RICE UNIVERSITY

Toward Data-centric Automated Machine Learning

By

Kwei-Herng "Henry" Lai

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

Doctor of Philosophy

APPROVED, THESIS COMMITTEE

Xia Hu
Xia Hu (Apr 11, 2023 21:52 CDT)

Xia Ben Hu
Associate Professor of Computer Science

[illegible]
Arlei Lopes da Silva (Apr 10, 2023 16:26 CDT)

Arlei Lopes da Silva
Assistant Professor of Computer Science

[illegible]
Akane Sano (Apr 10, 2023 16:31 CDT)

Akane Sano
Assistant Professor of Electrical and
Computer Engineering and Bioengineering

HOUSTON, TEXAS

April 2023

ABSTRACT

Toward Data-centric Automated Machine Learning

by

Kwei-Herng "Henry" Lai

Machine learning has become increasingly popular and has shown significant success in many fields. There are four main processes involved in developing a machine learning solution: data preparation, model selection, hyper-parameter tuning, and deployment for feedback collection. While automated machine learning (AutoML) has been proposed to streamline the middle two processes and deliver efficient solutions without requiring laborious trial-and-error efforts, the framework requires a well-prepared dataset and a perfectly defined setting, which may limit its capability toward more challenging real-world applications. Recent studies suggest that data preparation is often the key to optimal solutions in many challenging real-world applications. To bridge the gap between model selection and data preparation, we propose a complimentary AutoML framework that focuses on data-centric operations, which perform automated data preparations in different stages of a machine learning pipeline. Our framework includes a data-centric model customization framework to generate sample-specific learning strategies based on the attributes of individual data samples, a data-centric knowledge acquisition framework to effectively collect expert knowledge based on data distribution while considering its long-term effects on the model training procedure, and a model-aware data preparation framework that takes data distribution and attributes into consideration to further improve the datasets

for challenging problem settings. Our goal is to develop an end-to-end data-centric AutoML system for real-world applications. To achieve this, we propose developing an end-to-end AutoML system for anomaly detection on time series data as a prototype to promote the proposed framework. With all these efforts, our research could further expand the capability of AutoML toward real-world applications.

Acknowledgement

I cannot express enough how deeply grateful I am to my advisor, Dr. Xia (Ben) Hu, for his unwavering support and invaluable mentorship during my Ph.D. journey. Dr. Hu's guidance went beyond imparting knowledge in machine learning and data mining; he taught me the essential skills of conducting research, writing scholarly papers, selecting impactful topics, and becoming an independent researcher. His encouragement, constructive criticism, and patience have been instrumental in shaping my academic and research career. I feel truly fortunate to have had him as my advisor.

I would like to extend my heartfelt appreciation to the members of my Ph.D. committee, Dr. Arlei Silva and Dr. Akane Sano, for their insightful feedback, constructive criticism, and valuable suggestions that helped to refine and enhance the quality of my research. Their expertise and encouragement have been a source of inspiration for me.

I am also deeply grateful to all the members of my research group at DATA Lab, with whom I have had the pleasure of collaborating and sharing ideas. Their insights and constructive feedback have been indispensable in advancing my research and helping me to overcome challenges along the way.

Last but not the least, I would like to express my deep appreciation and gratitude to my beloved wife and family for their unwavering support and encouragement throughout my personal and professional journey. Their constant love, understanding, and patience have been my guiding lights, motivating me to pursue my dreams and overcome challenges. I am truly grateful for their sacrifices and contributions, which have made it possible for me to achieve my goals and aspirations.

Contents

Illustrations

Tables

Chapter 1

Introduction

1.1 Motivation and Challenges

Machine learning (ML) has achieved remarkable progress in analyzing various types of data, such as image data [2], tabular data [3], graph data [4, 5], and time series data [6]. The fruitful progress further leads to successful applications such as object recognition [7, 8, 9], recommender systems [10, 11], financial forecasting [12, 13], and system monitoring [14, 15, 16]. Machine learning solutions induce knowledge from a given set of data through statistical learning techniques, and subsequently use the learned knowledge to make inferences for unseen scenarios that never encountered before.

To develop a machine learning solution, there are mainly four stages: data preparation, model selection, hyper-parameter tuning, and deployment for feedback collection [17]. However, this 4-stage procedure is often time-consuming as there exists a large number of ML models and each model has different hyper-parameters. Properly selecting an ML model with a set of properly tuned hyper-parameters for a given task and dataset requires numerous iterations of trial-and-error efforts from ML engineers.

To alleviate the burden of model selection and hyper-parameter tuning, automated machine learning (AutoML) [18, 19] has been proposed. AutoML is composed of three steps: generation, observation, and update. It first generates a model based on certain criteria. Then, the generated models will be trained on the target datasets to produce observations. Afterwards, the model architecture and the corresponding training performance will be adopted to update the model toward the next iteration of generation. By introducing AutoML, it can enable non-ML domain experts to leverage the power of ML and improve the productivity of ML engineers.

AutoML is widely acknowledged for its ability to generate models that are comparable to or even better than manually selected models. However, it has strict requirements on the quality of the given dataset and the defined problem scope. If the dataset is flawed or the target problem is too challenging, AutoML may fail to generate a quality model [20]. Particularly, recent studies [21, 22] and benchmarks [23, 24] show that complex models will not always lead to performance progress in real-world applications. In addition, from the top-1 solutions of world-class data challenges [25, 26], we can also discover that fine-grained data preparation plays a significant role in developing ML solutions with imperfect datasets towards real-world applications.

To complement the existing AutoML framework, we explore a data-centric AutoML approach that focuses on automating the data preparation and feedback collection and integration stages of building a ML solution. In particular, this thesis aims at addressing the following challenges in developing data-centric AutoML:

- **Model Customization** Individual data instances in real-world datasets may represent complex concepts or entities. Existing AutoML aims to search for a machine learning solution via a unified learning strategy. However, each data instance may have entirely different attributes and, therefore, may require customized strategies for modeling its information. In this thesis, we will study how to leverage the attribute differences between data instances and identify learning strategies for individual instances accordingly, towards automated sample-wise model customization.

- **Expertise Exploration** Shortage of label information is very common in real-world applications, and querying human knowledge is a critical solution to address this problem. The most efficient way to query human expertise is to greedily select instances with the highest prediction probability, but this may limit the exploration of different types of instances in the long term. To address this issue, we study how to consider both data distribution and model behavior simultaneously for automated knowledge acquisition.

- **Knowledge Augmentation** In many challenging real-world applications, labeled data is often scarce and limited, which makes it difficult to train a machine learning model directly. However, there may still be valuable information within the limited labeled data that can be leveraged to improve the overall quality of the dataset. To fully exploit the label information, we study on expanding the knowledge within the limited label information to automatically enhance the given dataset, making it more suitable for direct machine learning model training.

- **Framework Instantiate** In order to demonstrate the viability of the data-centric AutoML approach, it is essential to implement a practical framework on highly complex real-world applications. However, given the complexity of creating ML pipelines, there exists numerous operations for data-centric AutoML. Thus, in this thesis, we focus on developing an open-source toolkit as a practical example for outlier detection in time series data. Moreover, by using this toolkit, we investigate and analyze the advancements in this field, thereby showcasing the effectiveness and applicability of our proposed approach.

1.2 Contributions

To address the challenges in developing data-centric AutoML, this thesis introduces several approaches for facilitating data-centric AutoML. Concretely, we have made four major contributions:

- The first contribution is that we propose a meta-policy framework that models the node attributes for customizing node-wise message passing of graph neural networks (GNNs). Specifically, we uses a meta-policy to adaptively determine the number of aggregations for each node based on the node attributes. The meta-policy is trained with deep reinforcement learning (RL) by exploiting the feedback from the model. We further introduce parameter sharing and a buffer mechanism to boost the training efficiency. Experimental results on three real-world benchmark datasets suggest that our framework outperforms the state-of-the-art alternatives, showing the promise in data-centric model customization.

- The second contribution is that we propose a novel framework that learns a meta-policy for query selection. Specifically, our framework leverages deep reinforcement learning to train the meta-policy to select the most proper instance to explicitly optimize the number of discovered anomalies throughout the querying process. It is easy to deploy since a trained meta-policy can be directly applied to any new datasets without further tuning. Extensive experiments on 24 benchmark datasets demonstrate that our framework outperforms the state-of-the-art re-ranking strategies and the unsupervised baseline. The conducted empirical analysis shows the promise of data-centric automated knowledge acquisition.

- The third contribution is that we propose a data augmentation framework to iteratively generate synthetic samples by mixing up data samples from two different classes while training a ML model in a supervised fashion. Specifically, we formulate the iterative data mix-up problem into a Markov decision process (MDP), which maps data attributes into an augmentation strategy. To solve the MDP, we tailor a deep reinforcement learning framework to adapt to the discrete-continuous decision space for training a data augmentation policy and design a reward signal that explores the classifier uncertainty and encourages performance improvement regardless of the convergence of the classifier. The extensive experiments on 7 publicly available benchmark datasets with 3 different kinds of classifiers show the promise of model-aware data preparation.

- The fourth contribution is that we instantiate the data-centric AutoML on a challenging real-world application, time series outlier detection, with a highly modular system that supports easy pipeline construction. The basic building block of the system is primitive, which is an implementation of a function with hyperparameters. The system currently supports more than 70 primitives, including data processing, time series processing, feature analysis, detection algorithms, and a reinforcement module. Additionally, a data-driven searcher is provided to automatically discover the most suitable pipelines given a dataset. With the aid of the system, we comprehensively benchmark the advancements of outlier detection on time series data and show the effectiveness of the system.

Putting them together, our research validate the feasibility of data-centric AutoML and could facilitate its research as well as the development in various applications for our daily life.

1.3 Thesis Outline

The remainder of the thesis is organized as follows:

- **Chapter 2: Background.** In this chapter, we introduce the background of AutoML and the formal definition of the corresponding techniques: Markov decision process and deep reinforcement learning.

- **Chapter 3: Data-centric Automated Model Customization.** In this chapter, we formally define the problem of model customization on graph data and discuss how to customize training strategies for individual nodes in a graph.

- **Chapter 4: Data-centric Automated Knowledge Acquisition.** In this chapter, we identify the problem of greedily acquire human expertise without considering data distribution and present an adaptive querying strategy for maximizing long-term benefits.

- **Chapter 5: Model-aware Automated Data Preparation.** In this chapter, we study the data augmentation strategies and present an automated data preparation strategy that considers both data attributes and model status.

- **Chapter 6: Data-centric AutoML for Time Series Outlier Detection: System, Definition, and Benchmark.** In this chapter, we present an automated time series outlier detection system and discuss its applicability on studying the advancements in the field and the corresponding results.

- **Chapter 7: Conclusions and Future Research Opportunities.** We conclude this thesis by summarizing our contributions and providing outlooks. We also highlight several research directions to motivate future exploration

Chapter 2

Background

Automated machine learning (AutoML) [27] has been widely adopted to make ML accessible to the general public, improve the productivity of ML engineers, and even create better ML solutions compared to human-designed models. To realize AutoML [27] search process, existing methods exploit algorithms such as Bayesian optimization [28, 29], evolutionary algorithm [30, 31] and reinforcement learning [32]. However, Bayesian optimization and evolutionary algorithms are stateless approaches equipped with limited parameters, which are limited to a smaller search space [33]. In data-centric AutoML, as we are searching not only the model hyperparameters but also learning data-centric operations through traversing feature space or considering data distribution, reinforcement learning can better serve the role to tackle the problem. In addition, the temporal difference learning [34] and Bellman equation-formed reward signal [35] in reinforcement learning can better lead to long-term beneficial decisions compared to the stateless setting of the two alternatives. To drive the data-centric AutoML with reinforcement learning, one may need to formulate the problem into a Markov decision process. Here, we introduce the definition of the Markov decision process and reinforcement learning.

2.1 Sequential Decision Modeling with Markov Decision Process

Markov Decision Process (MDP) is a mathematical framework to describe sequential decision making process. Specifically, let $\mathcal{M}$ be an MDP, represented by a quintuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}_T, \mathcal{R}, \gamma)$, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, $\mathcal{P}_T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}^+$ is the state transition probability function that maps the current state s, action a and the next state s' to a probability, $\mathcal{R} : \mathcal{S} \rightarrow \mathbb{R}$ is the immediate reward function, and $\gamma \in (0, 1)$ is a discount factor. At each timestep t, the agent takes action $a_t \in \mathcal{A}$ based on the current state $s_t \in \mathcal{S}$, and observes the next state s_{t+1} as well as a reward signal $r_t = \mathcal{R}(s_{t+1})$. We aim to search for the optimal decisions so as to maximize the expected discounted cumulative reward, i.e., we would like find a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ to maximize $\mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_t]$.

2.2 Deep Reinforcement Learning

Deep reinforcement learning is a family of algorithms that solve the MDP with deep neural networks. To better adapting to large search space of data-centric AutoML, we consider model-free deep reinforcement learning, which learns to take the optimal actions through exploration. One of the pioneering work Deep-Q Learning (DQN) [36] uses deep neural networks to approximate state-action values Q(s, a) that satisfies

$$Q(s, a) = \mathbb{E}_{s'}[\mathcal{R}(s') + \gamma \max_{a'}(Q(s', a')], \tag{2.1}$$

where s' is the next state and a' is the next action. DQN introduces two techniques to stabilize the training: (1) a replay buffer to reuse past experiences; (2) a separate target network that is periodically updated. To obtain a policy $\widetilde{\pi}$, DQN select the action based on the Q value with epsilon-greedy algorithm.

Table 2.1 : Notations cover deep reinforcement learning in this thesis proposal.

Notation	Definition
$\mathcal{S}$	A finite set of state
$\mathcal{A}$	A finite set of actions
$\mathcal{P}_T$	State transition probability function.
s_t	A state in the timestep t.
a_t	An action in the timestep t.
r_t	Reward in the timestep t.
$Q(s,a)$	Q-function to evaluate the state s and action a.
$V(s)$	Value function to evaluate the state s.
$\widetilde{\pi}$	An optimal policy function to generate action a for a given state s.

However, as DQN only capable of addressing discrete action space, Deep Deterministic Policy Gradient (DDPG) [37] is introduced to tackle the problem. To perform a continuous action, instead of performing epsilon-greedy on the Q values, the DDPG learn an actor network $\pi(s_t|\theta)$ for obtaining an optimal policy $\widetilde{\pi}$ by deterministically mapping a given state s_t to an action vector a_t. The $\pi(s_t|\theta)$ is trained the by maximizing the approximated cumulative reward that generated by the critic network $Q(.)$ (i.e., Q-network). Specifically, given N transitions, a projected action $\pi(s_t|\theta)$ can be generated as the input of the critic to minimize the following loss function:

$$L_\pi(\theta) = -\frac{1}{N}\sum_{i=1}^{N} Q(s_i, \pi(s_i|\theta)), \tag{2.2}$$

where the action $\pi(s_i)$ is a real number vector.

The above two DRL algorithms are trained in an offline setting, and may be inefficient when the search procedure are required to be performed in online manner. To this end, an efficient online algorithm, named Proximal Policy Optimization (PPO) [38] is proposed to alleviate the problem. Specifically, sine training a Q-function requires past experience, PPO instead trains a value function $V(s)$ through direct interacting with the environment. Then, it exploits a generalized advantage estimator [39] to compute a short-term advantage:

$$\hat{A}_t = \delta_t + \sum_{t'=1}^{T-1+1} (\gamma\lambda)^{t'} \delta_{t+t'}, \tag{2.3}$$

where $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$, T is the total timesteps in an episode, γ is the discount factor, and λ is a hyper-parameter to control the bias-variance trade-off. This way, it trains the actor network π toward an optimal policy $\widetilde{\pi}$ via a clipped surrogate objective:

$$L_t^{CLIP}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)], \tag{2.4}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$, $\pi_{\theta_{old}}$ is the policy before the update, $clip(r_t(\theta), 1-\epsilon, 1+\epsilon)$ will clip $r_t(\theta)$ into range $[1-\epsilon, 1+\epsilon]$, and ϵ is a hyper-parameter to control the clip range. This objective ensures the new policy will not deviate too much from the old policy and therefore leads to stable policy improvement. To drive the policy training, a combinatorial loss is derived to simultaneously update the value loss:

$$L_t(\theta) = \hat{\mathbb{E}}_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 \cdot entropy(\pi_\theta(\cdot|s_t))], \tag{2.5}$$

where $L_t^{VF}(\theta)$ is a squared-error loss $(V_\theta(s_t) - V_t^{target})^2$, V_t^{target} is estimated based on the collected data, $entropy(\cdot)$ is a term to encourage exploration, c_1 and c_2 are hyper-parameters.

Chapter 3

Data-centric Automated Model Customization

3.1 Motivation

To study the possibility of sample-wise learning mechanism customization, we conducted research on graph-structured data since they are pervasive in many real-world applications, such as anomaly detection [40], molecular structure generation [41], social network analysis [42] and recommendation systems [43]. In particular, we focus on the key techniques behind these applications, graph representation learning [44], which aims at extracting information underlying the graph-structured data into low dimensional vector representations. Following the great success of convolution neural networks [45], a lot of momentum has been gained by graph convolution networks (GCNs) [46, 47] and graph neural networks (GNNs) [48, 49, 50, 51, 52] to model the local structures and hierarchical patterns of the graph.

To validate the need for model customization, we hypothesized that different nodes require different levels of aggregation iterations to capture the structural information fully. We conducted node classification on the Cora dataset using a standard GCN with different layers to explore this hypothesis. Figure 3.1 illustrates the results of 100 runs of 20 randomly sampled nodes. The figure shows that certain nodes achieve better classification performance with more GNN layers, indicating the need for model customization. For instance, node 17 and 20 perform better when aggregating 4 iterations, whereas node 2 requires more iterations of aggregation. While most nodes

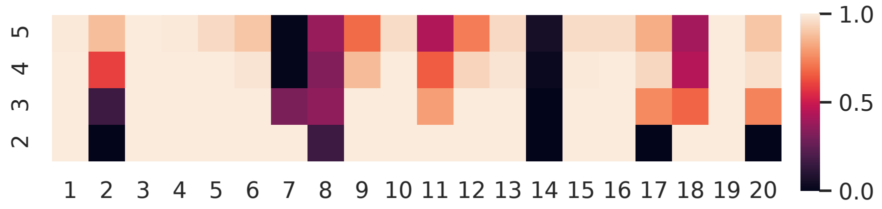


Figure 3.1 : The effect of different iterations of aggregation in GCN on 20 randomly sampled nodes of Cora dataset. The X-axis denotes the id of the sampled node, and Y-axis is the iteration number (layers) in GCN. The color from dark to light represents the ratio of being predicted correctly with 100 different runs.

are well-classified in 2 hop aggregation, our observations motivate us to investigate how to adaptively aggregate different hops of neighbors for each node to improve the performance of GNNs.

However, this task is nontrivial due to the following challenges. First, real-world graphs are usually complex with multiple types of attributes: it is hard to determine the suitable iterations of aggregation for each node. For example, in a citation network such as the Cora dataset, each node represents a piece of paper with 1,433 dimensions of bag-of-words features. It is difficult to design the aggregation strategy based on such a sparse and large number of features. Second, even though we can define a proper aggregation strategy for each node, it remains challenging to train GNNs on these nodes since we need to feed these nodes into different numbers of network layers. Improper management of the sampled nodes will greatly affect the training efficiency of the algorithms.

To address the above challenges, in this paper, we propose Policy-GNN, a meta-policy framework to model the complex aggregation strategy. Motivated by the recent success of meta-policy learning [53], we formulate the graph representation learning as a Markov decision process (MDP), which optimizes a meta-policy by exploiting the feedback from the model. Specifically, the MDP iteratively samples the number of hops of the current nodes and the corresponding neighboring nodes with a meta-policy, and trains GNNs by aggregating the information of the nodes within the sampled hops. The proposed MDP successfully integrates the sampling procedure and message passing into a combined learning process. To solve this MDP, we employ deep reinforcement learning algorithms enhanced by a tailored reward function to train the meta-policy. In addition, we introduce parameter sharing and a buffer mechanism that enables batch training to boost the training efficiency.

We demonstrate the effectiveness of our framework by implementing it using the widely-used deep Q-learning (DQN) algorithm [54] in combination with the graph convolution network (GCN) architecture. In order to validate the proposed approach, we conducted extensive experiments on various real-world datasets and compared our results with several state-of-the-art baseline methods, including model-centric AutoML solutions and graph neural architecture search methods. Our experimental results show that our proposed Policy-GNN outperforms these baselines on several benchmark datasets, demonstrating the efficacy of our approach in customizing the learning mechanism of GNNs for improved performance.

3.2 Preliminaries

In this section, we first define the problem of aggregation optimization. Then we introduce the background of graph neural networks, Markov decision process, and deep reinforcement learning. Finally, we discuss the related works to the proposed framework.

Table 3.1 : Main notations in this section. The top rows are for graph representation learning; the bottom rows cover deep reinforcement learning.

Notation	Definition
V	The set of vertices in the graph.
E	The set of edges in the graph.
G	A graph G with node set V and edge set E.
$\mathbf{A}$	The adjacency matrix of graph G.
$\widetilde{\mathbf{A}}$	The normalized adjacency matrix of graph G
$\mathbf{X}$	The attribute information matrix.
$N_k(v)$	The k-hop neighborhood of the node v.
$\mathcal{S}$	A finite set of state
$\mathcal{A}$	A finite set of actions
$\mathcal{P}_T$	State transition probability function.
s_t	A state in the timestep t.
a_t	An action in the timestep t.
r_t	Reward in the timestep t.
$Q(s, a)$	Value function to evaluate the state s and action a.
$\widetilde{\pi}$	Policy function to generate action a for a given state s.
b	The window size of the baseline in reward function.

3.2.1 Problem Definition

Let $G = (V, E)$ denote a graph, where $V = \{v_1, v_2, v_3 ... v_n\}$ is the set of vertices, $E \subseteq V \times V$ is the set of edges and n is the number of vertices in G. Each edge $e = (u, v, w) \in E$ consists of a starting node $u \in V$, an end node $v \in V$ and a edge weight $w \in \mathbb{R}^+$ that indicates the strength of the relation between u and v. We use the convention of adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ and attribute information matrix $\mathbf{X} \in \mathbb{R}^{n \times m}$to represent G, where each entry $\mathbf{A}_{ij} \in \mathbb{R}^+$ is the edge weight between v_i and v_j, and each row $\mathbf{X}_i$ is a m-dimensional attribute vector for node v_i. For each node v, we define a set of k-hop neighborhood as $N_k(v)$, where each $v' \in N_k(v)$ is the k-order proximity of the node v. Graph representation learning aims at embedding the graph into low-dimensional vector spaces. Formally, the objective can be expressed by a mapping function $\mathcal{F} : v \to \mathbb{R}^d$, where d is the dimension of the vector. Following the message passing strategy in GNNs, our goal is to learn the node representation through aggregating the information from k-hop neighborhood $N_k(v)$. In this way, the proximity information is preserved in the low dimensional feature vectors and the learned representations can be used in down downstream tasks such as node classification. Table 3.1 lists the main notations used in the paper.

Based on the notations defined above, we formulate the problem of aggregation optimization as follows. Given an arbitrary graph neural network algorithm, our goal is to jointly learn a meta-policy $\widetilde{\pi}$ with the graph neural network, where $\widetilde{\pi}$ maps each node $v \in V$ into the number of iterations of aggregation, such that the performance on downstream task is optimized.

3.2.2 Learning Graph Representations with Deep Neural Networks

Graph neural networks (GNNs) have gained attention due to their great success on graph data, similar to convolutional neural networks on image data. Several GNN models have been proposed, including Graph Convolutional Networks (GCNs) [46], GraphSAGE [55], and GAT [52]. Inspired by Weisfeiler-Lehman (WL) graph isomorphism test [56], the process of learning a GNN involves three steps: (1) **initialization**, which involves initializing the feature vectors of each node by the attributes of the vertices, (2) **neighborhood detection**, which involves determining the local neighborhood for each node to gather further information, and (3) **information aggregation**, which involves updating the node feature vectors by aggregating and compressing feature vectors of the detected neighbors. These steps enable GNNs to capture the local structures and hierarchical patterns of the graph and learn the node representations.

