., & Akoglu, L. (2022). Hyperparameter sensitivity in deep outlier detection: Analysis and a scalable hyper-ensemble solution. In *NeurIPS*.

[72] Domingues, R., Filippone, M., Michiardi, P., & Zouaoui, J. (2018). A comparative evaluation of outlier detection algorithms: Experiments and analyses. *Pattern Recognition*, 74, 406–421.

[73] Du, N., Huang, Y., Dai, A. M., Tong, S., Lepikhin, D., Xu, Y., Krikun, M., Zhou, Y., Yu, A. W., Firat, O., et al. (2022). Glam: Efficient scaling of language models with mixture-of-experts. In *ICML* (pp. 5547–5569).: PMLR.

[74] Du, X., Zhang, J., Han, B., Liu, T., Rong, Y., Niu, G., Huang, J., & Sugiyama, M. (2021). Learning diverse-structured networks for adversarial robustness. In M. Meila & T. Zhang (Eds.), *ICML*, volume 139 of *PMLR* (pp. 2880–2891).: PMLR.

[75] Duan, S., Matthey, L., Saraiva, A., Watters, N., Burgess, C., Lerchner, A., & Higgins, I. (2020). Unsupervised model selection for variational disentangled representation learning. In *ICLR*: OpenReview.net.

[76] Emmott, A., Das, S., Dietterich, T., Fern, A., & Wong, W.-K. (2015). A meta-analysis of the anomaly detection problem. *ArXiv preprint*, abs/1503.01158.

[77] Evett, I. W. & Spiehler, E. J. (1987). Rule induction in forensic science. In *KBS in Government* (pp. 107–118).: Online Publications.

[78] Falkner, S., Klein, A., & Hutter, F. (2018). Bohb: Robust and efficient hyperparameter optimization at scale. In *ICML* (pp. 1437–1446).: PMLR.

[79] Fan, L., Liu, S., Chen, P.-Y., Zhang, G., & Gan, C. (2021). When does contrastive learning preserve adversarial robustness from pretraining to finetuning? *NeurIPS*, 34.

[80] Fan, X., Yue, Y., Sarkar, P., & Wang, Y. X. R. (2019). A unified framework for tuning hyperparameters in clustering problems. *ArXiv preprint*, abs/1910.08018.

[81] Feurer, M. & Hutter, F. (2019). Hyperparameter optimization. In *Automated machine learning* (pp. 3–33). Springer, Cham.

[82] Feurer, M., Letham, B., & Bakshy, E. (2018). Scalable meta-learning for bayesian optimization. *ArXiv preprint*, abs/1802.02219.

[83] Feurer, M., Springenberg, J., & Hutter, F. (2015). Initializing bayesian hyperparameter optimization via meta-learning. In *AAAI*, volume 29.

[84] Feurer, M., Springenberg, J. T., & Hutter, F. (2014). Using meta-learning to initialize bayesian optimization of hyperparameters. In *MetaSel@ ECAI*.

[85] Frey, P. W. & Slate, D. J. (1991). Letter recognition using holland-style adaptive classifiers. *Machine learning*, 6(2), 161–182.

[86] Fréry, J., Habrard, A., Sebban, M., Caelen, O., & He-Guelton, L. (2017). Efficient top rank optimization with gradient boosting for supervised anomaly detection. In *ECML/PKDD*, volume 10534 (pp. 20–35).

[87] Fu, C., Xiang, C., Wang, C., & Cai, D. (2019). Fast approximate nearest neighbor search with the navigating spreading-out graph. *VLDB*, 12(5), 461–474.

[88] Gao, Y., Liu, Y., Zhang, H., Li, Z., Zhu, Y., Lin, H., & Yang, M. (2020). Estimating GPU memory consumption of deep learning models. In *ESEC/FSE* (pp. 1342–1352).: ACM.

[89] Garcia-Teodoro, P., Diaz-Verdejo, J., Maciá-Fernández, G., & Vázquez, E. (2009). Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security*, 28(1-2), 18–28.

[90] Gera, P., Kim, H., Sao, P., Kim, H., & Bader, D. (2020). Traversing large graphs on gpus with unified memory. *VLDB*, 13(7), 1119–1133.

[91] Goix, N. (2016). How to evaluate the quality of unsupervised anomaly detection algorithms? *arXiv preprint arXiv:1607.01152*.

[92] Goldstein, M. & Dengel, A. (2012). Histogram-based outlier score (hbos): A fast unsupervised anomaly detection algorithm. *KI-2012: poster and demo track*, 9.

[93] Goldstein, M. & Uchida, S. (2016). A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PloS one*, 11(4), e0152173.

[94] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

[95] Goodge, A., Hooi, B., Ng, S.-K., & Ng, W. S. (2022). Lunar: Unifying local outlier detection methods via graph neural networks. In *AAAI*, volume 36 (pp. 6737–6745).

[96] Gopalan, P., Sharan, V., & Wieder, U. (2019). Pidforest: Anomaly detection via partial identification. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, & R. Garnett (Eds.), *NeurIPS* (pp. 15783–15793).

[97] Gorishniy, Y., Rubachev, I., Khrulkov, V., & Babenko, A. (2021). Revisiting deep learning models for tabular data. *NeurIPS*, 34.

[98] Grinsztajn, L., Oyallon, E., & Varoquaux, G. (2022). Why do tree-based models still outperform deep learning on tabular data? *ArXiv preprint*, abs/2207.08815.

[99] Grunau, C. & Rozhoň, V. (2020). Adapting k-means algorithms for outliers. *ArXiv preprint*, abs/2007.01118.

[100] Guan, M., Gulshan, V., Dai, A., & Hinton, G. (2018). Who said what: Modeling individual labelers improves classification. In *AAAI*, volume 32.

[101] Guha, S., Mishra, N., Roy, G., & Schrijvers, O. (2016). Robust random cut forest based anomaly detection on streams. In M. Balcan & K. Q. Weinberger (Eds.), *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings* (pp. 2712–2721).: JMLR.org.

[102] Guyon, I. & Elisseeff, A. (2003). An introduction to variable and feature selection. *JMLR*, 3(Mar), 1157–1182.

[103] Han, B., Yao, Q., Liu, T., Niu, G., Tsang, I. W., Kwok, J. T., & Sugiyama, M. (2020). A survey of label-noise representation learning: Past, present and future. *ArXiv preprint*, abs/2011.04406.

[104] Han, B., Yao, Q., Yu, X., Niu, G., Xu, M., Hu, W., Tsang, I., & Sugiyama, M. (2018). Co-teaching: Robust training of deep neural networks with extremely noisy labels. *NeurIPS*, 31.

[105] Han, S., Hu, X., Huang, H., Jiang, M., & Zhao, Y. (2022). Adbench: Anomaly detection benchmark. In *NeurIPS*.

[106] Hardin, J. & Rocke, D. M. (2004). Outlier detection in the multiple cluster setting using the minimum covariance determinant estimator. *Computational Statistics & Data Analysis*, 44(4), 625–638.

[107] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., et al. (2020). Array programming with numpy. *Nature*.

[108] Hastie, T., Tibshirani, R., Friedman, J. H., & Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer.

[109] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770–778).: IEEE.