To effectively gather the information from neighbors, several feature aggregation functions (graph convolution) have been proposed. One of the most representative method is **Graph Convolution Network (GCN)** [46]. Let $\widetilde{\mathbf{A}}$ be the normalized adjacency matrix where $\widetilde{\mathbf{A}}_{uv} = \frac{\mathbf{A}_{uv}}{\sum_v \mathbf{A}_{uv}}$, GCN performs weighted feature aggregation from the neighborhood:

$$\mathbf{h}_v^k = \sigma(\sum\nolimits_{u \in \{v\} \cup N_1(v)} \widetilde{\mathbf{A}}_{uv} \mathbf{W}_k \mathbf{h}_u^{k-1}). \tag{3.1}$$

where $\mathbf{W}_k$ is the trainable parameter in the k-th graph convolution layer, $N_1(v)$ is the one hop neighborhood of the node v, and $\mathbf{h}_u$ and $\mathbf{h}_v$ are d dimensional feature vectors. In this work, we have utilized GCN as the fundamental GNN model. However, our proposed framework is not restricted to GCN, and other advanced GNN models can also be incorporated into our framework.

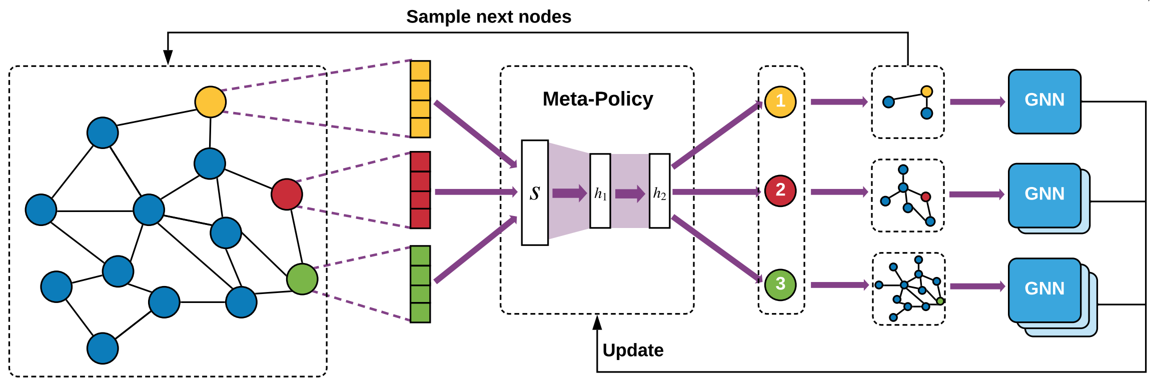


Figure 3.2 : An illustration of Policy-GNN with meta-policy sampling 1, 2 and 3 iterations of aggregation for different nodes. The learning procedure follows a Markov Decision Process. In each timestep, the mata-policy samples the number of layers (action) based on the attributes of the nodes (state). The next nodes (next state) are obtained by randomly sampling a node from the k-hop neighbors of the current nodes, where k is the output of the mata-policy (action). The meta-policy is updated based on the feedback from the GNN.

3.2.3 Related Works

Graph Neural Networks

Neural network model [57] has been studied for years to preserve the information of a graph into vector space. Based on the graph spectral theory, graph convolutional networks have been proposed to capture the proximity information of the graph-structured data [47, 46]. To deal with the large scale real-world data, the spatial-based GNNs are proposed with a message passing strategy, called graph convolution. Specifically, the graph convolution learns the node representations by aggregating features from its neighbors. With k times of graph convolution, the k-hop neighborhood information of nodes are encoded into feature vectors. Recently, various strategies have been proposed to advance the message passing in GNNs, including the advanced aggregation functions [55], node attention mechanisms [52], graph structure poolings [58], neighborhood sampling methods [59], graph recurrent networks [60], and multi-channel GNNs [61]. In this work, we explore an orthogonal direction that learns a meta-policy to customize message passing strategy for each node in the graph.

Graph Neural Architecture Search

Given a specific task, neural architecture search (NAS) aims to discover the optimal model without laborious neural architecture tuning from the predefined architecture search space [62, 63, 64, 65, 66]. Following the success of NAS, the concept of automation has been extended to GNNs architectures[67, 68]. The search space consists of all the possible variants of GNN architecture. A controller is then designed to explore the search space to find the optimal model and maximize the performance. The concept of meta-policy in *Policy-GNN* is similar to the controller in NAS. Instead of focusing on searching the space of neural architecture, our meta-policy learns to sample the best iterations of aggregation for each node, and achieves much better performance in real-world benchmarks.

Meta-policy Learning

Recent advances in deep reinforcement learning (RL) have shown its wide applicability, such as games [69, 70] and neural architecture search [62]. In general, the policy of deep neural networks is used to extract features from observations and make the corresponding decisions. The idea of meta-policy learning [53] is to train a meta-policy by exploiting the feedback from the model to optimize the performance. The meta-policy learning has been widely studied to generate and replay experience data [53], to learn a dual policy for improving the sample efficiency of the agent [71] and to maximize the performance by learning reward and discount factor [72]. However, the successes of these studies are limited in simulated environments, which are far from real-world applications. Our work demonstrates the effectiveness of meta-policy learning in the context of GNNs in real-world scenario.

3.3 Methodology

Our framework is composed of two key components, as depicted in Figure 3.2: the *Meta-Policy* module and the *GNN* module. The goal of the *Meta-Policy* module is to learn the correlation between node attributes and the appropriate iterations of aggregation. Meanwhile, the *GNN* module utilizes the learned *Meta-Policy* to perform graph representation learning.

By combining the two modules, we formulate graph representation learning as a Markov decision process. Firstly, the *Meta-Policy* module treats the node attributes as states, represented as red/yellow/green feature vectors, and maps these states into actions, which correspond to the number of hops shown in the red/yellow/green circles. Next, we sample the next state from the k-hop neighbors of each node, where k is the output of the *Meta-Policy*. The *GNN* module, located at the right side of Figure 3.2, selects a pre-built k-layer GNN architecture, determined by the output of the meta-policy, to learn the node representations and obtain a reward signal for updating the *Meta-Policy*.

In what follows, we elaborate on the details of Policy-GNN. We first describe how we can train the meta-policy with deep reinforcement learning. Then, we show how we train GNNs with the meta-policy algorithm. Last, we introduce the buffer mechanism and parameter sharing strategy, which boost the training efficiency in real-world application scenarios.

3.3.1 Aggregation Optimization with Deep Reinforcement Learning

We discuss how the process of learning an optimal aggregation strategy can be naturally formulated as a Markov decision process (MDP). As discussed in Section 2.1, the key components of an MDP include states, actions and rewards, as well as a transition probability that maps the current state and action into the next state. With the above in mind, we now discuss how we define these components in the context of graph representation learning:

- **State ($\mathcal{S}$)**: The state $s_t \in \mathcal{S}$ in timestep t is defined as the attribute of the current node.
- **Action ($\mathcal{A}$)**: The action $a_t \in \mathcal{A}$ in timestep t specifies the number of hops of the current node.
- **Reward Function($\mathcal{R}$)**: We define the reward r_t in timestep t as the performance improvement on the specific task comparing with the last state.

Based on the definitions above, the proposed aggregation process consists of three phases: 1) selecting a start node and obtaining its attributes as the current state s_t, 2) generating an action a_t from $\pi(s_t)$ to specify the number of hops for the current node, and 3) sampling the next node from a_t-hop neighborhood and obtaining its attributes as the next state s_{t+1}. Figure 3.3 gives an simple example of how does MDP work for graph representation learning. Specifically, we formulate the sampling process of GNN into a MDP. With the specified action k, we sampled the next state from the k-hop neighborhood.

We propose to employ deep reinforcement learning algorithms to optimize the above MDP. Since the action space of the aggregation process is a discrete space, i.e.,

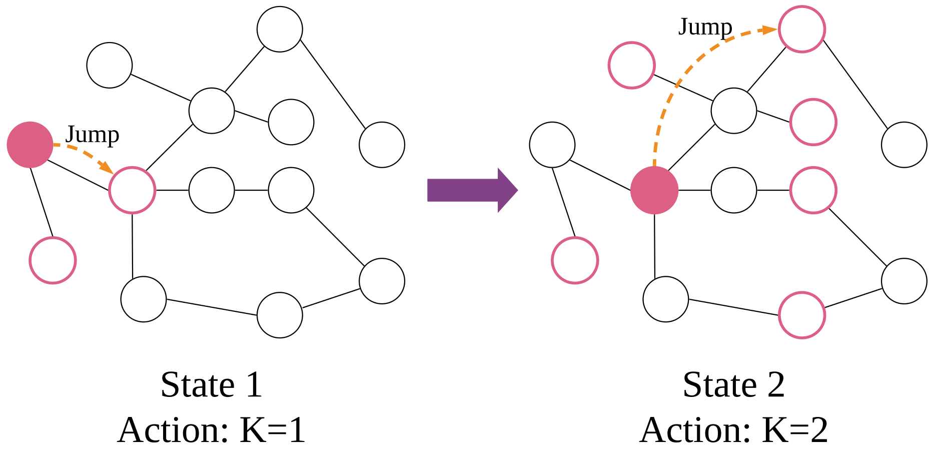


Figure 3.3 : An illustration of node sampling as Markov decision process (MDP). State 1 is the attribute of the solid-pink node in the left figure. We take action $K = 1$ and randomly jump to one of 1-hop neighboring nodes (pink-framed). As a result, state 1 transits to state 2, which is the attribute of the solid-pink node in the right figure. We then sample the next node from 2-hop neighbors by taking action $K = 2$. The above procedure can be naturally treated as an MDP.

$\mathcal{A} \in \mathbb{N}$ and any arbitrary action $a \in \mathcal{A}$ is always a finite positive integer, we introduce the deep Q-learning [69, 54] to address the problem.

The key factor in guiding the deep Q-learning is the reward signal. We employ a baseline in the reward function, defined as

$$\mathcal{R}(s_t, a_t) = \lambda(\mathcal{M}(s_t, a_t) - \frac{\sum_{i=t-b}^{t-1} \mathcal{M}(s_i, a_i)}{b-1}), \tag{3.2}$$

where $\frac{\sum_{i=t-b}^{t-1} \mathcal{M}(s_i,a_i)}{b-1}$ is the baseline for each timestep t, $\mathcal{M}$ is the evaluation metric of a specific task, b is a hyperparameter to define the window size for the historical performance to be referred for the baseline, λ is a hyperparameter to decide the strength of reward signal and $\mathcal{M}(s_t, a_t)$ is the performance of the task on the validation set at timestep t. The idea of employing a baseline function is to encourage the learning agent to achieve better performance compared with the most recent b timesteps. In this work, we showcase the learning framework by using the node classifications

Algorithm 1 Policy-GNN

```
1: Input: Maximum layer number $K$, DQN training step $S$, total training epoch $T$, epsilon probability $\epsilon$, window size of reward function $b$.
2: Initialize $K$ GNN layers, $Q$ function, memory buffer $\mathcal{D}$, GNN buffer $\mathcal{B}$
3: Randomly sample a node, generate a state $s_0$ by attribute of the nodes.
4: for $t = 0, 1, 2..., T$ do
5:     With probability $\epsilon$ randomly choose an action $a_t$,
       otherwise obtain $a_t = \text{argmax}_a Q(s_t, a)$.
6:     Store $s_t$ and $a_t$ into GNN buffer $\mathcal{B}_{a_t}$
7:     Apply Algorithm 2 with input $a$ and $\mathcal{B}$ to train the GNNs.
8:     Obtain $r_t$ on validation dataset via equation 3.2
9:     Sample the next state $s_{t+1}$ from $a_t$-hop neighborhood.
10:    Store the triplet $T_t = (s_t, a_t, s_{t+1}, r_t)$ into $\mathcal{D}$
11:    for step = 1, 2, .., $S$ do
12:        Optimize $Q$ function using the data in $\mathcal{D}$ via equation 2.1.
13:    end for
14: end for
```

accuracy on the validation set as the evaluation metric.

Based on the above reward function, we train the Q function by optimizing Equation (2.1). In this way, we can apply epsilon-greedy policy [73] to obtain our policy function $\widetilde{\pi}$.

3.3.2 Graph Representation Learning with Meta-Policy

We discuss the role that meta-policy plays in graph representation learning and how to learn the node embeddings with *Meta-Policy*. In the proposed framework, we explicitly learn the aggregation strategy through the aforementioned meta-policy learning. As Figure 3.2 shows, the *Meta-Policy* maps the node attributes to the number of hops, and uses the number of hops to construct a specific GNN architecture for each node. Compared with the recent efforts [74, 75] on skip-connection, which implicitly learns an aggregation strategy for the whole graph, the proposed framework explicitly aggregates the information. Furthermore, the Policy-GNN provides the flexibility to include all of the previous research fruits, including skip-connection, to facilitate graph representation learning.

To showcase the effectiveness of the framework, we apply the meta-policy on basic GCN [46], and learn the node representations by equation 3.1. Instead of aggregating a fixed number of layers for every nodes, our framework aggregates a_t layers for each node attribute s_t at timestep t. The GNN architecture construction can be represented as the transition equations as follows:

$$
\begin{aligned}
&\mathbf{h}_v^1 = \sigma\left(\textstyle\sum_{u_1 \in \{v\} \cup N_1(v)} \widetilde{a}_{u_1 v} \mathbf{W}_1 \mathbf{X}_u\right), \\
&\vdots \\
&\mathbf{h}_v^{k=a_t} = \sigma\left(\textstyle\sum_{u_k \in \{u_{k-1}\} \cup N_k(v)} \widetilde{a}_{u_k u_{k-1}} \mathbf{W}_k \mathbf{h}_v^{k-1}\right), \\
&output = \mathrm{softmax}(\mathbf{h}_v^{a_t}),
\end{aligned}
$$

where $\mathbf{h}_v^k$ is the d dimensional feature vector of node v of the k layer, $\mathbf{X}_u$ is the input attribute vector of node u, $\mathbf{W}_k$ is the trainable parameters of the k layer, $\sum \widetilde{a}$ indicates the neighborhood specified by the meta-policy, σ is the activation function of each layer, and $k = a_t$ is the number of aggregation that decided by $\widetilde{\pi}$ function in

Algorithm 2 Training GNN with Buffer Mechanism

```
1: Input: Action $a$, GNN buffer $\mathcal{B}$
2: if $\mathcal{B}_a$ is full then
3:     for layers = 1, 2, ... $a$ do
4:         Stack GNN layers as subsection 3.3.2 mentioned.
5:     end for
6:     Train the stacked GNNs on the buffer of action $\mathcal{B}_a$
7:     Clear the buffer $\mathcal{B}_a$
8: end if
```

timestep t. Note that, we can always replace the aggregation function in each layer as well as the final output layer for different tasks in our framework.

3.3.3 Accelerating Policy-GNN with Buffer Mechanism and Parameter Sharing

One of the challenging problems to practically develop the proposed framework is training efficiency. Since re-constructing GNNs at every timestep is time-consuming, and the number of parameters may be large if we train multiple GNNs for different actions. To address the problems, we introduce the following techniques to improve training efficiency.

Parameter Sharing. We reduce the number of training parameters via parameter sharing mechanism. Specifically, we first initialize the maximum layer number (k) of GNN layers in the beginning. Then, in each timestep t, we repeatedly stack the layers by the initialization order for the action a_t to perform GNN training. In this way, if the number of hidden units for each layer is n, we only need to train $k \times n$

parameters, rather than $\frac{nk(k+1)}{2}$ parameters.

Buffer mechanism for graph representation learning. We construct action buffers for each possible action. In each timestep, after we obtain a batch of nodes' attributes and a batch of actions, we store them into the action buffer and check if the buffer has reached the batch size. If the buffer of a specific action is full, then we construct the GNN with selected layers, and train the GNN with the data in the buffer of the action. After the data has been trained on the GNN, we clear the data in the buffer. Practically, if the batch size is too large, the mechanism will still take much time. However, compared with the costly GNN construction in every timestep for training only one instance, the mechanism significantly improves the training efficiency [*].

The detail of the algorithm is provided in algorithm 1 and 2. We first initialize k GNN layers, the Q function, the GNN buffer $\mathcal{B}$ and the memory buffer $\mathcal{D}$ to store the experiences. Then, we randomly sample a batch of nodes and generate the first state with the node attributes. We feed the state to Q function and obtain the actions with ϵ probability to randomly choose the action. Based on the chosen action, we stack the layers from the initialized GNN layers in order, and train the selected layers to get the feedback on the validation set. With the feedback on the validation set, we further obtain the reward signal via equation 3.2 and store the transitions into a memory buffer. In this way, with past experiences in the memory buffer, we randomly sample batches of data from the memory buffer and optimize the Q function based on equation 2.1 for the next iteration.

[*]The preliminary result on comparing the training time with/without the mechanisms on Cora dataset shows that the training efficiency is significantly improved by 96 times.

Table 3.2 : Summary of data statistics used in our experiment, including Cora, Citeseer, and Pubmed [1].

Dataset	#Nodes	#Features	#Classes	#Training	#Validation	#Testing
Cora	$2,708$	$1,433$	7	140	500	$1,000$
Citeseer	$3,327$	$3,703$	6	120	500	$1,000$
Pubmed	$19,717$	500	3	60	500	$1,000$

3.4 Experiment

In this section, we empirically evaluate the proposed *Policy-GNN* framework. We mainly focus on the following research questions:

- How does *Policy-GNN* compare against the state-of-the-art graph representation learning algorithms (Section 3.4.4)?
- What does the distribution of the number of layers look like for a trained meta-policy (Section 3.4.5)?
- Is the search process of *Policy-GNN* efficient in terms of the number of iterations it needs to achieve good performance (Section 3.4.6)?

3.4.1 Datasets

We consider the benchmark datasets [1] that are commonly used for studying node classification, including the citation graph-structured data of Cora, Citeseer, and Pubmed. The statistics of these datasets are summarized in Table **??**. In these citation graphs, the nodes and the edges correspond to the published documents and their citation relations, respectively. The node attribute information is given by bag-of-words representation of a document, and each node is associated with a class label. We follow the standard training/validation/test splits as in previous studies [52, 76, 68, 67] Specifically, we use 20 nodes per class for training, 500 nodes for validation, and 1000 nodes for testing. The proposed model and baselines are all trained and evaluated with the complete graph structure and node features in the training dataset, without using the node labels in the held-out validation and testing sets. The model hyperparameters are selected based on the performance on the validation set. The final classification accuracy is reported on the test set.

3.4.2 Baselines

We compare *Policy-GNN* with the state-of-the-art graph representation learning algorithms. Specifically, we consider the baselines in the following three categories: network embedding methods, traditional graph neural networks (Static-GNNs), and the recently proposed neural architecture search based GNNs (NAS-GNNs). Note that NAS-GNNs also include a meta-policy that operates upon the GNNs. The search space of NAS-GGNs focuses on the message passing functions, whereas our meta-policy searches for the optimal iterations of aggregation. We are interested in studying whether our search strategy is more effective than NAS.

Network Embedding. Network embedding learns the node representations in an unsupervised fashion. To capture the proximity information, network embedding utilizes the learning objective of the skip-gram method [77] and introduces various sampling techniques such as random walk. We consider *DeepWalk* [78] and *Node2vec* [79], which are the most commonly used baselines in network embedding. Both methods learn node feature vectors via random walk sampling. Compared with *DeepWalk*, *Node2vec* introduces a biased random walking strategy to sample the neighborhood for each node in BFS and DFS fashion.

Static-GNNs. GNNs learn the node representations in a supervised fashion, where neural networks are trained with both feature vectors and the labels in the training set. There are a huge number of algorithms in the literature. To better understand the performance of our *Policy-GNN*, we try to include all the state-of-the-art methods that we are aware of for comparison. Specifically, we consider the following baselines which use different layer structure and batch sampling techniques. *Chebyshev* [47] and *GCN* [46] perform information aggregation based on the Laplacian or adjacent matrix of the complete graph. *GraphSAGE* [80] proposes neighborhood batch sampling to enable scalable training with max, min and LSTM aggregation functions. *GAT* [52] introduces multi-head attention mechanism into aggregation function, which learns the importance of the neighborhood of each node for information aggregation. *LGCN* [76] automatically selects a fixed number of neighbors for each node based on value ranking, and transforms graph data into grid-like structures in 1-D format to extract the proximity information. *g-U-Nets* [58] proposes an encoder-decoder structure with gPool layer to adaptively sample nodes to form a smaller graph, and a gUnpool to restore the original graph from the smaller graph and use skip-connection for deeper architectures.

NAS-GNN. Recently, neural architecture search has been extensively introduced to search the optimal GNN architectures within fix iterations of aggregation. *GraphNAS* [67] uses a recurrent neural network to sample the variable-length architecture strings for GNNs, and applies reinforcement rule to update the controller. *AGNN* [68] follows the setting in GraphNAS, and further introduces the constrained parameter sharing and reinforced conservative controller to explore well-performing GNNs efficiently. The search space of both baselines covers the aggregation functions, activation functions, attention functions, and the number of heads for multi-head attention mechanism.

In addition to the above three categories, we also include a variant of our *Policy-GNN*, named *Random Policy*. *Random Policy* will randomly make the decisions for each node without the reinforcement learning update. This baseline can be regarded as a special case of our framework epsilon probability 1.0.

Table 3.3 : Classification accuracy on benchmark datasets Cora, Citeseer adn Pubmed. Symbol ↑ denotes the performance improvement achieved by *Policy-GNN*, compared with network embedding, Static-GNNs, NAS-GNNs and random policy.

Baseline Class	**Model**	**#Layers**	**Cora**		**Citeseer**		**Pubmed**	
			Accuracy	↑	Accuracy	↑	Accuracy	↑
Network	DeepWalk [78]	-	0.672	+36.8%	0.432	+107.6%	0.653	+41.0%
Embedding	Node2vec [79]	-	0.749	+22.7%	0.547	+64.0%	0.753	+22.3%
Static-GNNs	Chebyshev [47]	2	0.812	+13.2%	0.698	+28.5%	0.744	+23.8%
	GCN [46]	2	0.815	+12.8%	0.703	+27.6%	0.790	+16.6%
	GraphSAGE [55]	2	0.822	+11.8%	0.714	+25.6%	0.871	+5.7%
	GAT [52]	2	0.830	+10.7%	0.725	+23.7%	0.790	+16.6%
	LGCN [76]	2	0.833	+10.3%	0.730	+22.9%	0.795	+15.8%
	g-U-nets [58]	2	0.844	+8.9%	0.732	+22.5%	0.796	+15.7%
NAS-GNNs	GraphNAS [67]	2	0.833	+10.3%	0.724	+22.0%	0.788	+17.9%
	AGNN [68]	2	0.827	+9.9%	0.727	+21.5%	0.790	+15.6%
Policy-GNN	Random Policy	$2 \sim 5$	0.770	+19.4%	0.656	+36.7%	0.708	+16.9%
	Policy-GNN	$2 \sim 5$	**0.905**	-	**0.897**	-	**0.921**	-

3.4.3 Implementation Details

In the experiments, we implement our framework using *GCN* [46] and *DQN* [69, 54]. For *GCN* layers, we apply *relu* as the activation function for each layer and include the dropout mechanism between the layers with 0.5 dropout rate. To train the GCN, we utilize *Adam* optimizer with a learning rate 0.01 and a weight decay rate 0.0005, and set the batch size as 128. We use the *DQN* implementation in [70] and construct 5-layers of *MLP* with $(32, 64, 128, 64, 32)$ hidden units for Q function. We set the memory buffer size to be 10000 and the training batch size to be 256. For the epsilon probability of policy function, we set up a linear scheduler with starting probability 1.0 to the end probability 0.1, where the probability linearly decades every 10 training steps. We follow the setting of GraphNAS and AGNN by training the meta-policy for 1000 episodes. We save the model that has the best accuracy on the validation set and report the performance of the model by applying it on the test set.