[110] He, R. & McAuley, J. J. (2016). Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In J. Bourdeau, J. Hendler, R. Nkambou, I. Horrocks, & B. Y. Zhao (Eds.), *WWW* (pp. 507–517).: ACM.

[111] He, X., Zhao, K., & Chu, X. (2021). AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212, 106622.

[112] He, Z., Xu, X., & Deng, S. (2003). Discovering cluster-based local outliers. *Pattern recognition letters*, 24(9-10), 1641–1650.

[113] Hinton, G. E., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531.

[114] Horton, P. & Nakai, K. (1996). A probabilistic classification system for predicting the cellular localization sites of proteins. In *Ismb*, volume 4 (pp. 109–115).

[115] Hu, J., Wang, F., Sun, J., Sorrentino, R., & Ebadollahi, S. (2012). A healthcare utilization analysis framework for hot spotting and contextual anomaly detection. In *AMIA*: AMIA.

[116] Hu, X., Huang, Y., Li, B., & Lu, T. (2021). Uncovering the source of machine bias. *KDD 2021, Machine Learning for Consumers and Markets Workshop*.

[117] Huang, H., Qin, H., Yoo, S., & Yu, D. (2014). Physics-based anomaly detection defined on manifold space. *TKDD*, 9(2), 1–39.

[118] Huang, K., Fu, T., Gao, W., Zhao, Y., Roohani, Y., Leskovec, J., Coley, C., Xiao, C., Sun, J., & Zitnik, M. (2021). Therapeutics data commons: Machine learning datasets and tasks for drug discovery and development. *NeurIPS*.

[119] Huang, K., Fu, T., Gao, W., Zhao, Y., Roohani, Y., Leskovec, J., Coley, C. W., Xiao, C., Sun, J., & Zitnik, M. (2022). Artificial intelligence foundation for therapeutic science. *Nature Chemical Biology*, 18(10), 1033–1036.

[120] Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In C. A. C. Coello (Ed.), *LION*, volume 6683 of *Lecture Notes in Computer Science* (pp. 507–523).: Springer.

[121] Ismail Fawaz, H., Forestier, G., Weber, J., Idoumghar, L., & Muller, P.-A. (2019). Deep learning for time series classification: a review. *Data mining and knowledge discovery*, 33(4), 917–963.

[122] Jacobs, R. A., Jordan, M. I., Nowlan, S. J., & Hinton, G. E. (1991). Adaptive mixtures of local experts. *Neural computation*, 3, 79–87.

[123] Janssens, J., Huszár, F., Postma, E., & van den Herik, H. (2012). *Stochastic outlier selection*. Technical report, Technical report TiCC TR 2012-001, Tilburg University, Tilburg Center for Cognition and Communication, Tilburg, The Netherlands.

[124] Järvelin, K. & Kekäläinen, J. (2002). Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4), 422–446.

[125] Jerez, C. I., Zhang, J., & Silva, M. R. (2022). On equivalence of anomaly detection algorithms. *TKDD*.

[126] Ji, Z. & Wang, C. (2021). Accelerating DBSCAN algorithm with AI chips for large datasets. In X. Sun, S. Shende, L. V. Kalé, & Y. Chen (Eds.), *ICPP 2021: 50th International Conference on Parallel Processing, Lemont, IL, USA, August 9 - 12, 2021* (pp. 51:1–51:11).: ACM.

[127] Jia, Z., Lin, S., Gao, M., Zaharia, M., & Aiken, A. (2020). Improving the accuracy, scalability, and performance of graph neural networks with roc. In *MLSys*: mlsys.org.

[128] Jia, Z., Zaharia, M., & Aiken, A. (2019). Beyond data and model parallelism for deep neural networks. In A. Talwalkar, V. Smith, & M. Zaharia (Eds.), *Proceedings of Machine Learning and Systems 2019, MLSys 2019, Stanford, CA, USA, March 31 - April 2, 2019*: mlsys.org.

[129] Jiang, M., Hou, C., Zheng, A., Hu, X., Han, S., Huang, H., He, X., Yu, P. S., & Zhao, Y. (2023). Weakly supervised anomaly detection: A survey. *arXiv preprint arXiv:2302.04549*.

[130] Johnson, W. B. & Lindenstrauss, J. (1984). Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206), 1.

[131] Jones, D. R., Schonlau, M., & Welch, W. J. (1998). Efficient global optimization of expensive black-box functions. *J. Global Optimization*, 13(4), 455–492.

[132] Kadioglu, S., Malitsky, Y., Sellmann, M., & Tierney, K. (2010). Isac - instance-specific algorithm configuration. In *ECAI*, volume 215 of *Frontiers in Artificial Intelligence and Applications* (pp. 751–756).: IOS Press.

[133] Kahan, W. (1996). Ieee standard 754 for binary floating-point arithmetic. *Lecture Notes on the Status of IEEE*, 754(94720-1776), 11.

[134] Karmaker, S. K., Hassan, M. M., Smith, M. J., Xu, L., Zhai, C., & Veeramachaneni, K. (2021). Automl to date and beyond: Challenges and opportunities. *CSUR*, 54(8), 1–36.

[135] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T. (2017). Lightgbm: A highly efficient gradient boosting decision tree. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, & R. Garnett (Eds.), *NeurIPS* (pp. 3146–3154).

[136] Keller, F., Müller, E., & Böhm, K. (2012). Hics: High contrast subspaces for density-based outlier ranking. In A. Kementsietsidis & M. A. V. Salles (Eds.), *ICDE* (pp. 1037–1048).: IEEE Computer Society.

[137] Kim, M., Tack, J., & Hwang, S. J. (2020). Adversarial self-supervised contrastive learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.), *NeurIPS*.

[138] Kiran, B. R., Thomas, D. M., & Parakkal, R. (2018). An overview of deep learning based methods for unsupervised and semi-supervised anomaly detection in videos. *Journal of Imaging*, 4(2), 36.

[139] Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5), 604–632.

[140] Knorr, E. M., Ng, R. T., & Tucakov, V. (2000). Distance-based outliers: Algorithms and applications. *VLDB J.*, 8(3-4), 237–253.

[141] Komsta, L. & Komsta, M. L. (2011). *Package 'outliers'*. R package version 0.14.

[142] Koutsoukos, D., Nakandala, S., Karanasos, K., Saur, K., Alonso, G., & Interlandi, M. (2021). Tensors: An abstraction for general data processing. *VLDB*, 14(10), 1797–1804.

[143] Kriegel, H.-P., Kröger, P., Schubert, E., & Zimek, A. (2009). Outlier detection in axis-parallel subspaces of high dimensional data. In *PAKDD* (pp. 831–838).: Springer.

[144] Kriegel, H.-P., Kroger, P., Schubert, E., & Zimek, A. (2011). Interpreting and unifying outlier scores. In *SDM* (pp. 13–24).: SIAM.

[145] Kriegel, H.-P., Schubert, M., & Zimek, A. (2008). Angle-based outlier detection in high-dimensional data. In *KDD* (pp. 444–452).: ACM.

[146] Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. *Master's thesis, University of Tront*.

[147] Kudo, M., Toyama, J., & Shimbo, M. (1999). Multidimensional curve classification using passing-through regions. *Pattern Recognition Letters*, 20(11-13), 1103–1111.

[148] Kumar, S., Hooi, B., Makhija, D., Kumar, M., Faloutsos, C., & Subrahmanian, V. S. (2018). Rev2: Fraudulent user prediction in rating platforms. In *WSDM* (pp. 333–341).: ACM.

[149] Kwon, D., Natarajan, K., Suh, S. C., Kim, H., & Kim, J. (2018). An empirical study on network anomaly detection using convolutional neural networks. In *ICDCS* (pp. 1595–1598).

[150] Lai, K., Zha, D., Xu, J., Zhao, Y., Wang, G., & Hu, X. (2021a). Revisiting time series outlier detection: Definitions and benchmarks. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*: Joaquin Vanschoren and Sai-Kit Yeung.

[151] Lai, K.-H., Zha, D., Wang, G., Xu, J., Zhao, Y., Kumar, D., Chen, Y., Zumkhawaka, P., Wan, M., Martinez, D., et al. (2021b). Tods: An automated time series outlier detection system. In *AAAI* (pp. 16060–16062).

[152] Lamba, H. & Akoglu, L. (2019). Learning on-the-job to re-rank anomalies from top-1 feedback. In *SDM* (pp. 612–620).: SIAM.

[153] Lang, K. (1995). Newsweeder: Learning to filter netnews. In *ICML* (pp. 331–339). Elsevier.

[154] Lavin, A. & Ahmad, S. (2015). Evaluating real-time anomaly detection algorithms–the numenta anomaly benchmark. In *2015 IEEE 14th ICML and applications (ICMLA)* (pp. 38–44).: IEEE.

[155] Lazarevic, A., Ertoz, L., Kumar, V., Ozgur, A., & Srivastava, J. (2003). A comparative study of anomaly detection schemes in network intrusion detection. In *SDM* (pp. 25–36).: SIAM.

[156] Lazarevic, A. & Kumar, V. (2005). Feature bagging for outlier detection. In *KDD* (pp. 157–166).

[157] Leal, E. & Gruenwald, L. (2018). Research issues of outlier detection in trajectory streams using gpus. *SIGKDD Explor.*, 20(2), 13–20.

[158] LeCun, Y. (2019). 1.1 deep learning hardware: past, present, and future. In *2019 IEEE International Solid-State Circuits Conference-(ISSCC)* (pp. 12–19).: IEEE.

[159] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278–2324.

[160] Lee, H., Im, J., Jang, S., Cho, H., & Chung, S. (2019). Melu: Meta-learned user preference estimator for cold-start recommendation. In A. Teredesai, V. Kumar, Y. Li, R. Rosales, E. Terzi, & G. Karypis (Eds.), *KDD* (pp. 1073–1082).: ACM.

[161] Lee, M.-C., Shekhar, S., Faloutsos, C., Hutson, T. N., & Iasemidis, L. (2021a). Gen 2 out: Detecting and ranking generalized anomalies. In *Big Data* (pp. 801–811).: IEEE.

[162] Lee, M.-C., Zhao, Y., Wang, A., Liang, P. J., Akoglu, L., Tseng, V. S., & Faloutsos, C. (2020). Autoaudit: Mining accounting and time-evolving graphs. In *Big Data* (pp. 950–956).: IEEE.

[163] Lee, R., Zhou, M., Li, C., Hu, S., Teng, J., Li, D., & Zhang, X. (2021b). The art of balance: a rateupdb™ experience of building a cpu/gpu hybrid database product. *VLDB*, 14(12), 2999–3013.

[164] Lee, W., Sharma, R., & Aiken, A. (2018). On automatically proving the correctness of math.h implementations. *Proc. ACM Program. Lang.*, 2(POPL), 47:1–47:32.

[165] Lefohn, A. E., Sengupta, S., Kniss, J., Strzodka, R., & Owens, J. D. (2006). Glift: Generic, efficient, random-access gpu data structures. *ACM Transactions on Graphics (TOG)*, 25(1), 60–99.

[166] Leite, R., Brazdil, P., & Vanschoren, J. (2012). Selecting classification algorithms with active testing. In *MLDM* (pp. 117–131).: Springer.

[167] Li, G., Xie, Y., & Lin, L. (2018a). Weakly supervised salient object detection using image labels. In S. A. McIlraith & K. Q. Weinberger (Eds.), *AAAI* (pp. 7024–7031).: AAAI Press.

[168] Li, L., Jamieson, K. G., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2017). Hyperband: A novel bandit-based approach to hyperparameter optimization. *J. Mach. Learn. Res.*, 18, 185:1–185:52.

[169] Li, L., Littman, M. L., & Walsh, T. J. (2008). Knows what it knows: a framework for self-aware learning. In W. W. Cohen, A. McCallum, & S. T. Roweis (Eds.), *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*, volume 307 of *ACM International Conference Proceeding Series* (pp. 568–575).: ACM.

[170] Li, S., Liu, F., & Jiao, L. (2022a). Self-training multi-sequence learning with transformer for weakly supervised video anomaly detection. *AAAI*, 24.

[171] Li, W., Wang, Y., Cai, Y., Arnold, C. W., Zhao, E., & Yuan, Y. (2018b). Semi-supervised rare disease detection using generative adversarial network. *CoRR*.

[172] Li, Y., Chen, Z., Zha, D., Zhou, K., Jin, H., Chen, H., & Hu, X. (2021). Autood: Neural architecture search for outlier detection. In *ICDE* (pp. 2117–2122).: IEEE.

[173] Li, Y., Zha, D., Venugopal, P. K., Zou, N., & Hu, X. (2020a). Pyodds: An end-to-end outlier detection system with automated machine learning. In A. E. F. Seghrouchni, G. Sukthankar, T. Liu, & M. van Steen (Eds.), *www* (pp. 153–157).: ACM / IW3C2.

[174] Li, Z., Zhao, Y., Botta, N., Ionescu, C., & Hu, X. (2020b). Copod: copula-based outlier detection. In *ICDM* (pp. 1118–1123).: IEEE.

[175] Li, Z., Zhao, Y., Hu, X., Botta, N., Ionescu, C., & Chen, G. (2022b). Ecod: Unsupervised outlier detection using empirical cumulative distribution functions. *TKDE*, (pp. 1–1).

[176] Liang, J. & Parthasarathy, S. (2016). Robust contextual outlier detection: Where context meets sparsity. In *CIKM* (pp. 2167–2172).: ACM.

[177] Lin, Z., Thekumparampil, K., Fanti, G., & Oh, S. (2020). InfoGAN-CR and ModelCentrality: Self-supervised model training and selection for disentangling GANs. In *ICML* (pp. 6127–6139).

[178] Lioma, C., Simonsen, J. G., & Larsen, B. (2017). Evaluation measures for relevance and credibility in ranked lists. In *ICTIR* (pp. 91–98).: ACM.

[179] Liu, B., Tan, P., & Zhou, J. (2022a). Unsupervised anomaly detection by robust density estimation. In *AAAI* (pp. 4101–4108).: AAAI Press.

[180] Liu, F. T., Ting, K. M., & Zhou, Z.-H. (2008). Isolation forest. In *ICDM* (pp. 413–422).: IEEE.