3.4.4 Performance Comparison on Benchmark Datasets

Table 3.3 presents a summary of the performance of the proposed *Policy-GNN* and the considered baselines on the three benchmark datasets. For *Policy-GNN*, the number of layers for each node is randomly sampled between 2 and 5. The performance evaluation is based on classification accuracy on the test data. The experimental results demonstrate that our *Policy-GNN* outperforms all the baselines consistently and significantly across all datasets. Specifically, compared to the second-best algorithm, *Policy-GNN* achieves 8.9%, 21.5%, and 5.7% improvements in accuracy on Cora, Citeseer, and Pubmed, respectively. The experimental results also reveal some interesting observations as follows.

First, the end-to-end learning procedure outperforms a separate pipeline. Specifically, compared with the network embedding approaches, all of the GNN approaches, including our framework, achieve much better performance. It is reasonable since network embedding approaches are unsupervised with two separate components of random walk sampling and skip-gram objective, which may be sub-optimal in node classification tasks.

Second, using different iterations of aggregation for different nodes is critical for boosting the performance of GNNs. Specifically, built on *GCN*, *Policy-GNN* is able to significantly improve the vanilla *GCN* by 12.8%, 27.6% and 16.6% on the three datasets, respectively. *Policy-GNN* also outperforms the other more advanced Static-GNNs approaches. For example, our *Policy-GNN* consistently outperforms *GCN*, *GAT*, and *GraphSAGE*, which use fix number of layers, simply use pure *GCN* layer. The result suggests that the meta-policy successfully models the sampling procedure and message passing under a joint learning framework.

Third, learning to sample different iterations of aggregation for different nodes is more effective than searching the neural architectures. Specifically, with only the basic *GCN* layers, our *Policy-GNN* is able to achieve state-of-the-art results by learning a proper aggregation strategy. Whereas, *GraphNAS* and *AGNN* only achieve marginal improvement over *GCN* by searching the neural architectures. The possible reason is that, NAS methods require manually defined search space with the same number of layers setting for all nodes. This makes the discovered architecture limited by the search space, and therefore perform similar to static-GNNs on all three datasets. Thus, we argue that more research efforts should be made on aggregation strategy since proper iterations of aggregation will significantly boost the performance.

Last but not least, the result of *Random Policy* demonstrates that reinforcement learning is a promising way to train the meta-policy in *Policy-GNN*. Specifically, we observe that the performance of *Random Policy* is significantly lower than *Policy-GNN* and most of the baselines. This suggests that the proposed *Policy-GNN* indeed learns an effective meta-policy for aggregation.

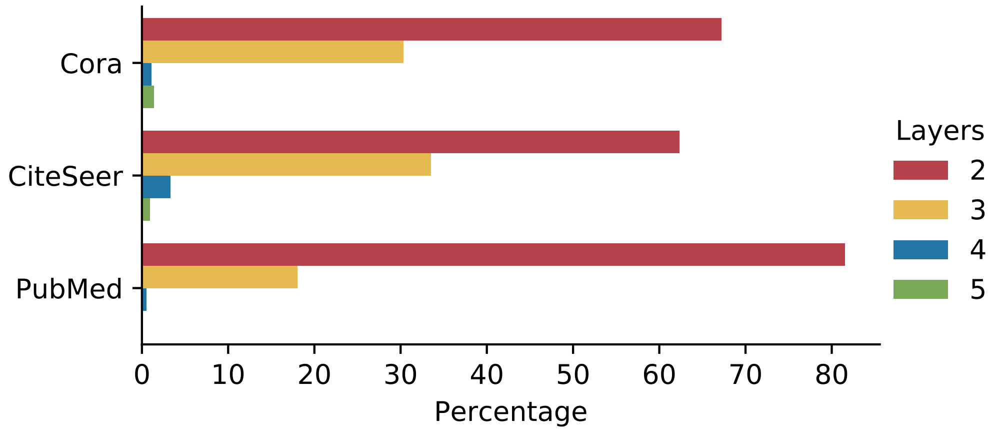
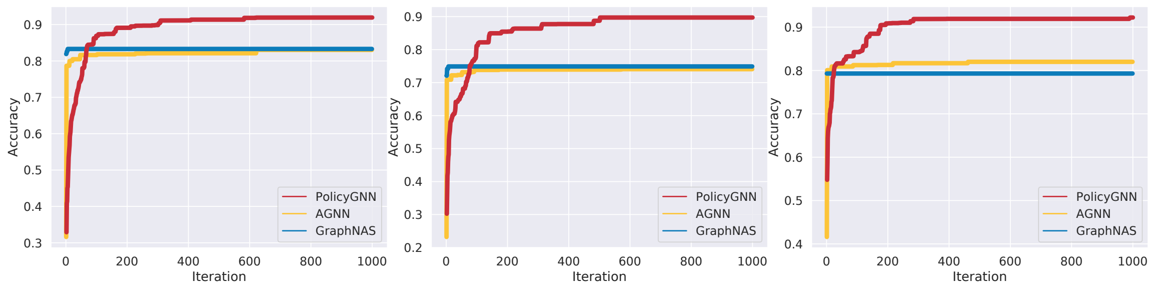


Figure 3.4 : The percentage of the nodes that are assigned to different layers of GCN by *Policy-GNN*.

3.4.5 Analysis of the Sampled Layers

To gain a deeper insight into the learned meta-policy, Figure 3.4 presents a visualization of the decision-making process, depicting the percentage of nodes assigned to different layers of the *GCN*. The results reveal that the majority of the nodes necessitate solely two layers of aggregation. Nevertheless, a certain proportion of nodes requires three or more layers of aggregation. For instance, in CiteSeer, more than 30% of the nodes require three iterations of aggregation, and a smaller subset of 5% of nodes necessitate four iterations of aggregation. Furthermore, the distributions of the iteration numbers exhibit diversity across the different datasets. These findings once again substantiate the indispensability of adopting a learning-based meta-policy to model the diverse characteristics of the data.



(a) Cora **(b)** CiteSeer **(c)** PubMed

Figure 3.5 : Learning curves of 1000 iterations. We plot the accuracy with respect to the number of iterations on the three datasets. Although *Policy-GNN* converges slightly slower than NAS methods, it achieves much higher accuracy later.

3.4.6 Analysis of the Learning Process

The plot in Figure 3.5 showcases the classification accuracy as a function of the number of iterations for our novel *Policy-GNN* framework and two existing neural architecture search baselines. As we scrutinize the results, we discern that *Policy-GNN* manifests a marginally slower learning pace than *AGNN* and *GraphNAS* in the initial stages. This may be attributed to the fact that determining the optimal number of iterations for the nodes is a more intricate task than searching for the neural architectures. Nevertheless, it is quite evident that *Policy-GNN* outperforms the NAS baselines by a considerable margin in the later stages, thereby affirming the merit of our aggregation strategy search in enhancing the efficacy of GNNs.

3.5 Conclusion

The challenge of customizing models on a sample-wise basis is tackled in this study by introducing *Policy-GNN*, a meta-policy framework that dynamically learns an aggregation policy for sampling varying iterations of aggregations for different nodes. The graph representation learning problem is modeled as a Markov decision process, and deep reinforcement learning with a customized reward function is utilized to train the meta-policy. The experimental results demonstrate the feasibility and superior performance of automated model customization.

Chapter 4

Data-centric Automated Knowledge Acquisition

4.1 Motivation

Imperfect datasets are a common occurrence in real-world applications, which often necessitates the need for human expertise in order to learn an effective model. This is particularly evident in challenging tasks like anomaly detection, where the available label information is extremely sparse. In most cases, an analyst is presented with a ranked list of anomalies and is tasked with investigating the top instances to identify as many true anomalies as possible before exhausting the allotted time budget. In practice, human expertise can be utilized to improve the anomaly detection process by allowing the analyst to adjust the decision functions. To this end, we explore a scenario in which the anomaly detector selects one instance at a time and queries the analyst for adjustments to the decision functions, thus enabling the incorporation of human knowledge into the anomaly detection process.

To tackle the problem, some re-ranking strategies have been proposed to approximate the above sequential decision process by greedily optimizing the immediate performance [81, 82, 83, 84]. This greedy choice may benefit the immediate performance; however, this approach may not always lead to optimal long-term performance. For instance, some instances with high uncertainty might provide crucial information to correct the anomaly patterns [85], which can be lower-ranked and may harm immediate performance. Still, these instances could improve the anomaly detector's performance in the long run, resulting in better anomaly discovery in future iterations. Therefore, automatically querying knowledge from human experts to improve the underlying model's long-term performance is a critical problem in data-centric AutoML that requires further investigation.

However, achieving this goal poses significant challenges. Firstly, quantifying the long-term performance is not straightforward. While we can predict the immediate outcome in the current iteration, i.e., the likelihood of an instance being anomalous or not, we cannot accurately estimate future benefits. Furthermore, balancing long-term and short-term performance is challenging and varies across different scenarios. Secondly, the decision space is typically large, requiring us to examine all instances and select one for querying, making the selection strategy difficult to design, especially in large or high-dimensional data. Lastly, different datasets have different data distributions and decision space sizes, necessitating a simple and transferable selection strategy that can be applied to different datasets, further adding to the challenge of designing the strategy.

Our proposed solution to overcome the aforementioned challenges is called Meta-AAD (Active Anomaly Detection with Meta-Policy), which is a meta-policy learning approach to optimize the number of anomalies detected. In this approach, active anomaly detection is formulated as a Markov decision process, and a deep reinforcement learning technique is used to train the meta-policy to select the best instance in each iteration. The optimization of the meta-policy is achieved by maximizing the discounted cumulative reward, which takes into account both short-term and long-term benefits.

4.2 Preliminaries

In this section, we firstly discuss the related works to the proposed framework, then we formulate the problem of active detection with meta-policy. After that, we provide a naive approach to training the meta-policy with DRL and discuss its limitations. The main symbols used in this chapter are summarized in Table 4.1.

Table 4.1 : Main Symbols and definitions.

Symbol	**Definition**
n	The number of instances.
d	The feature dimension of each instance.
l	The dimension of transferable features.
$\mathbf{X} \in \mathbb{R}^{n \times d}$	A dataset with n instances and d features.
$\mathbf{G} \in \mathbb{R}^{n \times l}$	Transferable features with dimension l.
$\mathbf{y} \in \mathbb{R}^{n}$	The n labels of dataset, where $\mathbf{y}_i \in \{-1, 1\}$.
$\hat{\mathbf{y}} \in \mathbb{R}^{n}$	The state vector, where $\hat{\mathbf{y}}_i \in \{-1, 0, 1\}$.
$\mathbf{c} \in \mathbb{R}^{n}$	The anomaly scores by an unsupervised detector.
$\mathcal{S}$	The state space in Markov Decision Process (MDP).
$\mathcal{A}$	The action space in MDP.
$\mathcal{R}$	The reward function in MDP.
γ	The discount factor in MDP.

4.2.1 Related Works

Anomaly detection

Anomaly detection has been extensively studied in the past decades, e.g., density-based approach [86], distance-based approach [87, 88], and ensembles [89, 90, 91]. Anomaly detection algorithms have also been developed for various types of data, such as categorical data [92], multi-dimensional data [89], time-series data [93] and graph data [94]. Most of these algorithms are unsupervised, with strong assumptions about the anomaly patterns [95]. However, these algorithms may not work well when the assumptions do not hold. On the contrary, our Meta-AAD rarely relies on the assumptions. It instead aligns anomaly patterns with human interests by leveraging human feedback

Semi-supervised anomaly detection

Semi-supervised learning methods [96, 97] have been studied in the context of anomaly detection. Semi-supervised anomaly detection assumes that a small set of labeled instances can be used to improve the performance [98]. In [99], a small set of anomalous instances are leveraged to re-weight the anomaly scores with belief propagation. [100] improves representation learning by using a few anomalous instances. [101] incorporates label information with support vector data description. AI2 [102] ensembles unsupervised and supervised anomaly detectors. AutoML methods use a set of labeled instances to perform automated algorithm selection and neural architecture search [66, 103]. More recently, [104] proposes a semi-supervised anomaly detection approach for deep neural networks. However, these methods are designed for batch setting, which could be sub-optimal in the active learning.

Active anomaly detection

Active learning in anomaly detection is much more challenging than traditional active learning [105, 106] because of the imbalanced data. Instead of assuming a batch of labeled data, active anomaly detection interacts with humans and recomputes the anomaly scores based on the feedback [81, 82, 107, 108]. These methods usually define an optimization problem based on the human feedback and re-weight the instances at each iteration. [109] proposes to adaptively adjust the ensemble for active anomaly detection. [83] proposes to incorporate feedback by leveraging online convex optimization to improve efficiency and simplicity. [110] proposes to use contextual multi-armed bandit and clustering techniques to identify the anomalies in attributed networks in an interactive manner. OJRANK [84] re-ranks the instances in each iteration based on the top-1 feedback. While these prior methods incorporate humans in the loop, they all adopt a greedy strategy to select the top-1 anomalous instance in each iteration, which fails to model long-term performance. Whereas, our Meta-AAD builds upon deep reinforcement learning, which inherently models and optimizes long-term performance. Moreover, the previous methods require complicated optimization to re-weight the instances in each iteration. On the contrary, the trained meta-policy of meta-AAD is easy to use since it can be directly applied to different datasets without further training or tuning.

4.2.2 Problem Formulation

We address the problem of anomaly detection where a dataset $\mathbf{X} = \mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n \in \mathbb{R}^{n \times d}$ is represented by a set of instances. Here, n represents the number of instances, and d represents the feature dimension. Each instance $\mathbf{x}_i$ is a d-dimensional vector $\mathbf{x}_{i,1}, \mathbf{x}_{i,2}, ...\mathbf{x}_{i,d}$, where $\mathbf{X}_{i,j}$ can be either real-valued or categorical. The ground-truths that correspond to the n instances in the dataset are represented by $\mathbf{y} \in \mathbb{R}^n$, where $\mathbf{y}_i \in -1, 1$. A label of -1 denotes an anomalous instance and a label of 1 denotes a normal instance. The goal of anomaly detection is to divide the instances into an anomaly set $A = \mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_a$ and a normality set $N = \mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_b$, where a and b are the number of anomalous and normal instances, respectively. Typically, the anomaly set A constitutes the minority of the data, i.e., $a \ll b$.

Conventional unsupervised anomaly detectors typically assign anomaly scores $\mathbf{c} \in \mathbb{R}^n$ to all instances in the set $\mathbf{X}$. This is achieved by learning a mapping $f : \mathbf{X} \to \mathbf{c}$, where lower scores indicate a higher likelihood of the instances being anomalous. Based on the anomaly scores, an anomaly ranking can be obtained, where anomalous instances are expected to have higher ranks than normal instances. However, the ranking is often not entirely accurate, as some higher-ranked instances could be normal, and some lower-ranked instances could be anomalous. Therefore, in practice, analyst (human) efforts are usually required to investigate the higher-ranked instances and determine whether they are genuinely anomalous or not.

Using the previous notations and concepts, we present a formal problem statement for active anomaly detection with meta-policy. We are given a dataset $\mathbf{X}$, and at each step, a meta-policy selects an instance $\mathbf{x}_i$ for querying, and a human labels it as anomalous or normal. The state vector $\hat{\mathbf{y}} \in \mathbb{R}^n$ corresponds to the n instances in the dataset. Here, $\hat{\mathbf{y}}_i \in \{-1, 0, 1\}$, where -1 indicates that the instance is selected for query and is an anomaly, 1 indicates that the instance is selected for query but is normal, and 0 suggests that the instance has not been queried yet. Initially, all instances have a state of 0, indicating that they have not been selected for query. The state of the selected instance will be updated to 1 or -1 after each query step, based on the human feedback. Our objective is to learn a meta-policy, trained from labeled datasets, to decide which instance to query at each step within a budget of T queries. We aim to maximize the number of true anomalies discovered among the chosen instances until the budget is exhausted. The meta-policy is defined as a mapping $\pi : \{\mathbf{X} \times \hat{\mathbf{y}}\} \rightarrow \{1, 2, ..., n\}$.

Learning meta-policy. Deep reinforcement learning algorithms have shown promise in various domains [36, 70] The idea of meta-policy learning is to train a reinforcement learning agent to make decisions with the objective of optimizing the overall performance of the task. Some recent studies about deep reinforcement learning have demonstrated the effectiveness of the meta-policy [53, 72, 71]. Some related studies in graph neural networks [111] and natural language processing [112] also show the effectiveness of meta-policy learning. In addition to the difference of objectives, these studies are limited to the same or parallel datasets. Whereas, we demonstrate that the meta-policy in Meta-AAD can be generally transferred across various datasets.

4.2.3 Limitations of a Naive Approach

A naive approach to training the meta-policy with deep reinforcement learning could involve treating the active learning process as a Markov Decision Process (MDP) by using the state vector and queried instance as the state and action, respectively. In other words, the state space $\mathcal{S}$ would be $\{\mathbf{X} \times \hat{\mathbf{y}}\}$ and the action space $\mathcal{A}$ would be $\{1, 2, ..., n\}$. By defining a suitable reward function, it is possible to model the process as an MDP and train a policy using deep reinforcement learning algorithms to optimize performance."

However, the feasibility of this approach is limited due to two factors. Firstly, the state and action spaces are excessively large. The state dimension and action dimension are $O(nd)$ and $O(n)$, respectively, as we need to observe the information of all n instances and select one of the n instances for query at each iteration. However, state-of-the-art deep reinforcement learning algorithms often perform poorly on large state and action spaces [113, 114]. Our preliminary experiments also indicate that the above naive method fails to train an effective meta-policy. Secondly, even if we manage to train a meta-policy, transferring it to another dataset becomes difficult because the state and action spaces differ across datasets. To be practical, a meta-policy should be transferable. Hence, we cannot directly apply this naive approach to our problem. In the following sections, we will explore how we can overcome these issues and achieve stable meta-policy training.

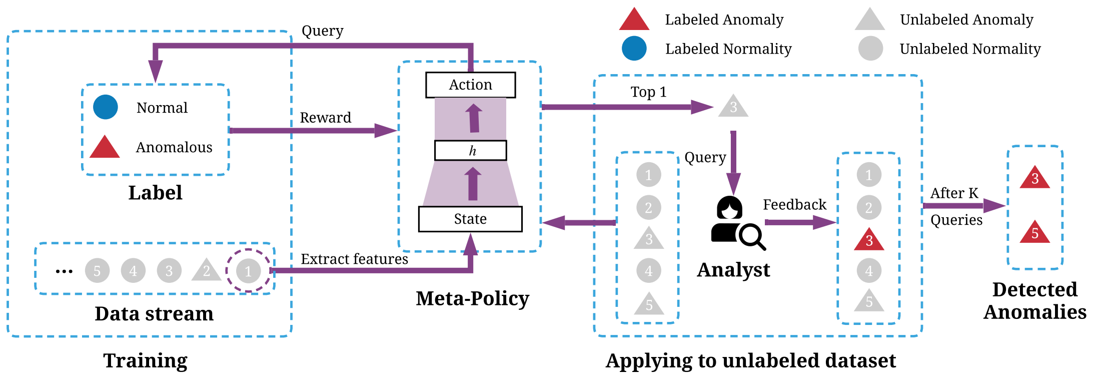


Figure 4.1 : An overview of Meta-AAD. In training, we shuffle the data and feed them to the meta-policy in a streaming manner. The meta-policy is rewarded based on the labels. The trained meta-policy can then be directly applied to a new unlabeled dataset. In each iteration, the meta-policy chooses one of the instances and queries an analyst (human).

4.3 Methodology

In this section, we provide a detailed description of our proposed approach, namely Active Anomaly Detection with Meta-Policy (Meta-AAD), as shown in Figure 4.1. First, we explain how we extract transferable features as states to reduce the state and action spaces (Section 4.3.1). Next, we describe how we stream the data in a shuffled manner during training to further reduce the state and action spaces (Section 4.3.2). We then train the meta-policy using deep reinforcement learning with labeled datasets (Section 4.3.3). Finally, we demonstrate that the trained meta-policy can be directly applied to new, unlabeled datasets for active anomaly detection without the need for further tuning (Section 4.3.4).

4.3.1 Extracting Transferable Meta-Features

The goal of this subsection is to derive meta-features that are transferable across different datasets. To achieve this, we aim to define a mapping $g : \{\mathbf{X} \times \hat{\mathbf{y}}\} \rightarrow \mathbf{G} \in \mathbb{R}^{n \times l}$, where $\mathbf{G}$ has a lower dependence on the dataset, and l is the dimension of the extracted features.

To make effective decisions about which instance to query, three types of information are important. First, the anomaly scores produced by the anomaly detector can indicate instances that are significantly different from the majority and assist the meta-policy in identifying more anomalous instances. Second, the labeled anomalous instances can be useful in identifying instances that are similar to known anomalies, and promoting these instances can improve performance. Third, the labeled normal instances can also provide useful information by discouraging instances that are similar to known normal instances and reducing false positives. To incorporate these intuitions, we extract a set of features empirically, consisting of a total of 6 features.

- **Detector features:** Unsupervised anomaly detectors output anomaly scores denoted as $\mathbf{c}$, which can be obtained using any existing anomaly detection algorithm.

- **Anomaly features:** The features that show the connection to the labeled anomalous instances are obtained using three features. Firstly, the original features $\mathbf{X}$ are standardized, and the minimum and mean Euler distances to the labeled anomalous instances are computed. Additionally, a binary feature is introduced to indicate the presence or absence of an anomalous instance in the k-nearest neighbors.

- **Normality features:** We adopt a similar approach for extracting normality features, where we use the minimum and mean Euler distances to the labeled normal instances.

It is worth noting that our framework provides flexibility in choosing features. For instance, we could potentially enhance performance by utilizing a combination of unsupervised anomaly detectors or by incorporating more detailed anomaly and normality features. However, for the sake of focus, we employ these basic features in all of our experiments, which yield satisfactory results based on our empirical evaluations. Further exploration into how to more effectively model transferable information is a promising avenue for future research aimed at enhancing the meta-policy.

In order to ensure that the feature dimension is consistent across different datasets, we map the original features to the transferable features described above, resulting in a feature dimension of l. However, the resulting features cannot be directly used for training, as each dataset may have a different number of instances n. In the next subsection, we will address this issue.

4.3.2 Learning from Data Streams

The transferable features $\mathbf{G} \in \mathbb{R}^{n \times g}$ and action space $\mathcal{A} = 1, 2, ..., n$ obtained in the previous section are too large for a learning algorithm to handle efficiently. Furthermore, the sizes of these spaces are proportional to the size of the dataset, which renders it impossible to transfer the meta-policy.

To facilitate the training of a transferable meta-policy, we propose working with data streams instead. Specifically, given the transferable features of the training data $\mathbf{G}^{train}$ and its corresponding labels $\mathbf{y}^{train}$, we randomly shuffle them in each episode to obtain a perturbation, denoted as $\mathbf{G}^{train'}$ and $\mathbf{y}^{train'}$. Instead of providing all the data to the meta-policy, we feed it with one instance at a time. In the streaming scenario, the Markov Decision Process (MDP) state, action, and reward are defined as follows.

- **State $\mathcal{S}$:**The transferable features corresponding to the currently observed instance can be denoted as $\mathbf{G}_i^{train'} \in \mathbb{R}^l$, where i represents the index of the instance.

- **Action $\mathcal{A}$:** The available actions are binary, with 1 indicating that the current instance should be selected and 0 indicating that the current instance should be ignored.

- **Reward $\mathcal{R}$:** When the meta-policy selects an instance, we assign a reward of 1 if the instance is actually anomalous, and a slightly negative reward of -0.1 if the instance is normal. If the meta-policy decides to ignore an instance, we assign a reward of 0. The reward function plays a crucial role in defining the intended behavior.

The MDP presented above outlines an active learning approach in a streaming setting. Essentially, the meta-policy is incentivized to select action 1 when the queried instance is anomalous and action 0 when the queried instance is normal. This helps the meta-policy learn to identify more anomalies within a given budget. We should note that the meta-policy trained in a streaming setting may not perform optimally when applied to a batch setting, as the two MDPs have distinct objectives. Nonetheless, we have found that in practice, the benefits of the streaming setting outweigh this concern. By reducing the state and action spaces, the streaming approach enables the successful training of a transferable meta-policy.