[181] Liu, K., Dou, Y., Zhao, Y., Ding, X., Hu, X., Zhang, R., Ding, K., Chen, C., Peng, H., Shu, K., et al. (2022b). Pygod: A python library for graph outlier detection. *ArXiv preprint*, abs/2204.12095.

[182] Liu, K., Dou, Y., Zhao, Y., Ding, X., Hu, X., Zhang, R., Ding, K., Chen, C., Peng, H., Shu, K., Sun, L., Li, J., Chen, G. H., Jia, Z., & Yu, P. S. (2022c). BOND: Benchmarking unsupervised outlier node detection on static attributed graphs. In *NeurIPS*.

[183] Liu, S. & Hauskrecht, M. (2021). Event outlier detection in continuous time. In M. Meila & T. Zhang (Eds.), *ICML*, volume 139 of *PMLR* (pp. 6793–6803).: PMLR.

[184] Liu, Y., Li, Z., Zhou, C., Jiang, Y., Sun, J., Wang, M., & He, X. (2019). Generative adversarial active learning for unsupervised outlier detection. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*.

[185] Liu, Z., Dou, Y., Yu, P. S., Deng, Y., & Peng, H. (2020). Alleviating the inconsistency problem of applying graph neural network to fraud detection. In J. Huang, Y. Chang, X. Cheng, J. Kamps, V. Murdock, J. Wen, & Y. Liu (Eds.), *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020* (pp. 1569–1572).: ACM.

[186] Loh, W.-Y. (2011). Classification and regression trees. *WIREs Data Mining and Knowledge Discovery*, 1.

[187] Looks, M., Herreshoff, M., Hutchins, D., & Norvig, P. (2017). Deep learning with dynamic computation graphs. In *ICLR*: OpenReview.net.

[188] Lozano, E. & Acuña, E. (2005). Parallel algorithms for distance-based and density-based outliers. In *ICDM* (pp. 729–732).: IEEE Computer Society.

[189] Lu, W., Cheng, Y., Xiao, C., Chang, S., Huang, S., Liang, B., & Huang, T. S. (2017). Unsupervised sequential outlier detection with deep architectures. *IEEE Trans. Image Process.*, 26(9), 4321–4330.

[190] Ma, M. Q., Zhao, Y., Zhang, X., & Akoglu, L. (2023). The need for unsupervised outlier model selection: A review and evaluation of internal evaluation strategies. *ACM SIGKDD Explorations Newsletter*, 25(1).

[191] Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., & Potts, C. (2011). Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies* (pp. 142–150). Portland, Oregon, USA: Association for Computational Linguistics.

[192] Malerba, D., Esposito, F., & Semeraro, G. (1996). A further comparison of simplification methods for decision-tree induction. In *Learning from data* (pp. 365–374). Springer.

[193] Mangasarian, O. L., Street, W. N., & Wolberg, W. H. (1995). Breast cancer diagnosis and prognosis via linear programming. *Operations Research*, 43(4), 570–577.

[194] Manolache, A., Brad, F., & Burceanu, E. (2021). DATE: Detecting anomalies in text via self-supervision of transformers. In *NAACL* (pp. 267–277). Online: Association for Computational Linguistics.

[195] Manzoor, E. A., Lamba, H., & Akoglu, L. (2018). xstream: Outlier detection in feature-evolving data streams. In *KDD* (pp. 1963–1972).: ACM.

[196] Marques, H. O., Campello, R. J., Sander, J., & Zimek, A. (2020). Internal evaluation of unsupervised outlier detection. *TKDD*, 14.

[197] Marques, H. O., Campello, R. J. G. B., Zimek, A., & Sander, J. (2015). On the internal evaluation of unsupervised outlier detection. In *SSDBM* (pp. 7:1–7:12).: ACM.

[198] Martinez-Guerra, R. & Mata-Machuca, J. L. (2016). *Fault detection and diagnosis in nonlinear systems*. Springer.

[199] Meghanath, M. M., Pai, D., & Akoglu, L. (2018). Conout: Contextual outlier detection with multiple contexts: Application to ad fraud. In *ECML/PKDD*, volume 11051 (pp. 139–156).: Springer.

[200] Meng, C., Sun, M., Yang, J., Qiu, M., & Gu, Y. (2017). Training deeper models by gpu memory optimization on tensorflow. In *Proc. of ML Systems Workshop in NIPS*: NIPS.

[201] Menon, P., Mowry, T. C., & Pavlo, A. (2017). Relaxed operator fusion for in-memory databases: Making compilation, vectorization, and prefetching work together at last. *VLDB*, 11(1), 1–13.

[202] Milligan, G. W. (1985). An algorithm for generating artificial test clusters. *Psychometrika*, 50(1), 123–127.

[203] Min, S. W., Mailthody, V. S., Qureshi, Z., Xiong, J., Ebrahimi, E., & Hwu, W.-m. (2020). Emogi: efficient memory-access for out-of-memory graph-traversal in gpus. *VLDB*, 14(2), 114–127.

[204] Min, S. W., Wu, K., Huang, S., Hidayetoğlu, M., Xiong, J., Ebrahimi, E., Chen, D., & Hwu, W. M. (2021). Large graph convolutional network training with gpu-oriented data communication architecture. *VLDB*, 14(11), 2087–2100.

[205] Mishra, A., Pudipeddi, J. S., & Akoglu, L. (2017). Ranking in heterogeneous networks with geo-location information. In *SDM* (pp. 408–416).: SIAM.

[206] Misir, M. & Sebag, M. (2017). Alors: An algorithm recommender system. *Artif. Intell.*, 244, 291–314.

[207] Moustafa, N. & Slay, J. (2015). Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *2015 military communications and information systems conference (MilCIS)* (pp. 1–6).: IEEE.

[208] Mu, N. & Gilmer, J. (2019). Mnist-c: A robustness benchmark for computer vision. *ArXiv preprint*, abs/1906.02337.

[209] Nakandala, S., Saur, K., Yu, G., Karanasos, K., Curino, C., Weimer, M., & Interlandi, M. (2020). A tensor compiler for unified machine learning prediction serving. In *OSDI* (pp. 899–917).: OSDI.

[210] Narayanan, D., Harlap, A., Phanishayee, A., Seshadri, V., Devanur, N. R., Ganger, G. R., Gibbons, P. B., & Zaharia, M. (2019). Pipedream: Generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP '19 (pp. 1–15). New York, NY, USA: Association for Computing Machinery.

[211] Neeb, H. & Kurrus, C. (2016). Distributed k-nearest neighbors.

[212] Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., & Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning. *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*.

[213] Neubig, G., Goldberg, Y., & Dyer, C. (2017). On-the-fly operation batching in dynamic computation graphs. In *NeurIPS* (pp. 3971–3981).

[214] Nguyen, D. T., Mummadi, C. K., Ngo, T., Nguyen, T. H. P., Beggel, L., & Brox, T. (2020). SELF: learning to filter noisy labels with self-ensembling. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*: OpenReview.net.

[215] Nguyen, T. T., Nguyen, A. T., Nguyen, T. A. H., Vu, L. T., Nguyen, Q. U., & Hai, L. D. (2015). Unsupervised anomaly detection in online game. In *Proceedings of the Sixth International Symposium on Information and Communication Technology* (pp. 4–10).: ACM.