4.3.3 Training Meta-Policy with Deep Reinforcement Learning

The MDP described in Section 4.3.2 can be trained with a variety of deep reinforcement learning (DRL) algorithms. In this study, we implement the Proximal Policy Optimization (PPO) algorithm [38]. However, more sophisticated algorithms such as those discussed in [115] can be investigated in future.

The parametric policy $\pi_\theta(a|s)$ represents the meta-policy, where s is an l dimensional feature, $a \in \{0, 1\}$, $\sum_{a\in\{0,1\}} \pi(a|s) = 1$, and θ denotes the network parameters. The aim is to maximize the discounted cumulative reward $\mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_t]$. To achieve this, we utilize the Proximal Policy Optimization (PPO) algorithm, which belongs to the category of actor-critic algorithms. Here, the critic estimates the state values and the actor represents the policy.

The training procedure of the meta-policy is summarized in Algorithm 3. We assume the availability of several labeled datasets. In each episode, we randomly choose a dataset, shuffle the instances and traverse the dataset from the beginning in a streaming manner.

Algorithm 3 Training meta-policy with PPO

```
1:  Input: A set of features $\{\mathbf{X}^i\}_{i=1}^N$ and the corresponding labels $\{\mathbf{y}^i\}_{i=1}^N$,
    rollout steps $T$
2:  Output: The trained meta-policy
3:  Initialize meta-policy $\pi_\theta$, $\theta_{old} \leftarrow \theta$
4:  for iteration = 1, 2, ... until convergence do
5:      if iteration = 1 or episode is over then
6:          Randomly sample $\{\mathbf{X}', \mathbf{y}'\}$ from $\{\mathbf{X}^i\}_{i=1}^N$, $\{\mathbf{y}^i\}_{i=1}^N$
7:      end if
8:      Run $\pi_{\theta_{old}}$ with $\{\mathbf{X}', \mathbf{y}'\}$ based on the MDP defined in Section 4.3.1 for $T$
        timesteps.
9:      Compute advantages $\hat{A}_1, ..., \hat{A}_t$ based on Equation (2.3)
10:     Update $\theta$ based on Equation (2.5)
11:     $\theta_{old} \leftarrow \theta$
12: end for
13: return $\pi_\theta$
```

4.3.4 Application of Meta-Policy

After training the meta-policy, it can be directly applied to any new unlabeled datasets without further tuning. However, there are some major differences between the application and the training phases. Instead of feeding one feature at a time, we give all the features to the meta-policy to compute the probabilities for all the instances. When applying the meta-policy to an unlabeled dataset $\mathbf{X} \in \mathbb{R}^{n \times d}$, we first extract the transferable features $\mathbf{G} \in \mathbb{R}^{n \times l}$ from $\mathbf{X}$ and the current state vector $\hat{\mathbf{y}} \in \mathbb{R}^n$. Next, we compute the action probabilities $\pi_\theta(a = 1|\mathbf{G}_i), \forall i \in \{1, 2, ..., n\}$,

Algorithm 4 Application of trained meta-policy

```
1: **Input:** Unlabeled dataset $\mathbf{X} \in \mathbb{R}^{n \times d}$, trained meta-policy $\pi_\theta$
2: **Output:**The detected anomalies
3: Initialize state vector $\hat{\mathbf{y}} = \{0\}_{i=1}^{n}$, anomalous list $\mathbf{A} = \{\}$
4: for iteration = 1, 2, ... until budget is used up do
5:     Obtain transferable features $\mathbf{G} \in \mathbb{R}^{n \times l}$ from $\{\mathbf{X}, \hat{\mathbf{y}}\}$
6:     Compute $\pi(a = 1|s)$ based on $\mathbf{G}$ as $\mathbf{p} \in \mathbb{R}^{n}$
7:     Query the instance with the highest probability
8:     if the instance is anomalous then
9:         Put the instance into $\mathbf{A}$
10:    end if
11:    Update $\hat{\mathbf{y}}$ based on human feedback
12: end for
13: return A
```

and obtain the probability vector $\mathbf{p} \in \mathbb{R}^n$. We then choose the instance with the highest probability for query, i.e., $\arg\max_i \mathbf{p}_i$. Intuitively, the instance that is very likely to be selected in the streaming setting is also very likely to be chosen in this batch setting. The above procedure is summarized in Algorithm 4.

It is worth mentioning that $\pi_\theta(a = 1|\mathbf{G}_i)$ differs fundamentally from the adjusted anomaly scores used in previous methods [82, 81, 83, 84]. In those methods, anomalous scores are adjusted to promote anomalous instances to the top, with the primary goal of maximizing immediate performance by making the top-1 instance more likely to be anomalous. In contrast, the probability of the meta-policy has a significantly different function. The probability is learned with the objective of maximizing discounted cumulative reward, which combines immediate and long-term rewards. Therefore, the probabilities and the top-1 selection strategy inherently incorporate long-term performance considerations.

4.4 Experiment

In this section, we conduct extensive experiments to evaluate Meta-AAD. We mainly focus on the following research questions.

- **RQ1:** How does the meta-policy select the query and how will the decision of the meta-policy evolve in different stages (Section 4.4.2)?
- **RQ2:** How does Meta-AAD compare with the state-of-the-art alternatives and unsupervised baseline (Section 4.4.3)?
- **RQ3:** How will Meta-AAD perform if using different features, the number of labeled datasets and reward functions (Section 4.4.4)?
- **RQ4:** How many computational resources are needed to train a meta-policy (Section 4.4.5)?
- **RQ5:** How does Meta-Policy balance long-term and short-term reward (Section 4.4.5)?

4.4.1 Experimental Settings

Datasets and evaluation metric. To demonstrate the generality of Meta-AAD, we select 24 datasets with various sizes, feature dimensions and anomaly ratios from ODDS[*]. Table **??** summarizes the statistics of the datasets. We also use a toy dataset

[*]`http://odds.cs.stonybrook.edu/`

from [82] for better visualization. For the evaluation metric, we use anomaly discovery curve [110], which plots the number of discovered anomalies with respect to the number of queries. A perfect result is a line with a slope 1, i.e., all the queries are anomalous. The worst case is a line with a slope 0, i.e., all the queries are normal. Following [83], we set the maximum budget to be 100 for all the datasets.

Baselines. We compare Meta-AAD with the state-of-the-art methods as well as an unsupervised baseline as follows.

- **AAD.** Active Anomaly Detection [82] is a state-of-the-art method based on node re-weighting.
- **FIF.** Feedback-Guided Isolation Forest [83] is a recently proposed active anomaly detector via online optimization.
- **SSDO.** Semi-Supervised Detection of Outliers [116] is a recent semi-supervised point-wise anomaly detector. We are interested in studying how semi-supervised methods will perform in the active learning setting since they are also designed to leverage label information.
- **Unsupervised.** We also include Isolation Forest (IF) [89] as an unsupervised baseline.

While our Meta-AAD can be generally applied to any unsupervised anomaly detectors or an ensemble of detectors, for a fair comparison, we follow the previous work [82, 83] and use Isolation Forest (IF) [89] with the same hyper-parameters as in [82, 83]. For SSDO and the unsupervised baseline, we select the top-1 anomalous instance in each iteration.

Table 4.2 : Statistics of the datasets.

Dataset	**Points**	**Dim.**	**Anomalies**	**Anomaly%**
Annthyroid	7200	6	534	7.4
Arrhythmia	452	274	66	15.0
Breastw	683	9	239	35.0
Cardio	1831	21	176	9.6
Glass	214	9	9	4.2
Ionosphere	351	33	126	36.0
Letter	1600	32	100	6.3
Lympho	148	18	6	4.1
Mammography	11183	6	260	2.3
Mnist	7603	100	700	9.2
Musk	3062	166	97	3.2
Optdigits	5216	64	150	3.0
Pendigits	6870	16	156	2.3
Pima	768	8	268	35
Satellite	6435	36	2036	32.0
Satimage-2	5803	36	71	1.2
Shuttle	49097	9	3511	7.0
Speech	3686	400	61	1.7
Thyroid	3772	6	93	2.5
Vertebral	240	6	30	12.5
Vowels	1456	12	50	3.4
Wbc	278	30	21	5.6
Wine	129	13	10	7.7
Yeast	1364	8	64	4.7

Implementation details. For training the meta-policy, we use the PPO implementation in OpenAI baselines[†]. Following the default settings, we set rollout steps $T = 128$, entropy coefficient $c_2 = 0.01$, learning rate to be 2.5×10^{-4}, value function coefficient $c_1 = 0.5$, $\lambda = 0.95$, clip range $\epsilon = 0.2$. Recall that γ is hyper-parameters to balance long-term and short-term rewards. We empirically set $\gamma = 0.6$. We train the meta-policy with the top 12 datasets (in alphabetical order) and apply it to the bottom 12 datasets in Table **??**. We do it reversely to evaluate the top 12 datasets. The meta-policy is trained with 2×10^5 timesteps with the same hyper-parameters across all the datasets. The episode length is set to $2,000$. For the base detector of Isolation Forest, we use the implementation in sklearn[‡] with the default hyper-parameters setting. We use the original implementations of FIF[§], AAD[¶] and SSDO[‖] by their authors. For FIF, we try both linear and log-likelihood losses, and report the best result. For SSDO, we find it beneficial to use Isolation Forest for the query at the beginning and then switch to SSDO when we have hit at least one anomaly. We report the results with this strategy since we observe that it outperforms randomly selecting instances at the beginning. All the experiments are run 5 times. The average results and standard errors are reported.

4.4.2 A Case Study on the Toy Data

To investigate **RQ1**, we examine the development of Meta-AAD's decision-making process on a toy dataset [82] in Figure 4.2. The toy dataset comprises two-dimensional

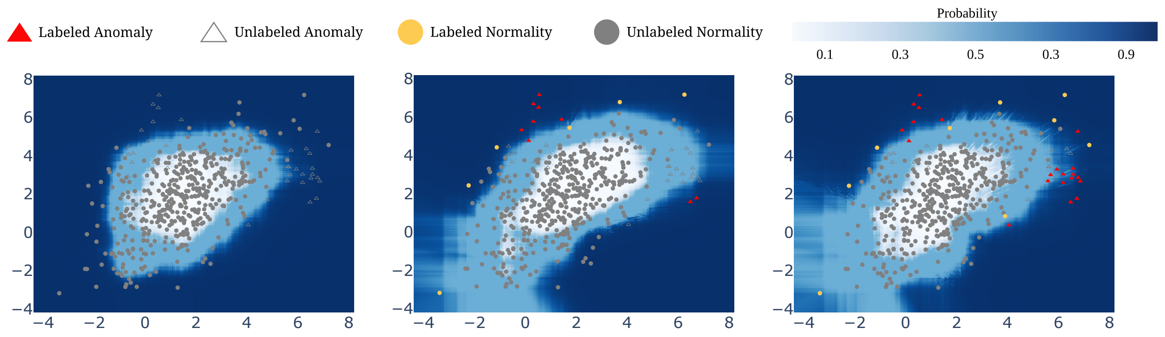
[†] `https://github.com/hill-a/stable-baselines`

[‡] `https://scikit-learn.org/`

[§] `https://github.com/siddiqmd/FeedbackIsolationForest`

[¶] `https://github.com/shubhomoydas/ad_examples`

[‖] https://github.com/Vincent-Vercruyssen/anomatools



(a) Initial state **(b)** 15 queries **(c)** 30 queries

Figure 4.2 : Evolution of the decision of Meta-AAD on toy data. Data in blue area are more likely to be presented to the analyst. In (a), the meta-policy prefers the instances that are far away from the majority, which is similar to an unsupervised anomaly detector. In (b) and (c), with more queries, the decision pattern evolves. The probability decreases in the regions around the normal instances (yellow). The probability increases for the regions around anomalies (red).

features, and we utilize the pre-trained meta-policy on the top 12 datasets in Table **??**. We plot the output of action 1 in the meta-policy, which is the probability of selection for the query. It is worth noting that the probability is akin to the anomaly score but is based on a distinct objective. We anticipate that the top instances should have excellent immediate performance, i.e., be highly likely to be anomalous, while benefit performance in the long term.

In the initial stage, the meta-policy prioritizes selecting instances that are distant from the majority, similar to how unsupervised anomaly detectors operate. It is assumed that the meta-policy has learned to give more importance to detector features when no labeled samples are available. As more queries are made, the decision-making process changes. On one hand, the likelihood of selecting instances close to normal instances (yellow instance on the bottom left corner) decreases. On the other hand, the probability of selecting instances around anomalies (red triangles on the right-hand side) increases. This change in pattern is in line with previous active anomaly detectors, such as those described in [82, 83]. Instead of adjusting anomaly scores, the meta-policy is optimized to maximize the discounted cumulative reward, which is more effective in modeling long-term performance compared to earlier methods.

4.4.3 Performance on Benchmark Datasets

To answer **RQ2**, we compare Meta-AAD against the baselines in the 24 real-world datasets. The anomaly discovery curves are illustrated in Figure 4.3. To better understand the performance, we rank the discovered anomalies of the four algorithms under 20, 40, 60, 80 and 100 queries, report the average rankings, and highlight the improvement of Meta-AAD over the second-best method in Table 4.3. We make the following observations.

To begin with, we note that all of the active anomaly detectors exhibit significantly improved performance over the unsupervised baseline and the semi-supervised method. Specifically, in 19 out of the 24 datasets, Meta-AAD, FIF, and AAD have been found to discover more anomalies while using the same number of queries, whereas they perform similarly in the remaining datasets. This is expected since the presence of labeled instances can provide informative cues that facilitate the identification of anomalies. We observe that SSDO, on the other hand, marginally outperforms the unsupervised baseline but falls far behind the active methods. A possible reason for this could be that SSDO is optimized for a different objective and, as a result, is sub-optimal in the active learning setting.

The second observation is that Meta-AAD consistently outperforms other state-of-

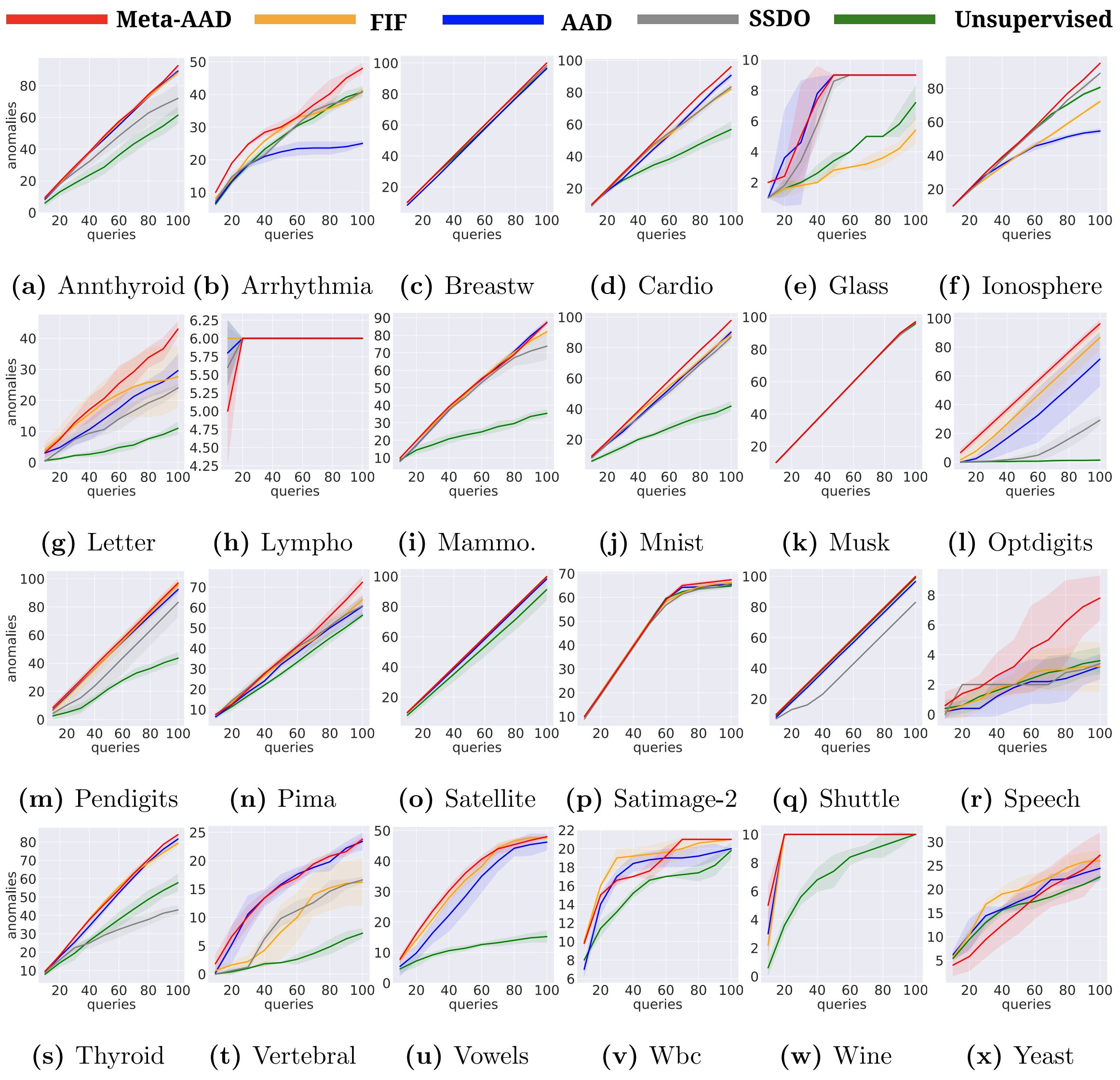


Figure 4.3 : Performance comparison of Meta-AAD against the state-of-the-art alternatives and unsupervised baseline.

the-art methods across all datasets. In most cases, Meta-AAD improves the anomaly detection performance compared to the baselines. For instance, Meta-AAD achieves over 25% improvement on Letter and Speech and more than 10% on Arrhythmia, Ionosphere, and Pima when compared with the best-performing alternative. Moreover, Meta-AAD either surpasses or achieves comparable results to other methods in the remaining datasets. The simplicity of using Meta-AAD is also noteworthy since it does not require any fine-tuning on target datasets. Thus, it is readily applicable in practical scenarios. Overall, the outcomes of the study substantiate the efficacy of training a meta-policy for active anomaly detection.

The third observation is that the Meta-Policy appears to be more effective in the long run. As shown in Table 4.3, the average ranking of Meta-AAD increases as more queries are made. Specifically, with 20 queries, the average ranking of Meta-AAD is 2.062, which is only slightly better than FIF. However, with 100 queries, the average ranking of Meta-AAD improves to 1.375. This suggests that Meta-AAD is better at modeling long-term rewards. It is possible that deep reinforcement learning inherently balances short-term and long-term performance, which is beneficial for anomaly detection in the long run.

4.4.4 Ablation Studies

In order to gain a deeper understanding of the factors contributing to our system's performance, we address **RQ3** through ablation studies focused on Annthyroid, Mammography, and Satimage-2, as depicted in Figure 4.4. Our analysis proceeds in three stages.

Firstly, we explore the impact of using different features by examining the contribution of the three types of features (detector, anomaly, and normality) to the

Table 4.3 : Average rankings of the number of discovered anomalies under different queries across 24 benchmarks, and the improvement of Meta-AAD over the second best state-of-the-art method. The improvement improves with more queries. Meta-AAD delivers stronger performance in long-term. ▲ denotes the cases where Meta-AAD is significantly better than the baseline w.r.t. the Wilcoxon signed rank test ($p < 0.01$).

Method	**20**	**40**	**60**	**80**	**100**
unsupervised [89]	4.188▲	4.146▲	4.167▲	4.333▲	4.375▲
SSDO [116]	3.312▲	3.396▲	3.500▲	3.625▲	3.438▲
AAD [82]	3.229▲	3.208▲	3.271▲	3.167▲	3.104▲
FIF [83]	2.208	2.333	2.312	2.396▲	2.708▲
Meta-AAD	**2.062**	**1.917**	**1.750**	**1.479**	**1.375**
Improvement	0.146	0.416	0.562	0.917	1.333

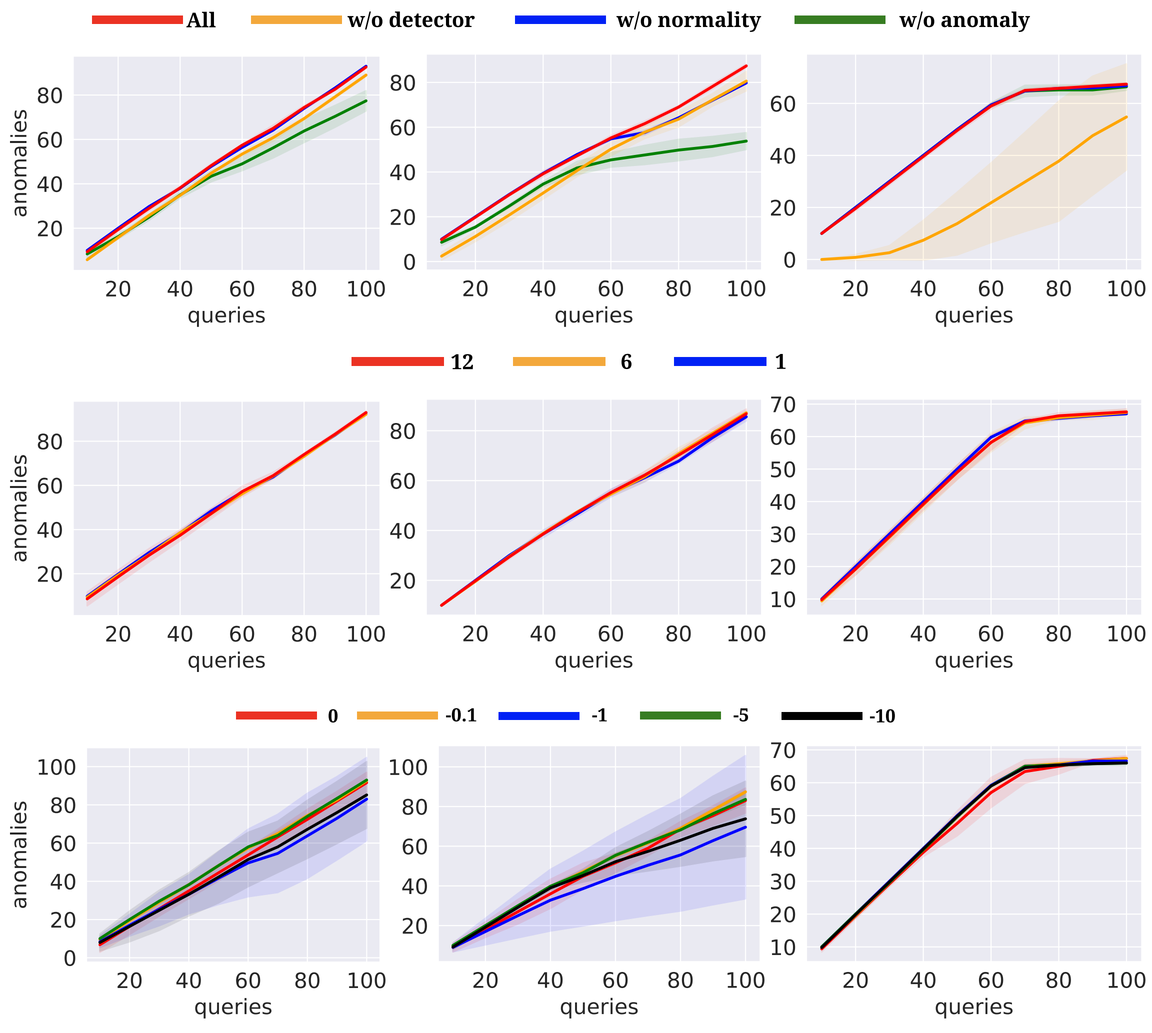


Figure 4.4 : Ablation study of Meta-AAD. We show the learning curves on Annthyroid, Mammography, Satimage-2 by dropping different features (top row), using different number of training datasets (mid row), and using different negative rewards for a missed query.

overall performance. The top section of Figure 4.4 displays the results of removing each feature type. Our findings indicate that all three types of features are essential for optimal performance, with the use of all three resulting in the best performance. This implies that the three types of features complement each other in training a high-quality meta-policy.