[216] Nguyen, V., Nguyen, T., & Nguyen, U. (2017). An evaluation method for unsupervised anomaly detection algorithms. *Journal of Computer Science and Cybernetics*, 32(3), 259–272.

[217] Nikolic, M., Maric, F., & Janicic, P. (2013). Simple algorithm portfolio for sat. *Artif. Intell. Rev.*, 40(4), 457–465.

[218] Oku, J., Tamura, K., & Kitakami, H. (2014). Parallel processing for distance-based outlier detection on a multi-core cpu. In *IEEE International Workshop on Computational Intelligence and Applications (IWCIA)* (pp. 65–70).: IEEE.

[219] Orair, G. H., Teixeira, C. H., Meira Jr, W., Wang, Y., & Parthasarathy, S. (2010). Distance-based outlier detection: consolidation and renewed bearing. *VLDB*, 3(1-2), 1469–1480.

[220] Otey, M. E., Ghoting, A., & Parthasarathy, S. (2006). Fast distributed outlier detection in mixed-attribute data sets. *Data Min. Knowl. Discov.*, 12(2-3), 203–228.

[221] Ousterhout, K., Wendell, P., Zaharia, M., & Stoica, I. (2013). Sparrow: distributed, low latency scheduling. In M. Kaminsky & M. Dahlin (Eds.), *ACM SIGOPS 24th Symposium on Operating Systems Principles, SOSP '13, Farmington, PA, USA, November 3-6, 2013* (pp. 69–84).: ACM.

[222] Pang, G. & Aggarwal, C. (2021). Toward explainable deep anomaly detection. In *KDD* (pp. 4056–4057).

[223] Pang, G., Cao, L., & Chen, L. (2021a). Homophily outlier detection in non-iid categorical data. *Data Mining and Knowledge Discovery*, 35(4), 1163–1224.

[224] Pang, G., Cao, L., Chen, L., & Liu, H. (2016). Unsupervised feature selection for outlier detection by modelling hierarchical value-feature couplings. In *ICDM* (pp. 410–419).: IEEE.

[225] Pang, G., Cao, L., Chen, L., & Liu, H. (2017). Learning homophily couplings from non-iid data for joint feature selection and noise-resilient outlier detection. In C. Sierra (Ed.), *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017* (pp. 2585–2591).: ijcai.org.

[226] Pang, G., Cao, L., Chen, L., & Liu, H. (2018). Learning representations of ultrahigh-dimensional data for random distance-based outlier detection. In Y. Guo & F. Farooq (Eds.), *KDD* (pp. 2041–2050).: ACM.

[227] Pang, G., Ding, C., Shen, C., & Hengel, A. v. d. (2021b). Explainable deep few-shot anomaly detection with deviation networks. *ArXiv preprint*, abs/2108.00462.

[228] Pang, G., Shen, C., Cao, L., & Hengel, A. V. D. (2021c). Deep learning for anomaly detection: A review. *ACM Computing Surveys (CSUR)*, 54(2), 1–38.

[229] Pang, G., Shen, C., Jin, H., & Hengel, A. v. d. (2019a). Deep weakly-supervised anomaly detection. *ArXiv preprint*, abs/1910.13601.

[230] Pang, G., Shen, C., & van den Hengel, A. (2019b). Deep anomaly detection with deviation networks. In A. Teredesai, V. Kumar, Y. Li, R. Rosales, E. Terzi, & G. Karypis (Eds.), *KDD* (pp. 353–362).: ACM.

[231] Papadimitriou, S., Kitagawa, H., Gibbons, P. B., & Faloutsos, C. (2003). LOCI: fast outlier detection using the local correlation integral. In *ICDE* (pp. 315–326).: IEEE Computer Society.

[232] Paparrizos, J., Kang, Y., Boniol, P., Tsay, R. S., Palpanas, T., & Franklin, M. J. (2022). Tsb-uad: an end-to-end benchmark suite for univariate time-series anomaly detection. *VLDB*, 15(8), 1697–1711.

[233] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *NeuIPS*, 32, 8026–8037.

[234] Patrini, G., Rozza, A., Krishna Menon, A., Nock, R., & Qu, L. (2017). Making deep neural networks robust to label noise: A loss correction approach. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1944–1952).

[235] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *the JMLR*, 12, 2825–2830.

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

[237] Prokhorenkova, L. O., Gusev, G., Vorobev, A., Dorogush, A. V., & Gulin, A. (2018). Catboost: unbiased boosting with categorical features. In S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.), *NeurIPS* (pp. 6639–6649).

[238] Qiu, C., Li, A., Kloft, M., Rudolph, M., & Mandt, S. (2022). Latent outlier exposure for anomaly detection with contaminated data. *ArXiv preprint*, abs/2202.08088.

[239] Qiu, C., Pfrommer, T., Kloft, M., Mandt, S., & Rudolph, M. (2021). Neural transformation learning for deep anomaly detection beyond images. In M. Meila & T. Zhang (Eds.), *ICML*, volume 139 of *PMLR* (pp. 8703–8714).: PMLR.

[240] Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, 1(1), 81–106.

[241] Quinlan, J. R., Compton, P. J., Horn, K., & Lazarus, L. (1987). Inductive knowledge acquisition: a case study. In *Australian Conference on Applications of expert systems* (pp. 137–156).

[242] Rahman, M. M., Balakrishnan, D., Murthy, D., Kutlu, M., & Lease, M. (2021). An information retrieval approach to building datasets for hate speech detection. In *NeurIPS*.

[243] Ramaswamy, S., Rastogi, R., & Shim, K. (2000). Efficient algorithms for mining outliers from large data sets. In *SIGMOD* (pp. 427–438).

[244] Raschka, S. (2015). *Python machine learning*. Packt publishing ltd.

[245] Rayana, S. (2016). ODDS library.

[246] Rayana, S. & Akoglu, L. (2016). Less is more: Building selective anomaly ensembles. *ACM Trans. Knowl. Discov. Data*, 10(4), 42:1–42:33.

[247] Rebjock, Q., Kurt, B., Januschowski, T., & Callot, L. (2021). Online false discovery rate control for anomaly detection in time series. *NeurIPS*, 34.

[248] Riquelme, C., Puigcerver, J., Mustafa, B., Neumann, M., Jenatton, R., Susano Pinto, A., Keysers, D., & Houlsby, N. (2021). Scaling vision with sparse mixture of experts. *NeurIPS*, 34, 8583–8595.

[249] Ristoski, P., Bizer, C., & Paulheim, H. (2015). Mining the web of linked data with rapidminer. *J. Web Semant.*, 35, 142–151.

[250] Rodrigues, F. & Pereira, F. (2018). Deep learning from crowds. In *AAAI*, volume 32.

[251] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6), 386.

[252] Rousseeuw, P. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20(1), 53–65.

[253] Ruff, L., Görnitz, N., Deecke, L., Siddiqui, S. A., Vandermeulen, R. A., Binder, A., Müller, E., & Kloft, M. (2018). Deep one-class classification. In J. G. Dy & A. Krause (Eds.), *ICML*, volume 80 of *PMLR* (pp. 4390–4399).: PMLR.