Secondly, we investigate the impact of varying the number of training datasets on the performance. By randomly dropping datasets and training the meta-policy on the resulting subset, we assess the performance when trained on 6 and 1 datasets, respectively (middle section of Figure 4.4). Our results reveal that while using more training datasets generally leads to greater robustness, we can still achieve a strong meta-policy even with just one dataset. Thus, our proposed features are transferable, and our training strategy is effective.

Finally, we examine how varying the rewards influences performance. Our rewards consist of a positive reward for discovering anomalies, a negative reward for selecting normal instances, and a neutral reward for not querying. We vary the negative rewards while keeping other rewards fixed (bottom section of Figure 4.4). Our findings suggest that the choice of rewards should depend on the context. For instance, a larger negative reward may be preferable when examining a difficult instance, whereas a smaller negative reward may suffice for easier instances. We observe that excessively large negative rewards can harm performance, with a small negative reward of -0.1 yielding optimal performance across the datasets.

In summary, our ablation studies indicate that the default choices of features and rewards deliver strong performance across different datasets, even with limited training data. Therefore, we conclude that Meta-AAD could be a versatile framework applicable to various scenarios.

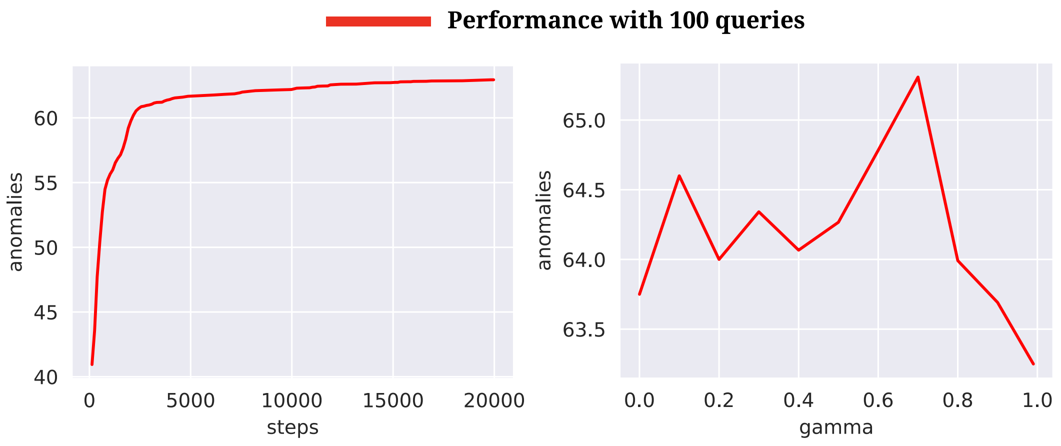


Figure 4.5 : The average discovered anomalies across all the datasets given 100 queries with respect to the number of training steps (left) and different γ values (right).

4.4.5 Analysis of the Meta-Policy

To further analyze our proposed method, we investigate two more research questions, namely **RQ4** and **RQ5**. In **RQ4**, we aim to determine the training efficiency of our meta-policy by examining the convergence rate. To achieve this, we plot the average performance with 100 queries across the 24 datasets with respect to the number of training steps of the meta-policy in the left-hand side of Figure 4.5. Our results show that the policy converges very fast, indicating that the training of the meta-policy is computationally efficient. In fact, we note that in a personal computer, it usually takes less than 30 seconds to train 20,000 steps with one process, which further reinforces the efficiency of our proposed method.

Moving on to **RQ5**, we investigate the impact of a hyper-parameter γ on the performance of our method. γ is used to balance short-term and long-term performance in the reinforcement learning process. We show the average performance with 100 queries using different γ in the right-hand side of Figure 4.5. We observe that giving too much preference for long-term or short-term rewards will both harm the performance. Therefore, we suggest that γ should be specified based on the specific needs of the application, i.e., whether we care more about long-term or short-term performance. In the conducted experiments, we set $\gamma = 0.6$ across all the datasets, as it was found to strike a good balance between short-term and long-term performance.

4.5 Conclusion

This study aims to address the challenge of automated expert knowledge acquisition by introducing Meta-AAD, a framework that incorporates human feedback into anomaly detection. The meta-policy in Meta-AAD is trained using deep reinforcement learning techniques to optimize performance in the long term. To evaluate the effectiveness of our approach, we test our framework on 24 different benchmark datasets. The results of our empirical analysis demonstrate that Meta-AAD performs better than existing state-of-the-art alternatives. We also conduct an extensive examination of our framework and observe that a single configuration can perform well across different datasets. Additionally, we find that Meta-AAD is capable of balancing long-term and short-term rewards, indicating its potential as a general framework for active anomaly detection.

Chapter 5

Model-aware Automated Data Preparation

5.1 Motivation

In many challenging real-world applications such as anomaly detection, label information is usually sparse, but not totally missing. Nevertheless, due to the sparsity of label information, unsupervised algorithms are often adopted to learn data patterns in industrial applications [117, 118]. However, unsupervised approaches rely on strong assumptions of data distribution and, therefore, may lead to a high false-positive rate when encountering unseen patterns [119]. To exploit the limited labels, label-informed learning methods [98, 120, 121, 122, 123, 124, 98] are proposed to exploit extra information from labels with tailored loss or scoring functions [120, 121, 122, 123, 124]. Still, the improvement may be limited without improving data quality. As such, a recent benchmark [23] finds that directly applying a supervised algorithm, such as XGBoost [125], can achieve comparable or even better performance than label-informed methods. To this end, extending the label information could be another promising direction toward challenging applications and scenarios.

Synthetic minority oversampling is one of the most popular solution for expanding label information. However, this may lead to noises due to the diverse distribution of real-world minorities. As shown in the left side of Figure 5.1, if we perform synthetic oversampling based on the anomalous account A and B, it is likely to create a synthetic account C labeled as anomalous but with moderate number of transactions

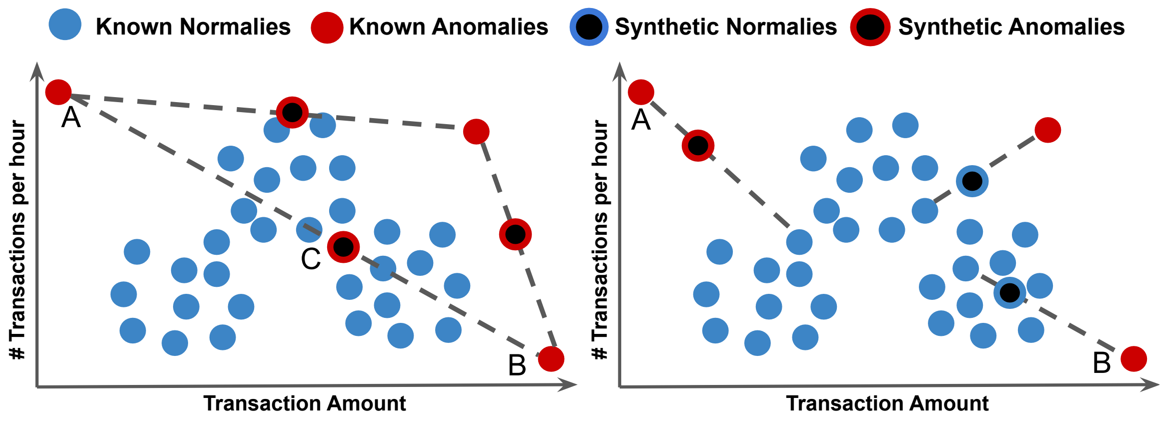


Figure 5.1 : Comparison between synthetic oversampling (left) and mix-up (right).

and transaction amount. As a result, account C interleaves with the normal accounts, which may degrade anomaly detection performance. In parallel, data mix-up [126] techniques in the image domain [127] have achieved great success in augmenting the training data by mixing up data samples from two different classes such that fine-grained supervisory signals of label differences between the two classes can be captured. The right of Figure 5.1 shows how mix-ups can help. A pair of normal and anomalous accounts with a properly chosen mix-up strategy can generate not only synthetic anomalies but also synthetic normalities to shape the boundary of anomalous and normal accounts. Thus, our objective is to extend the capacity of AutoML to automatically prepare data, especially in scenarios where the availability of labeled information is restricted and diverse.

However, it is challenging to develop a mix-up framework for the following reasons. First, it is challenging to identify the source samples for the mix-up. Randomly choosing two data samples for mix-up is likely to create noises due to the extreme imbalance between anomalies and normalities. The noisy samples can harm the classification performance. Additionally, even if we apply some strategies (e.g., only mixing up between anomalies), it is still likely to create noises due to the diverse distribution of anomalies. Second, even if we are able to get a proper pair of source samples, how to mix up remains to be a non-trivial problem. Randomly selecting a mix-up ratio may end up creating noises. We argue that the mix-up ratio needs to be tailored for the source samples. For example, if we select a pair of source samples with a normal sample in the center of normal instances and an anomaly that is surrounded by normal samples, a mix-up ratio that weighs more on anomaly samples should be selected. Conversely, for a pair of source samples with a normal sample that is deviated from most normalities and an anomaly that is nearby other anomalies, a mix-up ratio that weighs more on the normal sample may lead to a more informative synthetic sample. Third, even if we have a strategy to mix up data samples, simultaneously considering the underlying classifier and traversing the feature space to search for the most informative spot for synthetic sample generation is still challenging. For example, how can we know if the created sample can further benefit the formation of a model decision boundary? Likewise, if we are able to find a sweet spot for model training, how to guide the synthetic data generation process remains a challenging problem.

To tackle the challenges above, we propose MixAnN, a universal data mixer that is capable of incorporating different classifiers to exploit and explore potentially beneficial information from label information for supervised anomaly detection. Specifically, we propose an iterative mix-up process which traverses the feature space to create synthetic samples. Then, we formulate the iterative mix up into a Markov decision process (MDP) and design a reward function that provides model uncertainty and performance improvement to guide the policy learning procedure regardless of the convergence of the classifier. Finally, to solve the MDP, we tailor a deep actor-critical framework to learn for a discrete-continuous action space.

5.2 Preliminary

In this section, we firstly discuss the related work of our propsed framework. Then, we formally formulate our problem.

5.2.1 Related Works

Label-informed Anomaly Detection

Weakly/semi-supervised anomaly detection [128, 123, 121, 98, 120] are two main strategies to tackle the problem when either normal or anomaly samples are labeled. To leverage the large number of labeled normal samples, SAnDCat [128] selects top-K representative samples from the dataset as a reference to evaluate anomaly scores based on a model learned from pair-wise distances between labeled normal instances. On the other hand, to exploit a limited number of labeled anomalies, DevNet [123] enforces the anomaly scores of individual data instances to fit a one-sided Gaussian distribution for leveraging prior knowledge of labeled normal samples and anomalies. PRO [121] introduces a two-stream ordinal-regression network to learn the pairwise relations between two data samples, which is assumption-free on the probability distribution of the anomaly scores.

Recently, several endeavors [98, 120] further generalize the label-informed anomaly detection problem into semi-supervised classification setting [96, 129, 130] that limited numbers of both normal and anomaly samples are accessible. The main underlying assumptions are that similar points are likely to be of the same class and therefore densely clustered in low-dimensional feature space. XGBOD [98] extracts feature representation based on the anomaly score of multiple unsupervised anomaly detectors for training a supervised gradient boosting tree classifier. DeepSAD [120] points out that the semi-supervision assumptions only work for normal samples and further develops a one-class classification framework to cluster labeled normal samples while maximizing the distance between the labeled anomalies and the cluster in the high-dimensional space. However, weakly/semi-supervised learning methods focus on modeling the given label information without considering the relations between two labeled instances. Therefore, it is infeasible to generalize the label information when anomaly behaviors are diverse. By considering correlations between labeled samples and generating beneficial training data correspondingly, we are able to generalize the label information for training arbitrary classifiers.

Synthetic Oversampling for Imbalanced Classification

Data augmentation has been extensively studied for a wide range of data types [131, 132, 133] to enlarge training data size and generalize model decision boundaries for improving performance and robustness. Approaches to solving the imbalanced classification problem can be categorized into two types: algorithm-wise and data-wise. Algorithm-wise approaches directly tailor the loss function of classification models [134, 135, 136] to better fit the data distribution. However, modifying the loss function only facilitates the fitting of label information and may suffer from generalizing label information when the behavior between minority classes varies dramatically. Data-wise approaches generate new samples into the minority class [137, 138] or remove existing samples from majority classes. Synthetic Minority Oversampling (SMOTE) [137] generates new minority samples by linearly combining a minority sample with its k-nearest minority instances with a manually selected neighborhood size and the number of synthetic instances. A series of advancements on SMOTE [139] further introduce density estimation [138, 140], data distribution-aware sampling [141], and automated machine learning [142] to tackle the class imbalance problem without manual selection of neighborhood size and the number of synthetic instances.

Mix-up

Recently, instead of conducting synthetic data sampling on a single class, Mixup [143] achieves a significant improvement in the image domain by synthesizing data points through linearly combining two random samples from different classes with a given combination ratio and creating soft labels for training the neural networks. As Mixup assumes that all the classes are uniformly distributed for image classification, it does not apply when the class distribution is skewed. To tackle this limitation, Mix-Boost [144] introduces a skewed probability distribution to sample the ratio for linearly combining two heterogeneous samples. However, the imbalanced classification problem assumes that minority samples are clustered within the feature space, which may not be true when the minorities are anomalies. To this end, our work considers the attributes of a pair of normal and anomalous samples for jointly identifying the best k-nearest neighborhood and combination ratios. Then, we generate the synthetic samples with the combination ratios and identify the next pair of samples within the k-nearest neighborhood. This way, our framework is capable of exploiting the label information while exploring the diversely distributed anomalies.

5.2.2 Problem Statement

Following the setting of semi-supervised anomaly detection [123, 98, 120], a training dataset of supervised anomaly detection $\mathcal{X}^{\text{train}} = \{x_1, x_2, \ldots, x_{n+m}\}$ is composed of a labeled normalities $\mathcal{N} = \{x_1, x_2,$

$\ldots, x_n\}$ and a set of labeled anomalies $\mathcal{A} = \{x_{n+1}, x_{n+2}, \ldots, x_{n+m}\}$, where $n \gg m$.

The goal of supervised anomaly detection is to learn a classifier $\phi : \mathcal{X} \rightarrow \mathbb{R}$ which evaluates the probability of individual data points as anomalies in the given dataset $\mathcal{X}$ via exploiting the prior knowledge of labeled normalities $\mathcal{N}$ and anomalies $\mathcal{A}$, so that $\sum_{i \in \mathcal{A}} \phi(x_i)$ is maximized while $\sum_{j \in \mathcal{N}} \phi(x_j)$ is minimized with the constraints of $\phi(x_i) > 0.5$ and $\phi(x_j) \leq 0.5$.

To better generalize the knowledge from the label information, we formally define the problem of strategic data augmentation as follows. Given a dataset $\mathcal{X}^{\text{train}} = \{\mathcal{N}, \mathcal{A}\}$ where $\mathcal{X}^{\text{train}} \in \mathbb{R}^{(n+m) \times d}$, with a supervised classifier ϕ, we target on augmenting the $\mathcal{X}^{\text{train}}$ with a synthetic dataset $\mathcal{X}^{\text{syn}}$ according to the ϕ, where the synthetic dataset $\mathcal{X}_{\text{syn}} \in \mathbb{R}^{l \times d}$ is generated via mixing up samples from $\mathcal{N}$ with samples from $\mathcal{A}$. Specifically, our objective is to properly sample pairs of data instances from $\mathcal{N}$ and $\mathcal{A}$ with the corresponding mix-up ratio α to create synthetic instances $x^{\text{syn}} \in \mathcal{X}^{\text{syn}}$, such that the performance of ϕ can be maximized by being trained on $\hat{\mathcal{X}}^{\text{train}} = \{\mathcal{X}^{\text{train}} \cup \mathcal{X}^{\text{syn}}\}$.

5.3 Methodology

Figure 5.3 depicts an overview of our framework. We establish our framework by forming a classifier-driven Markov decision process with a data mixer that learns to generate synthetic samples for supervised anomaly detection. In this section, we elaborate on the details of the Markov decision process, including the reward signal for iterative data mix-up and a tailored sequential decision problem solver. Then, we provide algorithm and implementation details.

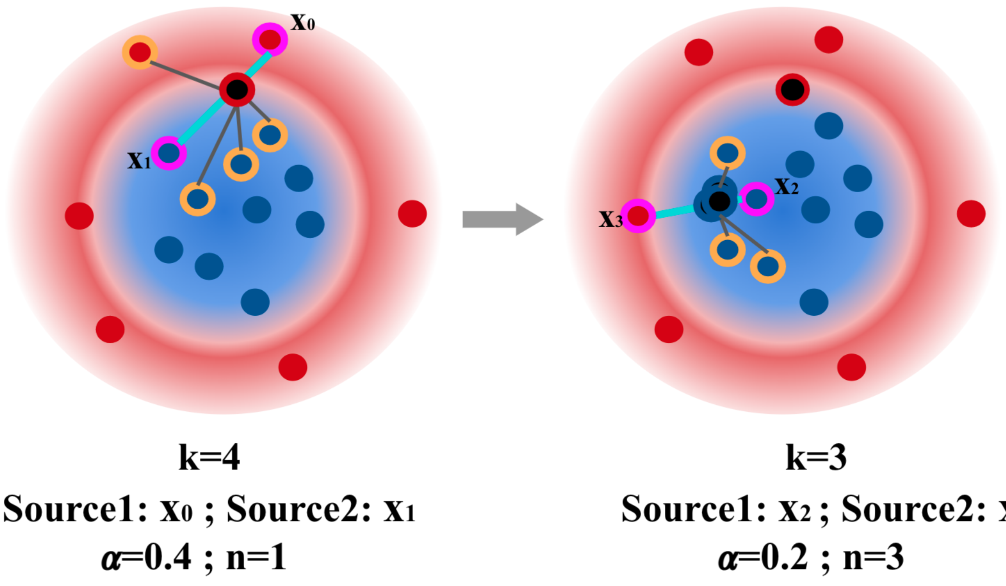


Figure 5.2 : An illustration of the iterative mix-up process. The background colors indicate the model decision boundary. The attributes of the source samples (purple circled) are considered to specify the composition ratio α for oversampling the synthetic sample (black points) with corresponding labels (outer circle of black points) n times. In the meantime, a k-nearest neighborhood is identified for randomly sampling the next pair of source samples. Finally, the process iterates to the next round with the new pair of source samples.

5.3.1 Iterative Mix-up Process

To generalize label information from two different classes, Mixup [143] performs synthetic data generation over two samples from different classes, which has been extensively studied to augment image and textual data. The core idea of Mixup is to linearly combine two samples as follows:

$$x_{\text{syn}} = \alpha * x_0 + (1 - \alpha) * x_1, \tag{5.1}$$

where x_0 and x_1 are the two selected source samples and $\alpha \in [0.0, 1.0]$ controls the composition of x_{syn}. Although existing works [143, 144] generate a soft label of x_{syn} in the same fashion for the imbalance classification problem, the diverse behaviors of the anomalies lead to similar labels with high granularity on diverse synthetic samples, which may prompt the model to over-fit on noisy synthetic labels. To this end, instead of generating soft labels, we synthesize hard labels for x_{syn} with a threshold η as follows:

$$y_{\text{syn}} = \begin{cases} y_0, & \alpha \geq \eta, \\ y_1, & \text{otherwise}, \end{cases} \tag{5.2}$$

where $0.5 \leq \eta \leq 1.0$. Due to the diverse behavior of anomalies, arbitrarily mixing up two random source samples from the X^{train} may lead to noisy samples. To tackle this problem, we seek to identify a meaningful pair of samples for synthesizing new samples. As normal samples are often concentrated in the latent space [120], inspired by the BorderlineSMOTE [141] that shows that borderline samples are more informative, we propose to traverse the feature space of X^{train} with the guidance of the decision boundary of the model ϕ for synthetic oversampling. Specifically, given a pair of arbitrary source samples, we consider the attributes of the two samples to identify corresponding α and number of oversampling for generating a set of $x_{syn} \in \mathcal{X}^{\text{syn}}$.

Meanwhile, an optimal range for uniformly sampling the next pair of source samples is identified according to the model impact for iterating to the next round of the mix-up process. The intuition behind uniform sampling is to consider the relationship between the attributes of the source samples and their entire neighborhood information instead of focusing on a certain sample in the neighborhood. Figure 5.2 illustrates an example of the iterative mix-up process. Based on the attributes of source samples x_0 and x_1, we first specify the composition ratio $\alpha = 4$ and the number of oversampling $n = 1$ for creating synthetic sample x_{syn}. Then, a k-nearest neighborhood is identified for sampling the next pair of source samples x_2 and x_3 for the next round of mix-up.

The proposed iterative mix-up process has several desirable properties. First, it can make personalized decisions. For example, we can generate more samples for some instances and fewer for other instances. Second, it can incorporate various information to guide the mix-up process. For instance, data attributes and model impact can be considered and serve as guidance for generating samples. Third, by simultaneously considering the model impact with the feature distribution, we directly generate information that is missing in the original dataset but beneficial for model training.

5.3.2 Formulating Iterative Mix-up as Markov Decision Process

The iterative mix-up process can be formulated as a Markov decision process with a quintuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ is the state transition function that maps the current state s, action a to the next state s', $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the immediate reward function that reflects the quality of action a for the state s, and γ is a decade factor to gradually consider the future transitions. Figure 5.3 illustrates the Markov decision process of

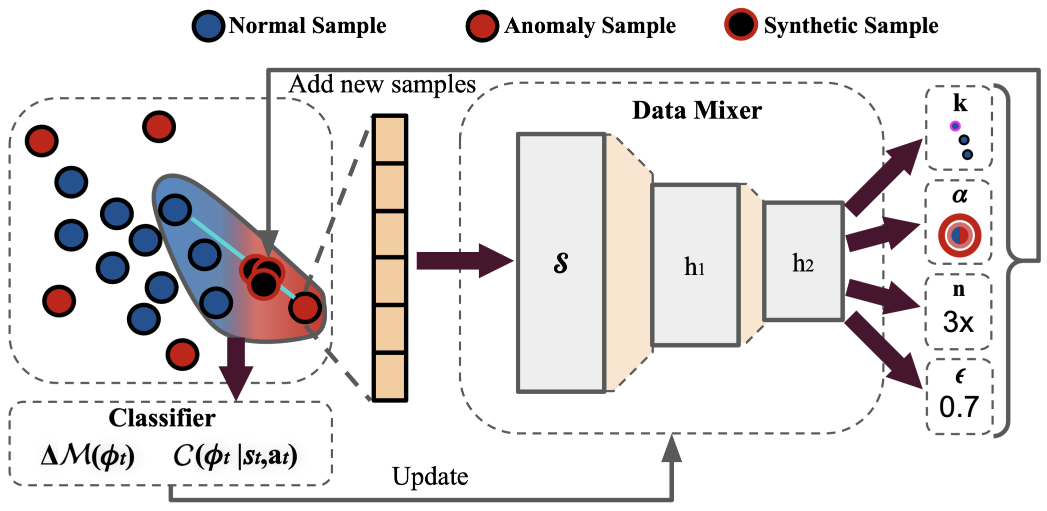


Figure 5.3 : An illustration of our framework. In each step, a pair of a normal sample and an anomaly is input to the data mixer. The corresponding action (i.e., neighborhood size k, composition ratio α, number of oversampling n, and termination probability ϵ) is then generated to create synthetic samples. After that, the synthetic samples are adopted to train the classifier and yield the reward signals (performance improvement $\Delta\mathcal{M}(\phi_t)$ and model confidence $\mathcal{C}(\phi_t|.)$) for updating the data mixer.

the framework. To sample-wise tailor the mix-up strategy, we target learning a data mixer that maps the attributes of source data samples into an augmentation strategy for creating synthetic samples while exploring model decision boundaries. The MDP can be defined as follows:

- **State Space ($\mathcal{S}$):** At each timestamp t, state $s_t \in \mathcal{S}$ is defined as $s_t = (x_0^t, x_1^t)$, where $s_t \in \mathbb{R}^{2m}$ is a concatenation of two m-dimensional feature vectors of the two source samples. Therefore, the state space is defined as $\mathcal{S} = \{(x_0^t, x_1^t)|x_0^t, x_1^t \in \mathcal{X}^{train}\}$.