[254] Ruff, L., Kauffmann, J. R., Vandermeulen, R. A., Montavon, G., Samek, W., Kloft, M., Dietterich, T. G., & Müller, K.-R. (2021). A unifying review of deep and shallow anomaly detection. *Proceedings of the IEEE*.

[255] Ruff, L., Vandermeulen, R. A., Görnitz, N., Binder, A., Müller, E., Müller, K., & Kloft, M. (2020). Deep semi-supervised anomaly detection. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*: OpenReview.net.

[256] Sakurada, M. & Yairi, T. (2014). Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Pacific Rim International Conference on Artificial Intelligence (PRICAI), Workshop on Machine Learning for Sensory Data Analysis (MLSDA)*.

[257] Schirrmeister, R., Zhou, Y., Ball, T., & Zhang, D. (2020). Understanding anomaly detection with deep invertible networks through hierarchies of distributions and features. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.), *NeurIPS*.

[258] Schölkopf, B., Platt, J. C., Shawe-Taylor, J., Smola, A. J., & Williamson, R. C. (2001). Estimating the support of a high-dimensional distribution. *Neural Comput.*, 13(7), 1443–1471.

[259] Schölkopf, B., Williamson, R. C., Smola, A. J., Shawe-Taylor, J., Platt, J. C., et al. (1999). Support vector method for novelty detection. In *NIPS*, volume 12 (pp. 582–588).: Citeseer.

[260] Schubert, E., Zimek, A., & Kriegel, H. (2014). Generalized outlier detection with flexible kernel density estimates. In *SDM* (pp. 542–550).: SIAM.

[261] Schubert, E., Zimek, A., & Kriegel, H. (2015). Fast and scalable outlier detection with approximate nearest neighbor ensembles. In M. Renz, C. Shahabi, X. Zhou, & M. A. Cheema (Eds.), *Database Systems for Advanced Applications - 20th International Conference, DASFAA 2015, Hanoi, Vietnam, April 20-23, 2015, Proceedings, Part II*, volume 9050 of *Lecture Notes in Computer Science* (pp. 19–36).: Springer.

[262] Schölkopf, B., Platt, J. C., Shawe-Taylor, J., Smola, A. J., & Williamson, R. C. (2001). Estimating the support of a high-dimensional distribution. *Neural Computation*, 13(7), 1443–1471.

[263] Sehwag, V., Chiang, M., & Mittal, P. (2021). SSD: A unified framework for self-supervised outlier detection. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*: OpenReview.net.

[264] Shafer, G. & Vovk, V. (2008). A tutorial on conformal prediction. *J. Mach. Learn. Res.*, 9, 371–421.

[265] Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., & de Freitas, N. (2016). Taking the human out of the loop: A review of bayesian optimization. *Proc. IEEE*, 104(1), 148–175.

[266] Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., & Dean, J. (2017). Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *ICLR*.

[267] Shekhar, S., Shah, N., & Akoglu, L. (2021). Fairod: Fairness-aware outlier detection. In *AIES* (pp. 210–220).

[268] Shen, L., Li, Z., & Kwok, J. T. (2020). Timeseries anomaly detection using temporal hierarchical one-class network. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.), *NeurIPS*.

[269] Shenkar, T. & Wolf, L. (2021). Anomaly detection for tabular data with internal contrastive learning. In *International Conference on Learning Representations*.

[270] Shieh, G. S. (1998). A weighted kendall's tau statistic. *Statistics & probability letters*, 39(1), 17–24.

[271] Shin, H. J., Eom, D.-H., & Kim, S.-S. (2005). One-class support vector machines—an application in machine fault detection and classification. *Computers & Industrial Engineering*, 48(2), 395–408.

[272] Shyu, M.-L., Chen, S.-C., Sarinnapakorn, K., & Chang, L. (2003). *A novel anomaly detection scheme based on principal component classifier*. Technical report, Miami Univ Coral Gables Fl Dept of Electrical and Computer Engineering.

[273] Sigillito, V. G., Wing, S. P., Hutton, L. V., & Baker, K. B. (1989). Classification of radar returns from the ionosphere using neural networks. *Johns Hopkins APL Technical Digest*, 10(3), 262–266.

[274] Sincraian, P. (2021). PyOD download statistics. https://pepy.tech/project/pyod. Accessed: 2021-09-09.

[275] Smith, L. N. (2017). Cyclical learning rates for training neural networks. In *WACV* (pp. 464–472).: IEEE Computer Society.

[276] Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *NIPS* (pp. 2960–2968).

[277] Soenen, J., Van Wolputte, E., Perini, L., Vercruyssen, V., Meert, W., Davis, J., & Blockeel, H. (2021). The effect of hyperparameter tuning on the comparative evaluation of unsupervised

anomaly detection methods. In *Proceedings of the KDD'21 Workshop on Outlier Detection and Description* (pp. 1–9).

[278] Song, H., Kim, M., Park, D., Shin, Y., & Lee, J.-G. (2022). Learning from noisy labels with deep neural networks: A survey. *IEEE Transactions on Neural Networks and Learning Systems*.

[279] Song, H., Li, P., & Liu, H. (2021). Deep clustering based fair outlier detection. In *KDD* (pp. 1481–1489).

[280] Spearman, C. (1904). The proof and measurement of association between two things. *The American Journal of Psychology*, 15(1), 72–101.

[281] Steinbuss, G. & Böhm, K. (2021). Benchmarking unsupervised outlier detection with realistic synthetic data. *TKDD*, 15(4).

[282] Stern, D. H., Samulowitz, H., Herbrich, R., Graepel, T., Pulina, L., & Tacchella, A. (2010). Collaborative expert portfolio management. In M. Fox & D. Poole (Eds.), *AAAI*: AAAI Press.

[283] Svedin, M., Chien, S. W. D., Chikafa, G., Jansson, N., & Podobas, A. (2021). Benchmarking the nvidia GPU lineage: From early K80 to modern A100 with asynchronous memory transfers. In *HEART '21* (pp. 9:1–9:6).: ACM.

[284] Tan, S. C., Ting, K. M., & Liu, F. T. (2011). Fast anomaly detection for streaming data. In *IJCAI* (pp. 1511–1516).: IJCAI/AAAI.

[285] Tang, J., Chen, Z., Fu, A. W.-C., & Cheung, D. W. (2002). Enhancing effectiveness of outlier detections for low density patterns. In *PAKDD* (pp. 535–548).: Springer.

[286] Thornton, C., Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2013). Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *KDD* (pp. 847–855).

[287] Tian, B., Su, Q., & Yin, J. (2022). Anomaly detection by leveraging incomplete anomalous knowledge with anomaly-aware bidirectional gans. *ArXiv preprint*, abs/2204.13335.

[288] Toliopoulos, T., Bellas, C., Gounaris, A., & Papadopoulos, A. (2020). PROUD: parallel outlier detection for streams. In D. Maier, R. Pottinger, A. Doan, W. Tan, A. Alawini, & H. Q. Ngo (Eds.), *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020* (pp. 2717–2720).: ACM.

[289] Unger, C., Jia, Z., Wu, W., Lin, S., Baines, M., Narvaez, C. E. Q., Ramakrishnaiah, V., Prajapati, N., McCormick, P., Mohd-Yusof, J., Luo, X., Mudigere, D., Park, J., Smelyanskiy, M.,

& Aiken, A. (2022). Unity: Accelerating DNN training through joint optimization of algebraic transformations and parallelization. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)* (pp. 267–284). Carlsbad, CA: USENIX Association.

[290] Vaithyanathan, S. & Dom, B. (1999). Generalized model selection for unsupervised learning in high dimensions. In *NIPS* (pp. 970–976).: The MIT Press.

[291] Van der Maaten, L. & Hinton, G. (2008). Visualizing data using t-sne. *JMLR*, 9(11).

[292] Vanschoren, J. (2018). Meta-learning: A survey. *arXiv preprint arXiv:1810.03548*.

[293] Vanschoren, J. (2019). Meta-learning. In *Automated Machine Learning* (pp. 35–61). Springer.

[294] Vargaftik, S., Keslassy, I., Orda, A., & Ben-Itzhak, Y. (2021). Rade: Resource-efficient supervised anomaly detection using decision tree-based ensemble methods. *Machine Learning*, 110(10), 2835–2866.

[295] Vartak, M., Thiagarajan, A., Miranda, C., Bratman, J., & Larochelle, H. (2017). A meta-learning perspective on cold-start recommendations for items. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, & R. Garnett (Eds.), *NeurIPS* (pp. 6904–6914).

[296] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, & R. Garnett (Eds.), *NeurIPS* (pp. 5998–6008).

[297] Venkatasubramanian, S. & Wang, Q. (2011). The johnson-lindenstrauss transform: An empirical study. In M. Müller-Hannemann & R. F. F. Werneck (Eds.), *Proceedings of the Thirteenth Workshop on Algorithm Engineering and Experiments, ALENEX 2011, Holiday Inn San Francisco Golden Gateway, San Francisco, California, USA, January 22, 2011* (pp. 164–173).: SIAM.

[298] Wang, G., Xie, S., Liu, B., & Yu, P. S. (2011). Review graph based online store review spammer detection. In *ICDM* (pp. 1242–1247).: IEEE Computer Society.

[299] Wang, H., Zhai, J., Gao, M., Ma, Z., Tang, S., Zheng, L., Li, Y., Rong, K., Chen, Y., & Jia, Z. (2021a). PET: optimizing tensor programs with partially equivalent transformations and automated corrections. In *OSDI* (pp. 37–54).: OSDI.

[300] Wang, M., Xu, X., Yue, Q., & Wang, Y. (2021b). A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *Proc. VLDB Endow.*, 14(11), 1964–1978.

[301] Wang, R. & Deng, D. (2020). Deltapq: lossless product quantization code compression for high dimensional similarity search. *VLDB*, 13(13), 3603–3616.

[302] Wang, R., Nie, K., Wang, T., Yang, Y., & Long, B. (2020a). Deep learning for anomaly detection. In *WSDM* (pp. 894–896).: ACM.

[303] Wang, S. & Ferhatosmanoglu, H. (2020). Ppq-trajectory: spatio-temporal quantization for querying in large trajectory repositories. *VLDB*, 14(2), 215–227.

[304] Wang, X., Wu, J., Zhang, D., Su, Y., & Wang, W. Y. (2019a). Learning to compose topic-aware mixture of experts for zero-shot video captioning. In *AAAI*, volume 33 (pp. 8965–8972).

[305] Wang, Z., Dai, B., Wipf, D. P., & Zhu, J. (2020b). Further analysis of outlier detection with deep generative models. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.), *NeurIPS*.

[306] Wang, Z., Li, H., Li, Z., Sun, X., Rao, J., Che, H., & Jiang, H. (2019b). Pigeon: an effective distributed, hierarchical datacenter job scheduler. In *Proceedings of the ACM Symposium on Cloud Computing, SoCC 2019, Santa Cruz, CA, USA, November 20-23, 2019* (pp. 246–258).: ACM.

[307] Wei, H., Feng, L., Chen, X., & An, B. (2020). Combating noisy labels by agreement: A joint training method with co-regularization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 13726–13735).

[308] Wei, H., Xie, R., Feng, L., Han, B., & An, B. (2022). Deep learning from multiple noisy annotators as a union. *IEEE Transactions on Neural Networks and Learning Systems*.

[309] Wenzel, F., Snoek, J., Tran, D., & Jenatton, R. (2020). Hyperparameter ensembles for robustness and uncertainty quantification. *NeurIPS*, 33, 6514–6527.

[310] Williams, C. & Rasmussen, C. (1995). Gaussian processes for regression. *NeurIPS*, 8.

[311] Wistuba, M., Schilling, N., & Schmidt-Thieme, L. (2015a). Hyperparameter search space pruning–a new component for sequential model-based hyperparameter optimization. In *PKDD* (pp. 104–119).: Springer.

[312] Wistuba, M., Schilling, N., & Schmidt-Thieme, L. (2015b). Learning hyperparameter optimization initializations. In *DSAA* (pp. 1–10).: IEEE.

[313] Wistuba, M., Schilling, N., & Schmidt-Thieme, L. (2016a). Hyperparameter optimization machines. In *DSAA* (pp. 41–50).: IEEE.

[314] Wistuba, M., Schilling, N., & Schmidt-Thieme, L. (2016b). Two-stage transfer surrogate model for automatic hyperparameter optimization. In *ECML/PKDD* (pp. 199–214).: Springer.

[315] Wistuba, M., Schilling, N., & Schmidt-Thieme, L. (2018a). Scalable gaussian process-based transfer surrogates for hyperparameter optimization. *Mach. Learn.*, 107(1), 43–78.

[316] Wistuba, M., Schilling, N., & Schmidt-Thieme, L. (2018b). Scalable gaussian process-based transfer surrogates for hyperparameter optimization. *Machine Learning*, 107(1).

[317] Wolberg, W. H. & Mangasarian, O. L. (1990). Multisurface method of pattern separation for medical diagnosis applied to breast cytology. *Proceedings of the national academy of sciences*, 87(23), 9193–9196.

[318] Wolpert, D. H. & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Trans. Evolutionary Computation*, 1(1), 67–82.

[319] Woods, K. S., Solka, J. L., Priebe, C. E., Kegelmeyer Jr, W. P., Doss, C. C., & Bowyer, K. W. (1994). Comparative evaluation of pattern recognition techniques for detection of microcalcifications in mammography. In *State of The Art in Digital Mammographic Image Analysis* (pp. 213–231). World Scientific.

[320] Wu, B., Chen, J., Cai, D., He, X., & Gu, Q. (2021). Do wider neural networks really help adversarial robustness? *NeurIPS*, 34.

[321] Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *ArXiv preprint*, abs/1708.07747.

[322] Xiao, Z., Yan, Q., & Amit, Y. (2021). Do we really need to learn representations from in-domain data for outlier detection? *ArXiv preprint*, abs/2105.09270.

[323] Xie, X. L. & Beni, G. (1991). A validity measure for fuzzy clustering. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(8), 841–847.

[324] Xu, H., Wang, Y., Jian, S., Huang, Z., Wang, Y., Liu, N., & Li, F. (2021a). Beyond outlier detection: Outlier interpretation by attention-guided triplet deviation network. In *WWW* (pp. 1328–1339).