- **Action Space ($\mathcal{A}$):** At each timestamp t, the action $a_t \in \mathcal{A}$ where $a_t = (k, \alpha, n, \epsilon)$ is a vector composed of the size of neighborhood k, composition ratio α, number of oversampling n and the termination probability of the iterative mix-up process ϵ. Therefore, the action space is defined as a discrete-continuous space $\mathcal{A} = \{(k_t, \alpha_t, n_t, \epsilon_t) | k_t, n_t, \epsilon_t \in \mathbb{N}, \alpha_t \in \mathbb{R}^+\}$.

- **Transition Function ($\mathcal{T}$):** Given a state $s_t = (x_0^t, x_1^t)$ and an action $a_t = (k, \alpha, n, \epsilon)$, the transition function firstly adopts α with Eq. 5.1 and Eq. 5.2 to oversample x_{syn} for n times. Then, the resulting synthetic samples $\mathcal{X}_{\text{syn}}$ will be adopted for training the classifier and lead to a classifier ϕ_t in timestamp t. Finally, the transition function shifts to the next state $s_{t+1} = (x_0^{t+1}, x_1^{t+1})$ where the x_0^{t+1} is randomly sampled from the k-nearest neighborhood of the x_{syn} and the x_1^{t+1} is identified as the nearest data point with a different label from x_0^{t+1}.

- **Reward Function ($\mathcal{R}$):** The reward signal r_t for each timestamp t is designed to encourage performance improvement while exploring the decision boundaries of the classifier ϕ. Therefore, the reward function is defined as:

$$\mathcal{R}(s_t, a_t) = \lambda * \Delta\mathcal{M}(\phi_t) * \mathcal{C}(\phi_t | s_t, a_t),$$

where λ is a hyperparameter to define the strength of the reward signal, $\mathcal{M}$ is an evaluation metric, and $\Delta\mathcal{M}(\phi_t)$ measures the performance improvement of ϕ_t. The $\mathcal{C}(\phi_t | s_t, a_t)$ evaluates the model confidence to encourage exploring the decision space of ϕ_t. This way, the reward signal drives the data mixer to explore the classifier while achieving maximum improvement with the newly synthesized data samples. The implementation details are provided in section 5.3.4.

5.3.3 Solving Markov Decision Process with Deterministic Actor-Critic

To solve the MDP above, we define a parameterized policy π_θ as the data mixer to maximize the reward signal of the MDP where the ultimate goal is to learn an optimal policy π_θ^* that maximize the cumulative reward $\mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_t]$. However, the action space of the iterative mix-up process is a discrete-continuous vector, and the reward signals generated from an under-fitted ϕ_t may be in-stable. To this end, we employ the deep deterministic policy gradient (DDPG) [37], an actor-critic framework that equips with two separate networks: actor and critic. The critic network $Q_(s_t, a_t|\theta_2)$ approximates the reward signal for a state-action pair from the MDP, while the actor network $\pi_(s_t|\theta_1)$ aims to learn the policy for a given state s_t based on the critic network. We note that an advanced actor-critic framework such as soft actor-critic [115] could be adopted to learn the π^*_θ, which will be our future exploration.

To perform a continuous action, DDPG learns an actor network $\pi(s_t|\theta_1)$ that deterministically maps a given state s_t to an action vector a_t and trains the network by maximizing the approximated cumulative reward that generated by the critic network $Q(.|\theta_2)$. Specifically, given N transitions, a projected action $\pi(s_t|\theta_1)$ can be generated as the input of the critic to minimize the following loss function:

$$L_\pi(\theta_1) = -\frac{1}{N}\sum_{i=1}^{N} Q(s_i, \pi(s_i)|\theta_2), \tag{5.3}$$

where the action $\pi(s_i)$ is a 4-dimensional real number vector. To fully leverage the expressive power of the deep neural network during the training while outputting a discrete-continuous vector for the MDP, we transform the continuous action vector $\pi(s_i)$ with a sigmoid function and yield the action vector $a_t = w \circ \sigma(\pi(s_i|\theta_1))$ where w specifies the value constraints of individual entries. For instance, if the maximum for the k and n are 10 and 5, then $w = [10, 1, 5, 1]$ since α and ϵ are expected to be ranging from 0 to 1. The outcome for the k and n are rounded to the nearest integer.

To tackle the in-stable reward signal issue, DDPG approximates the reward signal with the critic network $Q_(.|\theta_2)$ and trains the networks in an off-policy fashion. It introduces a replay buffer to store historical and randomly sampled transitions to minimize the temporal correlation between two transitions for learning across a set of uncorrelated transitions. Specifically, the critic network $Q_(.|\theta_2)$ maps a state-action pair into a real value y_t via minimizing the following loss function:

$$L_Q(\theta_2) = [Q(s_t, a_t) - b_t]^2 \tag{5.4}$$

Here, $b_t = \mathcal{R}(s_t, a_t) + \gamma Q(s_{t+1}, \pi(s_{t+1}|\theta_1)|\theta_2)$ is a signal derived from the Bellman equation [114] which considers the recursive relation between the current reward and the future approximated reward signals for maximizing cumulative reward where $\pi(s_{t+1}|\theta_1)$ is an action specified by the actor network and the γ is the decade factor.

5.3.4 Algorithm Details

Algorithm 5 illustrates the training procedure. We design two components to form the reward signal: improvement stimulation $\Delta\mathcal{M}(\phi_t)$ and model exploration $P(\phi_t|s_t, a_t)$. To learn an optimal policy for the target tasks, existing solutions [124, 145, 146] directly adopt the performance on a validation dataset as a reward signal. However, as the convergence of the underlying classifier is not guaranteed, directly learning a policy with the performance on a validation set may lead to noisy reward signals. As a result, rather than using the current model's performance ϕ_t, we design an improvement stimulation to pursue the maximum model improvement on the validation set with a baseline performance:

$$\Delta\mathcal{M}(\phi_t) = \mathcal{M}(\phi_t(\mathcal{X}^{\text{val}}), y^{\text{val}}) - \frac{\sum_{i=t-m}^{t-1} \mathcal{M}(\phi_i(\mathcal{X}^{\text{val}}), y^{\text{val}})}{m-1}, \tag{5.5}$$

where $\frac{\sum_{i=t-m}^{t-1} \mathcal{M}(\phi_i(\mathcal{X}^{\text{val}}), y^{\text{val}})}{m-1}$ is a baseline performance for the timestamp t, and m controls the sample size for the baseline.

In addition, as we aim at creating synthetic samples by mixing normal samples and anomalies while iteratively training the classifier ϕ, it is critical to explore model decision boundaries to create beneficial samples and prevent generating noisy samples. Inspired by the confidence quantification [147] for anomaly detectors, we develop a model exploration signal to quantify the instance-wise prediction uncertainty as follows:

$$\mathcal{C}(\phi_t|s_t, a_t) = \frac{1}{k}\sum_{i=0}^{k} P(y_i = 0|x_i, \phi_t)P(y_i = 1|x_i, \phi_t), \tag{5.6}$$

where x_i is the k-nearest neighbor of the synthesized sample in timestamp t because k is the size of the nearest neighborhood specified by a_t. The intuition behind this is to encourage the data mixer to explore the uncertain area in the feature space.

5.4 Experiment

Our experiments aim to answer the following research questions:

- **RQ1**: How does the proposed framework compare against the existing data augmentation methods? (Section 4.2)
- **RQ2**: How does the proposed framework compare against the existing label-informed anomaly detection methods? (Section 4.3)
- **RQ3**: How does each component contribute to the performance of the proposed framework? (Section 4.4)
- **RQ4**: How do the key hyperparameters affect the model performance? (Section 4.5)
- **RQ5**: How does the learned strategy of the proposed framework compare with the existing approaches? (Section 4.6)

5.4.1 Experiment Settings

We conduct horizontal analysis to compare the proposed framework with data augmentation methods on three different classifiers. We also conduct a vertical analysis that compares the proposed framework with label-informed anomaly detectors. The details of the experimental settings are provided as follows:

	# Samples	# Features	% Anomaly	Domain
Japanese Vowels	1,456	12	3.43%	Utterance
Annthyroid	7,200	22	7.42%	Clinical Record
Mammography	11,183	6	2.32%	Medical Image
Satellite	6,435	36	31.63%	Remote Sensing
SMTP	95,156	41	0.03%	Server Log
CIFAR-10	5,263	512	5.00%	General
FashionMNIST	6,315	512	5.00%	Fashion

Table 5.1 : Dataset statistics.

Datasets: Table 5.1 summarizes the statistics of the datasets. We adopt 5 benchmark datasets from different domains:

- **Japanese Vowels** contains utterances of /ae/that were recorded from nine speakers with 12 LPC cepstrum coefficients. The goal is to identify the outlier speaker.

- **Annthyroid** is a set of clinical records that record 21 physical attributes of over 7200 patients. The goal is to identify the patients that are potentially suffering from hypothyroidism.

- **Mammography** is composed of 6 features extracted from the images, including shape, margin, density, etc. The goal is to identify malignant cases that could potentially lead to breast cancer.

- **Satellite** contains the remote sensing data of 6,435 regions, where each region is segmented into a 3x3 square neighborhood region and is monitored by 4 different light wavelengths captured from the satellite images of the Earth. The goal is to identify regions with abnormal soil status.

- **SMTP** has 95,156 server connections with 41 connection attributes including duration, src_byte, dst_byte, and so on. The task is to identify malicious attacks from the connection log.

- **CIFAR-10 and FashionMNIST** are the benchmark datasets. We follow the setting of previous works [23, 120] to set one of the classes as normal and downsample the rest classes to 5% of the total instances as anomalies. We report the average performance of airplane and automobile for CIFAR-10; and the average performance of t-shirt/top and trouser for FashionMNIST.

All of the dataset are public available in OpenML [*][148] and ADBench [†]. We follow the widely adopted setting [120, 123, 23] with the pre-processing proposed by ODDS [149] and ADBench [23] to process the data.

Horizontal Baselines: We conduct a horizontal comparison between the proposed methods and the following representative data augmentation methods on 3 different classifiers (i.e., KNN, XGBoost, MLP): **Random** is a basic baseline which generates synthetic anomalies by randomly averaging two existing anomalies. **SMOTE** [137]

[*]`https://www.openml.org/`

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

linearly combines existing anomalies with their K-nearest anomalies through randomly sampled combination ratios. **BorderlineSMOTE** [141] identifies a borderline between anomalies and normal samples with the K-nearest neighborhood of each anomaly. Then, SMOTE is performed on the anomalies in the borderline area. **SVMSMOTE** [150] introduces a support vector classifier to identify a borderline between anomalies and normal samples and perform SMOTE on anomalies near to the borderline. **ADASYN** [138] calculates the number of synthetic samples generated from individual anomalies using the estimated local distribution and then performs SMOTE on them. **AutoSMOTE** [142] performs SMOTE by exploiting deep reinforcement learning to automatically select the anomalies and the corresponding oversampling strategies. **MixBoost** [144] uses a randomly sampled combination ratio to mix anomalies and normals. The number of oversamples for individual anomalies is weightily sampled based on the entropy of the underlying classifier.

Vertical Baselines: We perform a vertical comparison between the proposed method and label-informed detection algorithms to study the effectiveness of synthetic oversampling. **XGBOD** [98] exploits the anomalous scores of individual data points generated from multiple unsupervised algorithms as the input feature for the underlying XGBoost classifier. **DeepSAD** [120] develops a semi-supervised loss function that classifies the known normal samples and unlabeled samples into a unified cluster and deviates the known anomalies from the cluster. **DevNet** [123] uses label information with a prior probability to enforce significant deviations in anomaly scores from the majority of normal samples.

Evaluation Protocol: We adopt macro-averaged precision, recall, and F1-score, which compute the scores separately for each class and average them. The intuition behind this is to equalize the importance of anomaly detection and normal sample classification since the minimum false alarms are also a critical evaluation criterion. We use 80% data for training and 20% for testing, where 20% of the training data is further split into a validation set for our framework to generate reward signals or for baseline methods to perform model tuning. We run the experiments with 5 random seeds and report the average performance on the testing set.

Implementation Details: For the horizontal analysis, we use a KNN classifier with $k = 10$, an XGBoost classifier with the linear kernel, and the Adam optimizer with ReLU activation function for a 128-64 multi-layer perceptron classifier. For the vertical analysis, we adopt XGBOD [‡] and public available implementations of DevNet [§] and DeepSAD [¶]. Since the output of Dev and DeepSAD are anomaly scores, we search for the thresholds for the two methods from $\{0.5x, 1.0x, 1.5x, 2.0x\}$ of the anomaly ratio to perform classification and report the best result. For our own method, we select the maximum neighborhood size K from $\{5, 10, 15, 20, 25\}$, and set $\eta = 0.3$, the reward coefficient $\lambda = 10.0$, the window size for baseline $m = 25$ and adopt the macro-averaged F1-score for the reward signal $\Delta\mathcal{M}$.

5.4.2 Horizontal Analysis

To answer **RQ1**, we compare the MixAnN to cutting-edge synthetic oversampling methods for anomaly detection. Table 5.2 tabulates the macro-averaged precision, recall, and F1-score of each data augmentation method across 3 different classifiers. We also report the performances without data augmentation on the original classifiers to show insights into how different classifiers impact the performances. In general, the proposed MixAnN is able to outperform all of the baseline oversampling methods

[‡]`https://github.com/yzhao062/pyod`

[§]`https://github.com/GuansongPang/deviation-network`

[¶]`https://github.com/lukasruff/Deep-SAD-PyTorch` and achieve at least 7.6% and at most 33.3% improvements on the F1-score. Based on Table 5.2, we make the following observations.

First, by comparing the baseline augmentation methods with the classifiers without data augmentation, we observe that the performance of the baseline augmentation methods is generally inferior to the classifier trained without data augmentation. Specifically, the average F1-scores of the baseline augmentation methods are consistently lower than the vanilla classifiers on the five datasets. The only exception is the Mixboost with KNN classifier, which is due to its decent performance on the SMTP dataset. Our further investigation into this phenomenon suggests that randomly mixing up normal samples with anomalies can create beneficial synthetic normal samples that concrete the decision boundary when anomalies are extremely sparse. This supports our claim that the existing data augmentation methods are not capable of handling the diverse behavior of anomalies and may lead to noisy synthetic samples. But generalizing label anomaly information by mixing normal samples with anomalies could alleviate the problem.

Second, by comparing the MixAnN with the vanilla classifier without data augmentation, we observe that the MixAnN consistently outperforms all vanilla classifiers. On the five datasets, the MixAnN improves the F1-score of KNN, XGBoost, and MLP classifiers by 9.6%, 14.2%, and 11.5%, respectively. This phenomenon suggests that the MixAnN is able to adaptively create synthetic samples for different classifiers toward performance improvements. In addition, we also observe that the KNN classifier has the maximum improvement, which suggests that the nearest-neighbor exploration of our transition function favors the classifier with similar attributes. Another interesting observation is that the more complex the models, the fewer the improvements. A possible explanation is that complex models tend to be overconfident in the prediction [151], which may lead to noisy prediction uncertainty reward and mislead the learning procedure of the augmentation strategy. We will study this issue in the future.

Third, by comparing the MixAnN with all other data augmentation methods, we observe that the MixAnN outperforms all baselines with the three classifiers on Macro-F1 scores. Specifically, the MixAnN averagely outperforms the F1-score of the second-best augmentation method with KNN, XGBoost, and MLP classifiers on the seven datasets by 4.9%, 10.1% and 7.4%, respectively. Because Mixboost is similar to MixAnN with a random mixture policy, it implies that the proposed framework can learn tailored mix-up policies for different classifiers and data samples. We can also observe that, although the MixAnN is not always superior to all other baselines on precision and recall, the F1-scores are always the best. This phenomenon suggests that MixAnN is able to balance the trade-off between precision and recall, and therefore leads to superior F1-scores in all settings. The main reason behind this is that we adopt the Macro-F1 to form a reward signal. One may also tailor their own metrics (e.g., precision, recall, or tailored metrics) to obtain an anomaly detector that meets their requirements.

Fourth, by taking a detailed comparison between the MixAnN with SVMSMOTE and BorderlineSMOTE, we observe that the F1-score of the MixAnN is superior to the two baselines by at least 14.9%, 14.2% and 14.5% with the three classifiers respectively. This phenomenon suggests that the instance-wise prediction uncertainty in the reward function is a better approach to generating tailored beneficial synthetic samples for different classifiers. The rationale behind this is that the two baselines identify the class boundary in the label space and the hyperspace of the SVM, where the MixAnN identifies the boundary that is directly defined by the underlying classifier. By encouraging the policy to generate samples on the boundary defined by the classifier, it is more likely to create beneficial information that cannot be observed from the original feature space or the hyperspace of another classifier.

Algorithm 5 Training Procedure

```
1: Input: Input data $X^{\text{train}}$, maximum neighborhood size $K$, number of training episode $E$, DDPG training step $S$, reward coefficient $\lambda$, window size for baseline reward $m$.
2: Initialize classifier $\phi$, actor network $\pi$, critic network $Q$, memory buffer $\mathcal{B}$, $\epsilon = 0$.
3: Split partial $X^{\text{train}}$ to form $X^{\text{val}}$ and train $\phi$ on $X^{\text{train}}$.
4: Pre-compute $K$-nearest neighborhood for $X^{train}$.
5: Randomly sample a pair of source samples to form the initial state $s_0 = (x_0, x_1)$.
6: for $e = 0, 1, 2..., E$ do
7:     $t \leftarrow 0$.
8:     while $\epsilon < 0.5$ do
9:         Get the action $a_t = \pi(s_t)$ from actor network.
10:        Get $\alpha$, $n$ from $a_t$ and adopt Eq. 5.1, Eq. 5.2 to generate $x_{\text{syn}}$.
11:        Update $\phi$ with $x_{\text{syn}}$ and get $k$ from $a_t$.
12:        Obtain $k$-nearest neighborhood of $x_{\text{syn}}$.
13:        Form the reward $r_t = \lambda\Delta\mathcal{M}(.)\mathcal{C}(.)$ with $X^{val}$ and the $k$-nearest neighborhood of $x_{\text{syn}}$.
14:        Randomly choose next pair of source samples $(x_0^{t+1}, x_1^{t+1})$ from the $k$-nearest neighborhood of $x_{\text{syn}}$
15:        Form the next state $s_{t+1} = (x_0^{t+1}, x_1^{t+1})$ and get $\epsilon$ from $a_t$.
16:        Store the triplet $T_t = (s_t, a_t, s_{t+1}, r_t)$ into $\mathcal{B}$.
17:        $t \leftarrow t + 1$.
18:        for step = 1, 2, .., $S$ do
19:            Use data in $\mathcal{B}$ to update DDPG with Eq. 5.3 and Eq. 5.4.
20:        end for
21:    end while
22: end for
```

Classifier	Methods (Prec./Rec./F1)	Dataset							Avg. F1 ↑
		Vowels	Annthyroid	Mammography	Satellite	SMTP	CIFAR-10	FashionMNIST	
KNN	w/o Augmentation	0.98/0.72/0.82	0.93/0.61/0.67	0.86/0.74/0.79	0.90/0.88/0.89	1.00/0.75/0.83	0.92/0.54/0.56	0.96/0.82/0.87	+9.6%
	Random	0.87/0.82/0.84	0.67/0.63/0.65	0.64/0.77/0.68	0.90/0.88/0.89	0.51/0.99/0.51	0.55/0.55/0.55	0.85/0.85/0.85	+19.7%
	SMOTE	0.82/0.99/0.88	0.61/0.70/0.61	0.65/0.91/0.71	0.89/0.88/0.89	0.52/1.00/0.53	0.56/0.74/0.56	0.81/0.91/0.85	+18.1%
	BorderlineSMOTE	0.85/0.99/0.91	0.61/0.72/0.63	0.67/0.92/0.73	0.90/0.88/0.89	0.52/1.00/0.53	0.58/0.75/0.60	0.81/0.91/0.86	+14.9%
	SVMSMOTE	0.85/0.99/0.91	0.64/0.72/0.66	0.70/0.92/0.76	0.90/0.88/0.89	0.53/1.00/0.56	0.59/0.72/0.62	0.84/0.91/0.87	+13.3%
	ADASYN	0.82/0.99/0.88	0.59/0.71/0.60	0.63/0.92/0.69	0.90/0.88/0.89	0.55/1.00/0.60	0.56/0.73/0.55	0.79/0.92/0.83	+19.7%
	AutoSMOTE	0.94/0.94/0.94	0.67/0.75/0.70	0.78/0.85/0.81	0.86/0.90/0.89	0.86/0.90/0.87	0.62/0.62/0.62	0.89/0.87/0.88	+4.9%
	Mixboost	0.99/0.75/0.83	0.94/0.65/0.72	0.86/0.77/0.81	0.90/0.88/0.89	0.77/1.00/0.85	0.56/0.56/0.56	0.94/0.81/0.86	+7.6%
	MixAnN	**0.99**/0.93/**0.96**	**0.96**/**0.72**/**0.75**	**0.92**/0.80/**0.84**	**0.92**/**0.90**/**0.91**	0.93/0.99/**0.96**	0.84/0.58/**0.62**	**0.97**/0.83/**0.88**	-
XGBoost	w/o Augmentation	0.99/0.86/0.91	0.94/0.67/0.73	0.86/0.76/0.80	0.90/0.76/0.79	1.00/0.58/0.64	0.69/0.66/0.67	0.87/0.87/0.87	+14.2%
	Random	0.92/0.89/0.90	0.91/0.70/0.76	0.66/0.75/0.70	0.91/0.80/0.83	0.50/0.49/0.49	0.62/0.65/0.63	0.64/0.72/0.68	+23.9%
	SMOTE	0.78/0.98/0.85	0.86/0.90/0.88	0.59/0.90/0.62	0.91/0.80/0.84	0.55/1.00/0.59	0.62/0.67/0.64	0.84/0.90/0.87	+15.8%
	BorderlineSMOTE	0.79/0.98/0.86	0.87/0.92/0.89	0.58/0.90/0.60	0.86/0.81/0.83	0.50/0.94/0.47	0.63/0.66/0.64	0.87/0.88/0.87	+18.9%
	SVMSMOTE	0.82/0.99/0.88	0.86/0.87/0.86	0.63/0.92/0.68	0.84/0.81/0.82	0.56/1.00/0.60	0.64/0.66/0.65	0.86/0.89/0.87	+14.2%
	ADASYN	0.75/0.97/0.82	0.84/0.96/0.89	0.56/0.87/0.55	0.85/0.81/0.82	0.50/0.77/0.35	0.63/0.67/0.65	0.83/0.89/0.86	+23.9%
	AutoSMOTE	0.90/0.96/0.93	0.88/0.92/0.90	0.80/0.82/0.81	0.68/0.70/0.62	1.00/0.75/0.83	0.64/0.67/0.66	0.87/0.90/0.88	+10.1%
	Mixboost	0.99/0.79/0.86	0.96/0.55/0.57	0.90/0.62/0.68	0.89/0.70/0.73	0.50/0.50/0.50	0.67/0.61/0.63	0.66/0.70/0.68	+33.3%
	MixAnN	**1.00**/0.96/**0.98**	0.87 /**0.97**/**0.92**	0.81/0.85/**0.83**	**0.91**/**0.82**/**0.84**	**1.00**/**1.00**/**1.00**	**0.71**/**0.68**/**0.70**	**0.89**/**0.90**/**0.90**	-
MLP	w/o Augmentation	0.95/0.89/0.92	0.94/0.53/0.53	0.88/0.76/0.81	0.84/0.78/0.80	1.00/0.75/0.83	0.86/0.65/0.71	0.97/0.85/0.89	+11.5%
	Random	0.78/0.88/0.82	0.91/0.58/0.61	0.62/0.77/0.66	0.90/0.79/0.82	0.50/0.98/0.50	0.70/0.68/0.69	0.83/0.87/0.85	+22.5%
	SMOTE	0.85/0.99/0.91	0.59/0.73/0.59	0.62/0.92/0.67	0.88/0.81/0.83	0.51/0.99/0.51	0.73/0.69/0.71	0.88/0.92/0.90	+19.2%
	BorderlineSMOTE	0.82/0.99/0.88	0.59/0.73/0.59	0.64/0.92/0.70	0.85/0.80/0.82	0.50/0.97/0.49	0.73/0.71/0.72	0.89/0.92/0.90	+19.2%
	SVMSMOTE	0.89/0.99/0.93	0.62/0.74/0.65	0.71/0.93/0.78	0.88/0.81/0.83	0.51/0.99/0.53	0.75/0.70/0.72	0.96/0.87/0.90	+14.5%
	ADASYN	0.89/0.99/0.93	0.59/0.74/0.59	0.61/0.92/0.65	0.88/0.81/0.84	0.50/0.98/0.50	0.71/0.69/0.70	0.87/0.92/0.90	+19.2%
	AutoSMOTE	0.96/0.93/0.94	0.90/0.64/0.70	0.78/0.86/0.82	0.70/0.73/0.66	1.00/0.92/0.95	0.74/0.79/0.73	0.94/0.90/0.91	+7.4%
	Mixboost	0.65/0.60/0.62	0.96/0.52/0.52	0.90/0.72/0.78	0.89/0.73/0.76	0.90/0.83/0.86	0.67/0.58/0.61	0.93/0.95/0.88	+20.8%
	MixAnN	**0.99**/0.95/**0.96**	0.93/0.74/**0.79**	0.87/0.84/**0.85**	**0.91**/0.83/**0.85**	**1.00**/0.92/**0.95**	0.84/**0.71**/**0.76**	**0.97**/0.91/**0.93**	-

Table 5.2 : Horizontal comparison with data-augmentation methods

Methods	Label-informed			Data Augmentation	
Datasets (Prec./Rec./F1)	XGBOD	DevNet	DeepSAD	Best Baseline	MixAnN
Vowels	0.86/0.71/0.76	0.89/0.99/0.93	0.99/0.71/0.80	0.96/0.93/0.94	**1.00**/0.96/**0.98**
Annthyroid	0.88/0.52/0.52	0.63/0.61/0.62	0.90/0.56/0.59	0.88/0.92/0.90	0.87/**0.97**/**0.92**
Mammography	0.95/0.59/0.65	0.72/0.92/0.79	0.66/0.65/0.65	0.78/0.86/0.82	0.87/0.84/**0.85**
Satellite	0.91/0.80/0.84	0.71/0.71/0.71	0.81/0.73/0.75	0.90/0.88/0.89	**0.92**/**0.90**/**0.91**
SMTP	1.00/0.58/0.64	1.00/1.00/1.00	0.99/0.71/0.80	0.86/0.90/0.87	**1.00**/**1.00**/**1.00**
CIFAR-10	0.87/0.59/0.63	0.72/0.74/0.73	0.71/0.73/0.72	0.74/0.79/0.73	0.84/0.71/**0.76**
FashionMNIST	0.96/0.83/0.88	0.91/0.91/0.91	0.92/0.93/0.93	0.94/0.90/0.91	**0.97**/0.91/**0.93**
Avg. F1 ↑	+28.2%	+12.3%	+21.3%	+5.8%	-

Table 5.3 : Vertical comparison with label-informed methods. The best baseline in the data augmentation category refers to the horizontal baseline with the best F1-score across the three classifiers.