[325] Xu, L., Hutter, F., Shen, J., Hoos, H. H., & Leyton-Brown, K. (2012). Satzilla2012: Improved algorithm selection based on cost-sensitive classification models. *Proceedings of SAT Challenge*, (pp. 57–58).

[326] Xu, Y., Ding, J., Zhang, L., & Zhou, S. (2021b). Dp-ssl: Towards robust semi-supervised learning with a few labeled samples. *NeurIPS*, 34.

[327] Yan, Y., Cao, L., Kulhman, C., & Rundensteiner, E. (2017). Distributed local outlier detection in big data. In *KDD* (pp. 1225–1234).: ACM.

[328] Yang, C., Akimoto, Y., Kim, D. W., & Udell, M. (2019). OBOE: collaborative filtering for automl model selection. In A. Teredesai, V. Kumar, Y. Li, R. Rosales, E. Terzi, & G. Karypis (Eds.), *KDD* (pp. 1173–1183).: ACM.

[329] Yang, J., Zhou, K., Li, Y., & Liu, Z. (2021). Generalized out-of-distribution detection: A survey. *ArXiv preprint*, abs/2110.11334.

[330] Yang, Z., Bozchalooi, I. S., & Darve, E. (2020). Anomaly detection with domain adaptation. *ArXiv preprint*, abs/2006.03689.

[331] Yin, X., Han, J., & Yu, P. S. (2007). Truth discovery with multiple conflicting information providers on the web. In *KDD* (pp. 1048–1052).: ACM.

[332] Yogatama, D. & Mann, G. (2014). Efficient transfer learning method for automatic hyperparameter tuning. In *AISTATS* (pp. 1077–1085).

[333] Yu, W., Li, J., Bhuiyan, M. Z. A., Zhang, R., & Huai, J. (2017). Ring: Real-time emerging anomaly monitoring system over text streams. *IEEE Transactions on Big Data*, 5(4), 506–519.

[334] Yu, X., Han, B., Yao, J., Niu, G., Tsang, I., & Sugiyama, M. (2019). How does disagreement help generalization against label corruption? In *ICML* (pp. 7164–7173).: PMLR.

[335] Zenati, H., Romain, M., Foo, C.-S., Lecouat, B., & Chandrasekhar, V. (2018). Adversarially learned anomaly detection. In *ICDM* (pp. 727–736).: IEEE.

[336] Zha, D., Lai, K.-H., Wan, M., & Hu, X. (2020). Meta-aad: Active anomaly detection with deep reinforcement learning. In *ICDM* (pp. 771–780).: IEEE.

[337] Zhang, H. & Davidson, I. (2021). Towards fair deep anomaly detection. In *FAccT* (pp. 138–148).

[338] Zhang, S., Ursekar, V., & Akoglu, L. (2022). Sparx: Distributed outlier detection at scale. *KDD*.

[339] Zhang, X., Zhao, J. J., & LeCun, Y. (2015). Character-level convolutional networks for text classification. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, & R. Garnett (Eds.), *NeurIPS* (pp. 649–657).

[340] Zhao, J., Liu, X., Yan, Q., Li, B., Shao, M., & Peng, H. (2020). Multi-attributed heterogeneous graph convolutional network for bot detection. *Information Sciences*, 537, 380–393.

[341] Zhao, Y. (2021). PyOD citation statistics. https://scholar.google.ca/scholar?cites=3726241381117726876&as_sdt=5,39&sciodt=0,39&hl=en. Accessed: 2021-09-09.

[342] Zhao, Y. & Akoglu, L. (2022). Hyperparameter optimization for unsupervised outlier detection. *arXiv preprint arXiv:2208.11727*.

[343] Zhao, Y., Chen, G. H., & Jia, Z. (2023a). Tod: Gpu-accelerated outlier detection via tensor operations. *Proceedings of the VLDB Endowment*, 16(3).

[344] Zhao, Y. & Hryniewicki, M. K. (2018). Xgbod: improving supervised outlier detection with unsupervised representation learning. In *IJCNN* (pp. 1–8).: IEEE.

[345] Zhao, Y., Hu, X., Cheng, C., Wang, C., Wan, C., Wang, W., Yang, J., Bai, H., Li, Z., Xiao, C., et al. (2021a). Suod: Accelerating large-scale unsupervised heterogeneous outlier detection. *MLSys*, 3, 463–478.

[346] Zhao, Y., Nasrullah, Z., Hryniewicki, M. K., & Li, Z. (2019a). LSCP: locally selective combination in parallel outlier ensembles. In T. Y. Berger-Wolf & N. V. Chawla (Eds.), *SDM* (pp. 585–593).: SIAM.

[347] Zhao, Y., Nasrullah, Z., & Li, Z. (2019b). Pyod: A python toolbox for scalable outlier detection. *JMLR*, 20, 1–7.

[348] Zhao, Y., Rossi, R., & Akoglu, L. (2021b). Automatic unsupervised outlier model selection. In *NeurIPS*, volume 34.

[349] Zhao, Y., Zhang, S., & Akoglu, L. (2022). Toward unsupervised outlier model selection. In *ICDM*: IEEE.

[350] Zhao, Y., Zheng, G., Mukherjee, S., McCann, R., & Awadallah, A. (2023b). ADMoE: Anomaly detection with mixture-of-experts from noisy labels. In *AAAI*.

[351] Zheng, G., Awadallah, A. H., & Dumais, S. (2021). Meta label correction for noisy label learning. *AAAI*.

[352] Zheng, Y., Wang, X., Qi, Y., Li, W., & Wu, L. (2022). Benchmarking unsupervised anomaly detection and localization. *ArXiv preprint*, abs/2205.14852.

[353] Zheng, Z., Yang, J., & Zhu, Y. (2007). Initialization enhancer for non-negative matrix factorization. *Eng. Appl. Artif. Intell.*, 20(1), 101–110.

[354] Zheng, Z., Yuan, C., Zhu, X., Lin, Z., Cheng, Y., Shi, C., & Ye, J. (2019). Self-supervised mixture-of-experts by uncertainty estimation. In *AAAI*, volume 33 (pp. 5933–5940).

[355] Zhong, Q., Liu, Y., Ao, X., Hu, B., Feng, J., Tang, J., & He, Q. (2020). Financial defaulter detection on online credit payment via multi-view attributed heterogeneous information network. In *WWW* (pp. 785–795).: ACM.

[356] Zhou, C. & Paffenroth, R. C. (2017). Anomaly detection with robust deep autoencoders. In *KDD* (pp. 665–674).: ACM.

[357] Zhou, Y., Song, X., Zhang, Y., Liu, F., Zhu, C., & Liu, L. (2021). Feature encoding with autoencoders for weakly supervised anomaly detection. *TNNLS*.

[358] Zhou, Z.-H. (2018). A brief introduction to weakly supervised learning. *Natl. Sci. Rev.*, 5(1), 44–53.

[359] Zimek, A., Campello, R. J. G. B., & Sander, J. (2013). Ensembles for unsupervised outlier detection: challenges and research questions a position paper. *SIGKDD Explor.*, 15(1), 11–22.

[360] Zuo, S., Liu, X., Jiao, J., Kim, Y. J., Hassan, H., Zhang, R., Gao, J., & Zhao, T. (2021). Taming sparsely activated transformer with stochastic experts. In *International Conference on Learning Representations*.