5.4.3 Vertical Analysis

In order to answer **RQ2**, we compare the MixAnN to the most advanced label-informed anomaly detection algorithms and baseline augmentation methods. Table 5.3 presents the macro-averaged precision, recall, and F1-score of each method on the 5 datasets. "Best baseline" refers to the horizontal baseline with the best F1-score on individual datasets. We make the following two observations.

First, data augmentation methods generally outperform label-informed anomaly detectors. Comparing the best data augmentation baseline to the three label-informed approaches, the best data augmentation baseline outperforms the best label-informed algorithm by 6.2%. This suggests that data augmentation may be more effective in generalizing label information when incorporated with proper strategy. Additionally, the proposed MixAnN with a properly learned strategy achieves superior performance, which further validates the suggestion above.

Second, label-informed methods achieve better precision and lower recall. By comparing Table 5.3 with Table 5.2, we can observe that the average precision of label-informed approaches is generally superior to data augmentation methods on all three classifiers, whereas the average recall is generally inferior to those baselines. One possible explanation is that label-informed approaches can only exploit the labels themselves, while data augmentation methods are capable of exploring potentially beneficial information from the limited label information. This suggests that label-informed approaches tend to over-fit existing labels, which may be sub-optimal.

5.4.4 Ablation Study

To answer **RQ3**, we conducted an ablation study on the Japanese Vowels dataset with the KNN classifier. As the reward signal is the most critical guidance toward optimal

Ablations	Precision	Recall	F1-score
Random reward	0.85	0.60	0.66
w/o Improvement Stimulation (Eq. 5.5)	0.98	0.64	0.71
w/o Model Exploration (Eq. 5.6)	0.98	0.71	0.79
Full MixAnN	**0.99**	**0.93**	**0.96**

Table 5.4 : Macro-averaged scores of MixAnN with the KNN classifier and the ablations on Japanese Vowels.

augmentation strategies, we ablate the reward function to study the contribution of individual components. Note that a random reward generates a reward signal with a random floating number from 0 to 1, which potentially leads to a random augmentation strategy. From Table 5.4, we can make the following observations.

First, the proposed MDP is solvable, and the tailored RL agent is capable of learning an optimal strategy. By comparing the random reward baseline with the MixAnN, we observe that there are significant improvements in all three metrics. Both the two ablations on Eq. 5.5 and 5.6 are significantly better than the random reward baseline, which suggests that the tailored RL agent is capable of addressing the MDP toward an optimal augmentation strategy.

Second, the two components in the proposed reward signal play significant roles in an optimal augmentation strategy. Specifically, both components are capable of increasing the exploitation of the label information and therefore lead to significant improvements in precision. On one hand, as the classifier may suffer from under-fitting during the training procedure, learning the augmentation strategy without Eq. 5.5 may lead to a significant performance drop. On the other hand, without considering the model impact via Eq. 5.6, it is less possible to identify potentially beneficial information for the underlying classifier and therefore leads to a lower recall.

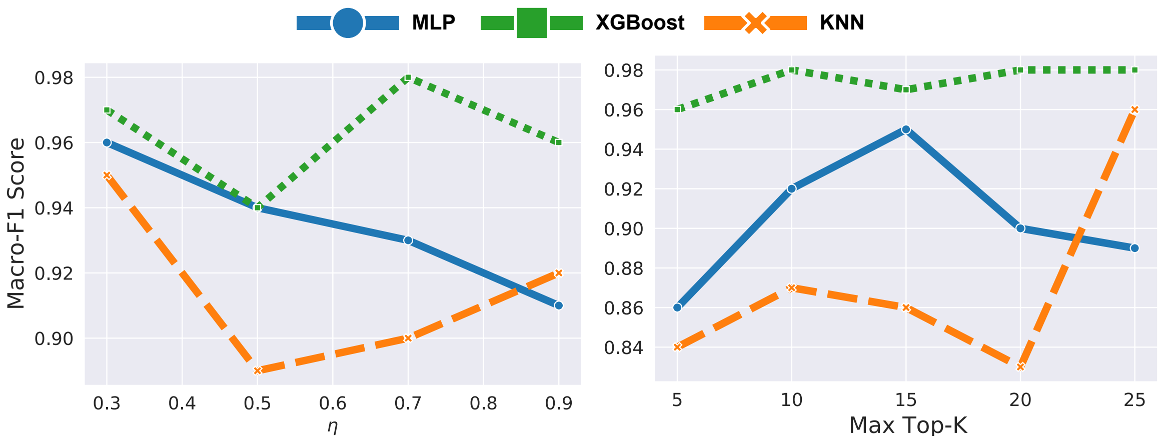


Figure 5.4 : Hyperparameter study on Vowels.

5.4.5 Hyper-parameter Study

To answer the **RQ4**, we study the two key hyperparameters (i.e., maximum neighborhood size K and the threshold η) on the vowels dataset with the three classifiers. Based on the results shown in Figure 5.4, we make the following observations.

For the size of the neighborhood K, simpler classifiers may require collecting a larger amount of neighborhood information to extend the supervisory information of the source samples. For a complex model such as a multi-layer perceptron, we may need to carefully select the size to prevent over-smoothing when collecting too much neighborhood information [152] for the classifier. As for the threshold η, we observe that a lower threshold for creating synthetic anomalies generally leads to better performance. One possible explanation is that a lower η prompts the data mixer to create more synthetic anomalies, which is therefore beneficial to the extreme imbalance setting of anomaly detection.

5.4.6 Case Study

To answer the **RQ5**, we apply MixAnN and the baselines to a publicly available 2-dimensional toy dataset ‖. We split 60% of the data for training, 20% of the data for validation, and 20% for testing and use Macro-F1 as the performance metric. Figure 5.5 illustrates the generated synthetic samples and the decision boundary obtained by training a KNN classifier on the over-sampled data. We observe that classical SMOTE-based algorithms and Mixboost tend to generate some noisy samples that interleave with the majorities between the anomalies from two different sides, which degrades the performance. AutoSMOTE is capable of generating synthetic samples that are less noisy; however, due to the true anomaly in the center, several synthetic anomalies are generated to extend the left upper decision boundary, resulting in the misclassification of true normalities. In contrast, MixAnN is capable of generating both synthetic normalities and anomalies with zero noises, so it achieves a better Macro-F1 score, which demonstrates the effectiveness of mixing up normalities and anomalies with the guidance of a learning-based agent.

‖`https://github.com/shubhomoydas/ad_examples/tree/master/ad_examples/datasets/anomaly/toy2/fullsamples`

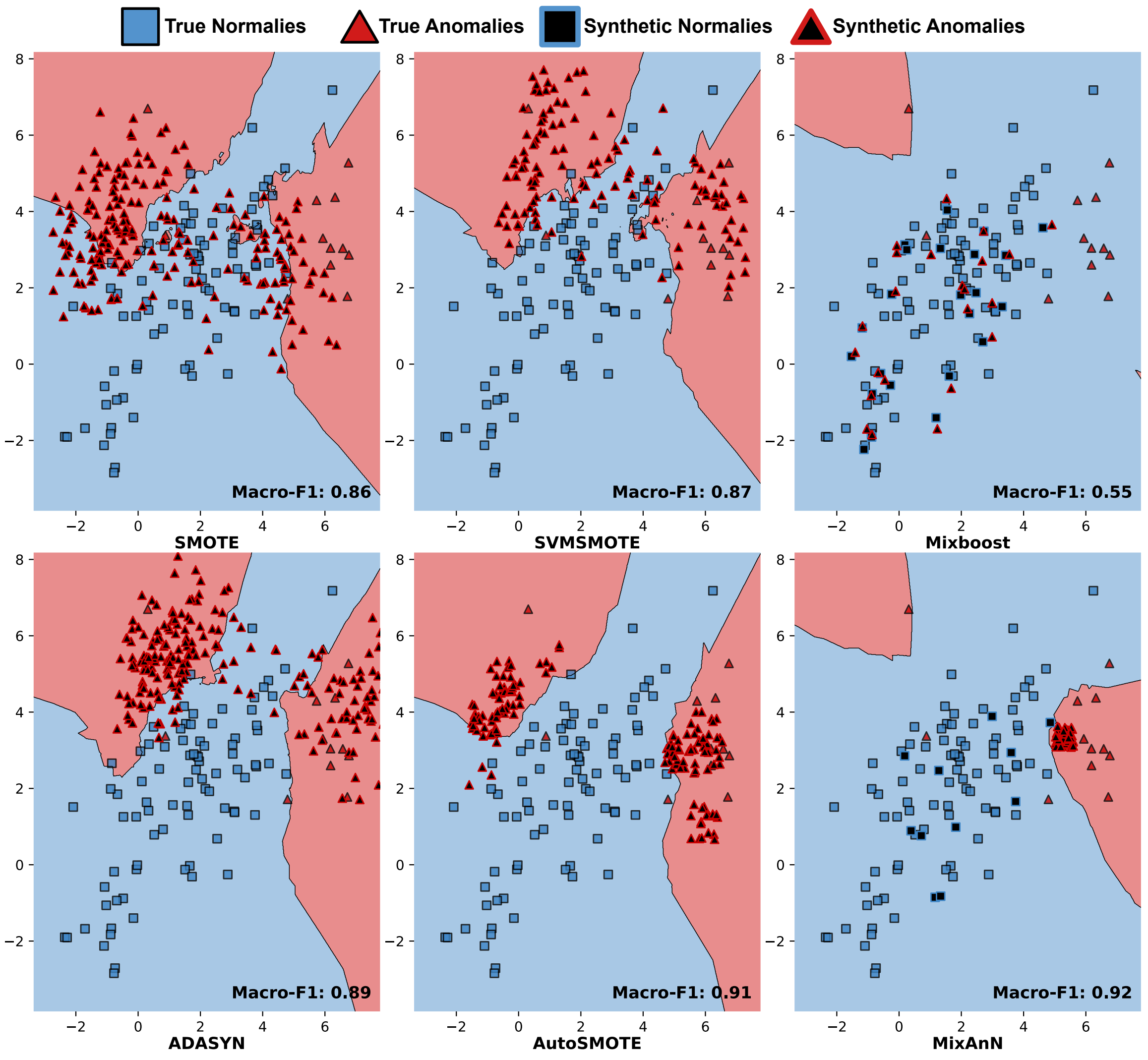


Figure 5.5 : Visualization of the generated synthetic samples and the decision boundary of DecisionTree on a toy data using AutoSMOTE and other over-sampling techniques.

5.5 Conclusion

This study aims to address the challenge of automated data preparation by introducing MixAnN, a framework that expands limited label information with considering underlying model performances. Specifically, we propose an iterative mix-up approach to generalize label information and formulate the iterative mix-up into a

Markov decision process. To guide the policy learning procedure, we create a reward function that explores the classifier's decision boundary and generates synthetic samples to maximize performance gain, and tailor a deep actor-critic framework to tackle the Markov decision process. We evaluate MixAnN in comparison to state-of-the-art label-informed anomaly detection algorithms and data augmentation approaches and demonstrate the superiority of MixAnN with extensive experimentation. The result suggests the validity of automated data preparation and its the capability of expanding AutoML toward challenging applications and scenarios.

Chapter 6

Benchmark Time Series Outlier Detection with Data-centric AutoML System

6.1 Motivation

In the previous chapters, we have addressed three critical challenges toward data-centric AutoML. As the aforementioned research fruits validate the feasibility of the data-centric AutoML, it is necessary to develop a practical instance for broader applications to further promote our idea to the general public and provide useful toolkits to better serve the humankind. Especially, detecting outliers from time series data has broad applications in various domains, such as manufacturers [153], edge devices [154] and HVAC systems [155, 156, 157].

Many algorithms have been proposed for time series outlier detection, including prediction-based models such as auto-regression [158] and recurrent neural networks [159], majority modeling approaches such as isolation forest [89] and autoencoder [160], and discords analysis methods such as subsequence clustering [161] and matrix profile [162]. Despite these efforts of advancing algorithm design, very few studies have investigated how we should benchmark the existing algorithms. Especially, given the complex nature of real-world time series outliers, the requirement of data preparation differs across algorithms, which makes it challenging to build up a benchmark platform for understanding the existing efforts and therefore leads to illusion of progress in the field [163].

In addition, while machine learning models have shown promise in time series outlier detection, the performances are still mainly depend on the entire machine learning pipeline and building an effective time series outlier detection pipeline heavily relies on human expertise. Due to the unique data characteristics of different application scenarios, one often needs expensive laboring trials to implement and identify the most suitable processing modules, detection algorithms, and hyperparameters for a specific task, which significantly impedes the real-world application and the developments of time series outlier detection.

To bridge this gap, we aim to save human efforts in building effective pipelines for time series outlier detection through data-centric AutoML and provide a transparent and extendable anomaly detection system with exhaustive functionalities, including data I/O, data processing, feature analysis, and detection algorithms, as well as an easy-to-use graphic user interface to allow human-AI interaction. With the aid of data-centric AutoML, a fair benchmark of existing efforts may be realize due to the adaptive data preparation pipeline for individual algorithms.

However, it is non trivial to develop a practical instance as a benchmark platform due to the following reasons. First, there exists various methods to prepare time series data for a detection algorithm, how to properly categorize them toward modularization and implementation may be challenging. Second, given numerous existing efforts for time series outlier detection, how to categorize the efforts toward different application scenarios and provide implementations of representative ones may be difficult. Third, given the complexity of building a ML pipeline for time series outlier detection, searching an optimal solution could be infeasible.

We present an automated **T**ime series **O**utlier **D**etection **S**ystem called **TODS**. It is designed under D3M infrastructure [164], i.e., data-driven model discovery via automated machine learning. It currently supports more than 70 primitives for data processing, time series processing, feature analysis, outlier detection, and incorporating human knowledge. It can be applied to various application scenarios, including point-wise, pattern-wise and system-wise detection. The goal of TODS is providing an end-to-end solution for real-world time series outlier detection. Furthermore, with the support of TODS, we conduct extensive experiments on the synthetic and the real-world datasets to benchmark 9 representivie algorithms, including prediction-based models, majority modeling approaches, and discords analysis methods. We surprisingly observe that some classical algorithms could outperform many recent deep learning approaches for all types of outliers. With the hope that these insights could motivate future works, we have open-sourced all the datasets, the pre-processing and synthetic scripts, and the algorithm implementation in TODS [165].

6.2 Preliminary

In this section, we introduce the problem definition of time series outlier detection. Then, we briefly discuss related works for time series outlier detection.

6.2.1 Problem Definition

Let $X = (v_0, v_1 \cdots, v_t)$ be a fully observed multivariate time series data. Let $Y \in \mathbb{N}^{t+1}$ be the label of X indicating whether individual time point v_t is an anomaly or not. Time series anomaly detection aims at recognizing if a time point $\hat{v}_t$ is anomalous or not. A scoring function $\mathcal{F} : \hat{x}_t \rightarrow \mathbb{R}$ evaluates the degree of anomalous of individual instances based on the input data $\hat{x}_t$; the higher the score, the more anomalous the time point t is.

6.2.2 Related Works

Existing time series outlier detection algorithms can be categorized into three types based on their working mechanisms: prediction deviation, majority modeling and discords analysis.

Prediction Deviation identifies the outliers by measuring the gaps between the predicted values and the original data. The assumption behind this type of algorithms is that the given data is reconstructable through regression analysis; if an individual instance is not regressable, then it is very likely to be an outlier. Autoregression (**AR**) [158] assume that each individual instance is linearly correlated to its past few instances. Gradient boosting regression (**GBRT**) [166] handles time series data in windowed-fashion and perform regression based on segmented subsequences. Derived from autoregression, recurrent neural networks with long short term memory units (**LSTM-RNN**) [159] is adopted to model the nonlinear temporal correlations between data instances.

Majority Modeling assumes that normal data instances are compact in hyperspace [167, 168]. It aims at identifying the decision boundary between outliers and normalities through modeling the regular data distribution. One-class SVM (**OCSVM**) [169, 170, 171] maximizes the margin between origin and the normalities and define the decision boundary as the hyper-plane that determines the margin. Isolation forest (**IForest**) [89, 172] builds an ensemble of binary trees to isolate the data points and defines the decision boundary as the closeness of an individual instance to the root. Autoencoder (**AE**) [160] maps the data points into low dimensional latent space, reconstructs the data points from the latent space representations, and defines the decision criteria by assuming the reconstruction error of outliers are significantly larger than normalies. Generative adversarial nerwork (**GAN**) [173] performs min-max optimization with a generator and a discriminator, where discriminator aims at modeling the normalities and generator targets on generating outliers that can be identified as normalities by discriminator. The decision criterion is defined as the discriminator loss on individual instances.

Discords Analysis measures the similarity [174] between subsequences and aims at identifying discords as outliers. Specifically, sequential data will be segmented into subsequences by a sliding window. Then, different distance computation will be performed to evaluate the discordance of each subsequence. Discords analysis is usually adopted to identify pattern-wise outliers. Subsequence clustering [161] leverages unsupervised algorithms such as OCSVM [171] and IForest [172] with segmented subsequences to detect pattern-wise outliers. Matrix profile (**MP**) [162, 175] constructs distance profiles by computing minimum distances of each subsequence to the rest of subsequences, then identifies anomalous subsequence based on the distance profile.

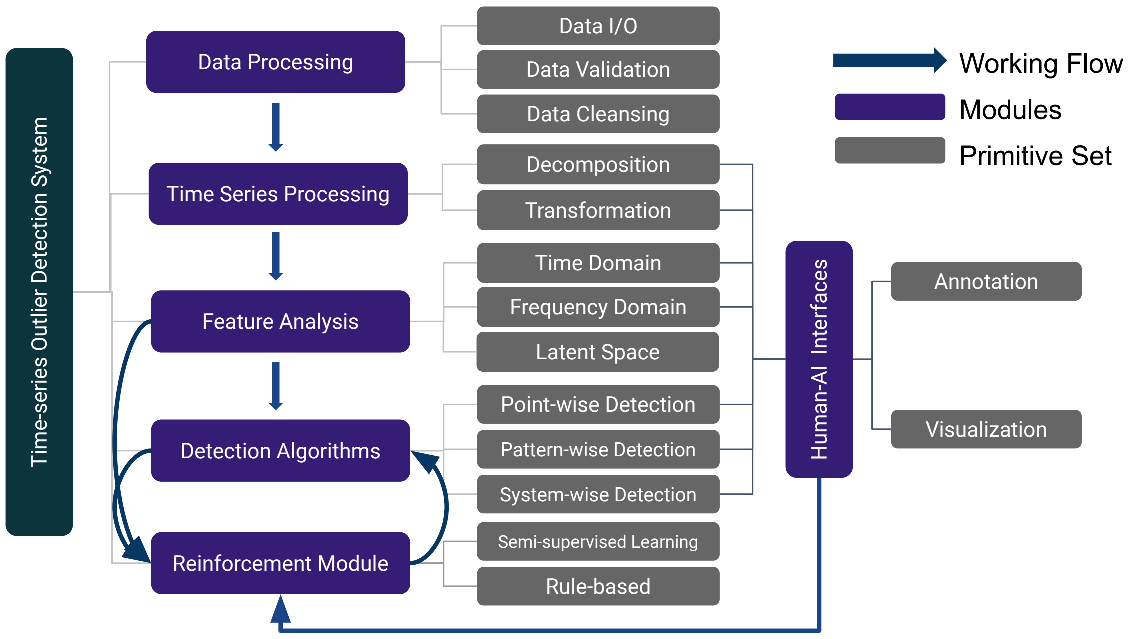


Figure 6.1 : The overview of the system

6.3 TODS: An Automated Time Series Outlier Detection System

TODS is an end-to-end **T**ime-series **O**utlier **D**etection **S**ystem. In TODS, we have mainly six modules: Data preprocessor, time-series processor, feature analyzer, a detection module, reinforcement module, and human-AI interface. Figure 6.2 illustrates the overall workflow and structural design of TODS. Specifically, each module consists of several primitive sets where each one is composed of various functions. This package aims to provide an end-to-end machine learning system for outlier detection tasks on time series data, and the target audience of this package is general software engineers with limited machine learning/data mining expertise. We define three scenarios that can include all of the outlier detection scenarios in our daily life: point-wise detection, pattern-wise detection, and system-wise detection.

- **Point-wise detection** aims at detecting the anomalous time point within the data by defining the anomalies as time points.
- **Pattern-wise detection** aims to identify odd patterns in the data by specifying the anomalies on subsequences.
- **System-wise detection** aims to find anomalous systems by defining a set of time series data as an anomaly.

In this section, we first illustrate the functionality of each module that composes TODS. Then, we introduce the provided application programming interfaces (APIs). Finally, we elaborate on the implementation of pipeline search procedure.

6.3.1 Modules Overviews

An overview of TODS is shown in Figure 6.2. The basic building block in TODS is *primitive*, which is an implementation of a function with some hyperparameters (e.g., Auto-Encoder algorithm). A *pipeline* is defined as a Directed Acyclic Graph (DAG), where each step represents a primitive. Typically, creating a ML pipeline for detecting anomalies includes four primitive steps: data processing, time series processing, feature analysis, and detection algorithms. We follow the primitive steps to develop following modules:

- **Data Processing:** The data processing module plays a crucial role in working with raw data stored in tabular format. It serves various general data processing purposes such as data loading, validation, conversion, standardization, encoding, and imputation. The primary objective of this module is to preprocess the input time series data based on its original tabular format. By doing so, it enables the data to be more manageable and usable for further analysis.

- **Time-series Processing:** The purpose of the time-series processing module is to handle time-series data by treating it as a continuous signal. This module is specifically designed to perform various time-series-specific pre-processing tasks, such as seasonality-trend decomposition, variable normalization, and distribution/domain transformation. Its objective is to prepare the time-series data for feature analysis by transforming it into a format that amplifies the temporal pattern and unifies the variable discrepancies.

- **Feature Analysis:** The objective of the feature analysis module is to extract significant features from three aspects of time series data: the temporal domain [176] (such as statistical features), the frequency domain [177] (such as a saliency map), and the latent feature space [178, 179, 180] (such as matrix factorization). Since each data point in the time series data contains limited information, the output of this module is an augmented time series with not only the information from the original data but also extended signals derived from it. In other words, the feature analysis module aims to enrich the time series data with meaningful features that can be used for detecting outliers.

- **Detection Algorithms:** The detection algorithm module performs outlier detection on the input data for three major application scenarios: point-wise, pattern-wise and system-wise outlier detection. Algorithms for point-wise detection identify outliers by comparing individual time points with the remaining time points. Examples of such algorithms include isolation-forest [89] and local outlier factors [86]. Algorithms for pattern-wise detection identify outliers by recognizing local patterns, which are described as sub-sequences that deviate from the general pattern of the majority of subsequences in the time series. Some examples including subsequence clustering [161], matrix profile [162] and generative adversarial neural networks [181]. Since there exists very limited research and development for system detection scenario, we implement several ensemble methods [182] to consider detection results from point- and pattern-wise algorithms on individual time series to address the problem.

- **Reinforcement Module**: This module aims at exploiting human knowledge to reinforce the detection performance. The key idea is to enable human experts to insert the domain knowledge for improving the detection results from ML models as unsupervised ML models may suffer from distribution shift and therefore leads to false alarms [119].

Module	Number	Example Primitives
Data Processing	7	TimestampValidation, TimeIntervalTransform, DuplicationValidation
Time-series Processing	9	SeasonalityDecomposition, MovingAverageTransform, HoltSmoothing
Feature Analysis	30	AutoCorrelation, NonNegativeMatrixFactorization, StatisticalGMean
Detection Algorithms	23	LSTM-RNN, IForest, AutoEncoder, LocalOutlierFactor
Reinforcement	1	RuleBasedFilter
Total	70	-

Table 6.1 : The number of primitives implemented in each module of TODS with example algorithms in each module.

The primitives developed in each module are shown in Table 6.1. Various pipelines can be constructed based on these primitives in different application scenarios. More comprehensive list of algorithms can be found on the GitHub repository released under Apache 2.0 license at `https://github.com/datamllab/tods`.

6.4 Automated Pipeline Searching

Beyond manual pipeline construction, TODS provides data-driven searchers to automatically find good pipelines for a given task. To perform pipeline search, users are required to provide a search space to specify candidate primitives of each module and the corresponding range of hyper-parameters. The, given a time budget and evaluation criteria, two kind of searchers can be employed for the pipeline searching: random searcher and hierarchical searcher.

- **Random Searcher:** The random searcher simplifies the search space by flattening the search space for each module. It then selects a primitive and a corre-

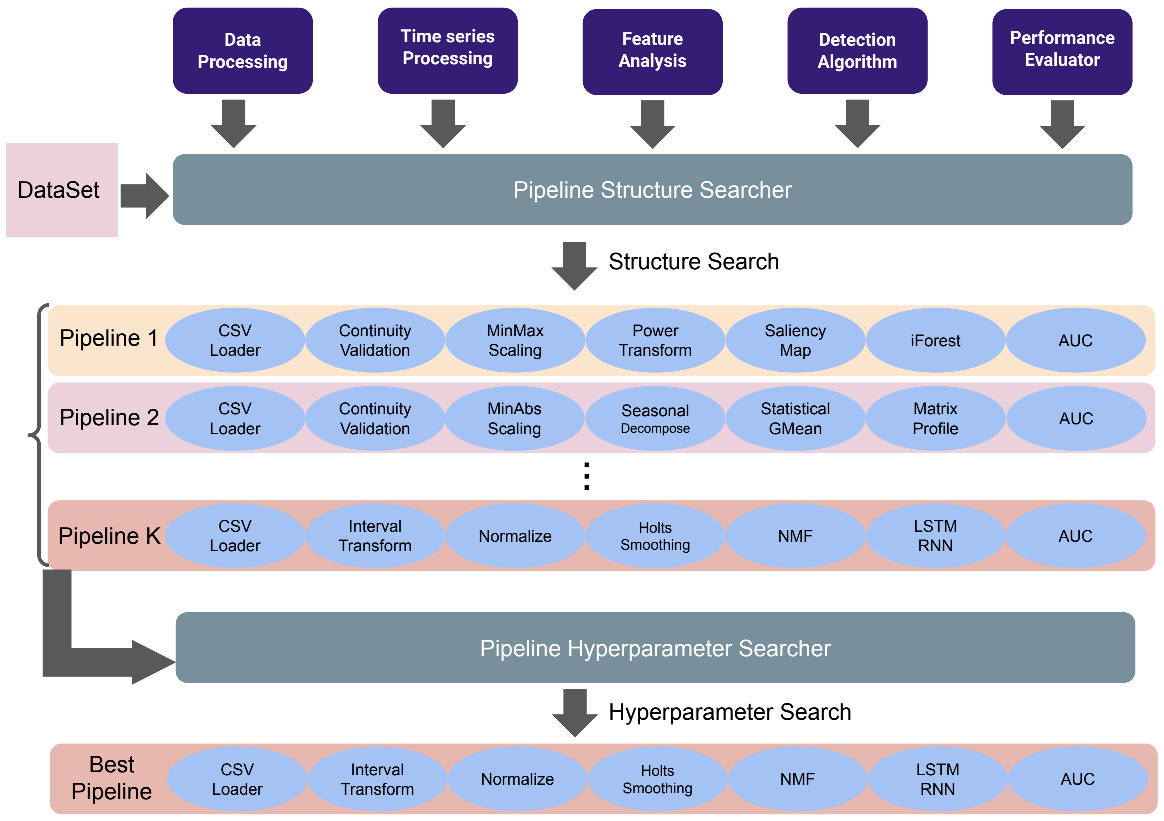


Figure 6.2 : The illustration of hierachical search

sponding set of hyper-parameters from each module through uniform sampling to generate a pool of candidate pipelines. Finally, the best pipeline discovered within the allotted time frame is returned as the final outcome.

- **Hierarchical Searcher:** A search procedure can be conducted using a hierarchical searcher that divides the search space into primitive and hyper-parameter levels. The process involves an evolutionary search algorithm [183] and has two stages. In the first stage, the searcher uses default hyper-parameters to perform a primitive-level search and identify a pool of candidate pipelines. The top-k pipelines are then selected and paired with corresponding hyper-parameter ranges for further optimization. Depending on the time budget allocated, the search can prioritize exploration to identify more pipeline structures or exploitation to optimize existing pipeline structures.

To facilitate pipeline search, TODS also provides a data preparation pipeline for data splitting, such as k-fold splitting, and a scoring pipeline to evaluate the performance based on a given metric, such as F1 score or customized score. The data-driven searcher takes a dataset as input to automatically discover a detection pipeline that can achieve the best performance on a given dataset. Then the discovered pipeline can be outputted for real-world deployment.

```
import pandas as pd
from tods import generate_dataset, load_pipeline,
from tods import evaluate_pipeline
from tods.utils import build_pipeline

# Read data and generate dataset and problem
data_path = '../../datasets/example.csv'
df = pd.read_csv(data_path)
dataset = generate_dataset(df, target_index=target_index)
target_index = 6 # what column is the target

metric = "AUC"

# define pipeline structure
pipeline = {
    'data_processing':[
            ('time_interval_transform', {'time_interval': 5T})
        ],
    'timeseries_processing':[
            ('standard_scaler', {'with_mean':True})
        ],
    'feature_analysis':[
            ('statistical_maximum', {'window_size':3}),
            ('statistical_minimum', {'window_size':15})
        ],
    'detection_algorithm':[
            ('pyod_ae', {'hidden_neurons':[32,16,8,16,32]})
        ],
}

# Executing Pipeline
pipeline = build_pipeline(config)
pipeline_result = evaluate_pipeline(dataset, pipeline, metric)
```

Code Listing 6.1: Pipeline interface for pipeline creation and execution

6.5 Application Development Interface

To enable users from diverse background to engage, we develop three application development interfaces (APIs):

```
import pandas as pd
from tods import generate_dataset, generate_problem
from tods.searcher import RaySearcher

# Read data and generate dataset and problem
data_path = '../../datasets/example.csv'
df = pd.read_csv(data_path)
dataset = generate_dataset(df, target_index=target_index)
target_index = 6 # what column is the target

time_limit = 300 # How many seconds for search process

# initialize the searcher
searcher = RaySearcher(dataframe=df, target_index=target_index,
              dataset=dataset, metric='F1_MACRO', budget=
                                                  time_limit)

# define search space
search_space = {
    "timeseries_processing": {
        "time_series_seasonality_trend_decomposition": {
            "use_semantic_types": [1, 0]
        },
        "moving_average_transform":{
            "window_size":[5, 30],
            "norm":["l1", "l2", "max"],
        }},
    "feature_analysis": {
        "statistical_h_mean": {
            "window_size": [10, 20]
        },
        "statistical_maximum": {
            "window_size": [10, 20]
        }},
    "detection_algorithm": {
        "pyod_ae": {
            "dropout_rate": [0.1, 0.2]
        },
        "pyod_loda": {
            "n_bins": [10, 20]
        }}
}

# Start searching
search_result = searcher.search(search_space=search_space)
```

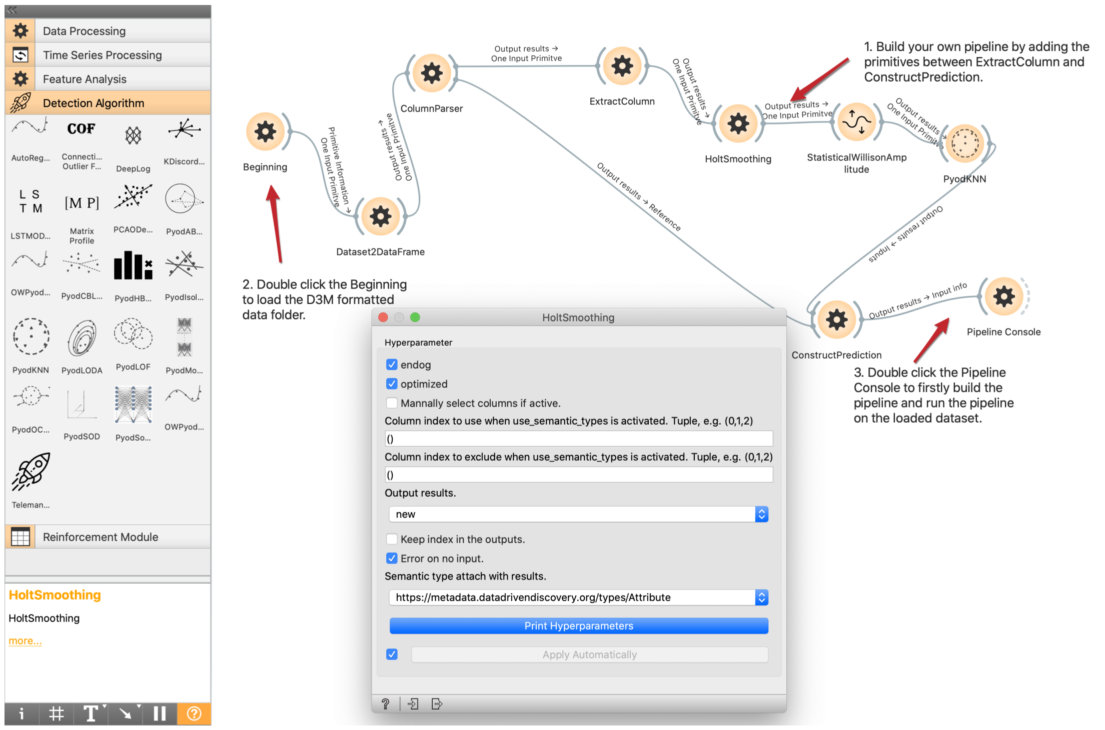
Code Listing 6.2: Pipeline interface for AutoML search

- **Pipeline programming interface** enables users to create pipelines using only a few lines of code. These pipelines can be described in JSON format and shared with colleagues, and then executed using the D3M engine. The interface has been created specifically for the purpose of scaling up the execution of machine learning pipelines. With this interface, users can simultaneously configure multiple modules, thereby enabling them to build machine learning solutions. Additionally, an AutoML is also supported with this interface. Code Listing 6.1 shows the example code snippets of creating and executing a pipeline; and Code Listing 6.1 shows the example code snippets of customizing search space for AutoML search.

```python
import numpy as np
from tods.sk_interface.detection_algorithm import DeepLogSKI
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import accuracy_score

#prepare the data
data = np.loadtxt("./benchmark1_10000_19280_19360.txt")

# first 10000 data points are training data
X_train = np.expand_dims(data[:10000], axis=1)
# second 10000 data points are testing data
X_test = np.expand_dims(data[10000:20000], axis=1)

# creating labels
y_true = np.zeros(10000).astype(np.int32)
y_true[9280:9360] = 1
y_true = np.expand_dims(y_test, axis=1)

transformer = DeepLogSKI()
transformer.fit(X_train)

y_pred = transformer.predict(X_test)
y_score = transformer.predict_score(X_test)

precision, recall, thresholds = precision_recall_curve(y_true,
                                     y_pred)
f1_scores = 2*recall*precision/(recall+precision)

print('Accuracy Score: ', accuracy_score(y_true, y_pred))
print('Best threshold: ', thresholds[np.argmax(f1_scores)])
print('Best F1-Score: ', np.max(f1_scores))
```

Code Listing 6.3: Primitive interface for incorporating with third party package

- **Primitive programming interface** follows the API design of scikit-learn with *fit* and *produce* functions for each primitive. This enables users to make use of the individual primitives within TODS and to easily combine them with other



1. Build your own pipeline by adding the primitives between ExtractColumn and ConstructPrediction.

2. Double click the Beginning to load the D3M formatted data folder.

3. Double click the Pipeline Console to firstly build the pipeline and run the pipeline on the loaded dataset.

HoltSmoothing

Hyperparameter

[✓] endog
[✓] optimized
[ ] Mannally select columns if active.

Column index to use when use_semantic_types is activated. Tuple, e.g. (0,1,2)
()

Column index to exclude when use_semantic_types is activated. Tuple, e.g. (0,1,2)
()

Output results.
new

[ ] Keep index in the outputs.
[✓] Error on no input.

Semantic type attach with results.
https://metadata.datadrivendiscovery.org/types/Attribute

Print Hyperparameters

[✓] Apply Automatically

Figure 6.3 : An illustration of the GUI. Users can easily build and evaluate a pipeline with drag-and-drop.

third-party packages for detailed experimentation and other data types. An illustration of the primitive programming interface of TODS can be seen in Code Listing 6.3.

- **Graphical User Interface (GUI)** is to provide a development environment for users who may not be familiar with coding. This interface is based on Orange [184] and is designed to enable users to build pipelines using a visual approach. The GUI provides users with access to various primitives, which are organized into tabs and represented by icons with text descriptions. Users can select the primitives they need, drag them onto the canvas, and connect them using lines. By double-clicking on the icon, users can easily configure the hyperparameters for each primitive, as illustrated in the example shown in Figure 6.3. TODS, the system underlying the GUI, automatically generates the pipeline language based on the structure created in the interface. The resulting pipeline is then sent to the backend for time series outlier detection.

6.6 Benchmark Time Series Outlier Detection with TODS

In this section, we introduce 35 synthetic datasets based on our previously proposed criterion [24] and identify four real-world multivariate sequential data which cover both point- and pattern-wise outliers. We benchmark 9 representative algorithms implemented in TODS [165] on these datasets. In what follows, we first describe the details of the synthetic datasets and the real-world datasets, and then elaborate on the included algorithms. Finally, we present the benchmark results and analysis.

Attributes / Type	Length	Dim.	Noise	Outlier Radius	Outlier Ratio	Specifications
Global	200	1	0.05	-	$\{0.05,0.1,0.15,0.2\}$	λ=6.0
Contextual	200	1	0.05	-	$\{0.05,0.1,0.15,0.2\}$	k=5, η=2.0
Shapelet	200	1	0.05	5	$\{0.05,0.1,0.15,0.2\}$	ρ=sgn(sin(.))
Seasonal	200	1	0.05	5	$\{0.05,0.1,0.15,0.2\}$	ω=4, $\hat{\omega}$=0.04
Trend	200	1	0.05	5	$\{0.05,0.1,0.15,0.2\}$	$m(\tau)$=5, $m(\hat{\tau})$=0
Multi-1	200	5	0.05	5	0.05	1 type of outlier
Multi-2	200	5	0.05	5	0.1	2 types of outlier
Multi-3	200	5	0.05	5	0.15	3 types of outlier
Multi-4	200	5	0.05	5	0.20	4 types of outlier
Multi-5	200	5	0.05	5	0.25	5 types of outlier

Table 6.2 : Details of synthetic datasets.

6.6.1 Descriptions of the Datasets

We conduct benchmark experiments in unsupervised setting. Each of the algorithm is trained and tested on the same dataset. The outliers are identified based on the outlierness score generated by individual algorithms with a given contamination ratio. The benchmark experiments are conducted on both synthetic and real-world datasets as follows:

Synthetic Datasets

The goal of synthetic datasets is to examine the ability of algorithms to identify 5 type of proposed outliers. We generate 35 synthetic datasets with 20 univariate and 15 multivariate datasets to examine the existing algorithms in detail. Specifically, we adopt sinusoidal wave as the base shapelet to generate 20 univariate sequential data with different ratio of outliers, where each dataset only include one kind of outlier. Then, we also generate 15 multivariate sequential data which combine different kinds of outliers into single dataset. The parameters for generating univariate datasets are provided in Table 6.2. We generate the dataset based on our developed synthetic data generator. Specifically, we generate 20 univariate datasets, where each dataset includes only one type of outlier with different outlier ratio. The outlier ratio is defined as $\frac{labeled\_timepoint}{total}$. For multivariate, we generate 15 five-dimensional data with 1 to 5 types of outlier in each dimension. We opensource our synthetic dataset in the "benchmark" branch of TODS [*], where the README.md file gives more details on the datasets.

[*]`https://github.com/datamllab/tods/tree/benchmark`

Attributes \ Dataset	#(Timestamps)	#(Dimensions)	Outlier Ratio
Credit Card	284,807	29	0.173%
Web Attack	170,231	79	1.281%
Water Quality	138,521	10	1.246%
Space Weather	120,000	39	23.8%

Table 6.3 : Details of real-world datasets.

Real-world Datasets

We identify four public available real-world datasets from four different application scenario with two event-driven application and two time-based application: credit card fraud detection, IoT for drinking water monitoring, server attack monitoring and extreme space weather detection. Compared to existing multivariate benchmark datasets such as SMD, SMAP, MSL, which only have point outliers, our datasets involve wide range types of outliers.

- **Credit Card** [†] is collected by openML which contains transactions made by credit cards in September 2013 by European cardholders. The fraudular transactions are labeled as outliers. In our repository, we name the processed data as "creditcard".

- **CICIDS** [‡] is collected by Canadian Institute for Cybersecurity in 2017. We adopt the "Thursday-WorkingHours-Morning-WebAttacks" file in 2017 datasets

[†] `https://www.openml.org/d/1597`

[‡] `https://www.unb.ca/cic/datasets/ids-2017.html`

which contains 3 kinds of intrusion attack: XSS, SQL injection and brute force attack. We drop all of the row and columns with NaN and label all kinds of attack as outliers. In our repository, we name the processed data as "web_attack".

- **GECCO** [§] is collected by SPOTSeven Lab [¶] for hosting a data challenge [‖] in 2018. We binarized all of the binary columns with 1 and 0. In our repository, we name the processed data as "water_quality".

- **SWAN-SF** [**] is collected by Harvard Dataverse. We adopt the labeled version from the demo of an open sourced multivariate time series preprocessing toolkit [††]which can be directly downloaded from bitbucket [‡‡]. We merge all of labeled csv file as a multivariate time series data. Then, we drop all columns and rows with NaN values, binarize all binary columns and label all of the flares as outliers. In our repository, we name the processed data as "swan_sf".

6.6.2 Algorithms, Hyper-parameter Space and Benchmark Method

We include 8 representative algorithms including prediction deviation, majority modeling and discord analysis algorithms. We adopt the implementations from open-sourced outlier detection library: TODS. Specifically, the benchmark algorithms including GAN, Autoencoder, IForest and OCSVM, subsequence segmentation (with IForest and OCSVM), LSTM-RNN, autoregression and matrix profile. For synthetic

[§]`https://bit.ly/3fOeRvI`

[¶]`https://www.spotseven.de`

[‖]`https://www.spotseven.de/gecco/gecco-challenge/gecco-challenge-2018/`

[**]`https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/EBCFKM`

[††]`https://github.com/AzimAhmadzadeh/mvtsdata_toolkit/blob/master/demo.ipynb`

[‡‡]`https://bitbucket.org/gsudmlab/mvtsdata_toolkit/downloads/petdataset_01.zip`

datasets, we align the contamination ratio for all of the algorithms with the outlier ratio. For real-world datasets, we search the contamination ratios from {0.01, 0.05, 0.1, 0.15, 0.2, 0.25} for all of the algorithms and report the best one. For each algorithm, we firstly perform pipeline structure search to identify best ML pipeline structure, then we adopt the following hyper-parameter space to search for the best results:

- **AR:** We set step=1 and search the window size within {3, 5, 10} for all of the datasets and report the best result.
- **GBRT:** We set step=1 and search the window size within {3, 5, 10} for all of the datasets and report the best result.
- **LSTM-RNN:** For LSTM-RNN, we search the number of hidden layer within {2, 5, 10, 20, 32} and number of hidden units within {32, 64, 128, 256}. We train the model for 20 epochs with batch size as 32 and search dropout ratio in {0.1, 0.2, 0.3}. Finally, we report the best detection result for each dataset.
- **IForest** For IForest, we search $n_estimators$ in {100, 300, 500, 700, 900} and use default for the rest.
- **OCSVM** We only set the $max\_iter = 1000$ to prevent from infinity iteration. We search kernel in {linear, poly, rbf, Sigmoid}, ν in {0.1, 0.3, 0.5, 0.7, 0.9} and use the default for the rest.
- **Autoencoder:** We search the autoencoder with the candidate neural architectures given by {(32,16,32), (16,8,16), (32,16,8,16,32), (64,32,16,32,64), (128,64,32, 64,128), (256,64,32,64,256)}, where the first and last element in each tuple indicate the number of units in the input and output layers, respectively, and other elements indicate that of hidden layers. The autoencoder is trained for 20

epochs with batch size as 32 and the activation function is searched in {Sigmoid, ReLU, Linear} with dropout rate in {0.1, 0.2, 0.3}.

- **GAN:** We adopt the MO-GAAL from TODS which is initially implemented in PyOD, and we search the hyperparameter k in {1,2,3,4,. . . , 20} and the training epochs within {20, 30, 50}. Since it fails to identify any outliers in real-world datasets, we only report this algorithm on synthetic datasets.
- **Matrix Profile:** We search the window size within {3, 5, 10}, and report the best result with window size as 10. Besides, we only report the results for synthetic datasets because MP cannot complete the training procedure within 1000 CPU hours for all of the real-world datasets.
- **Subsequence Clustering:** We search the window size within {3,5,10} to and the aforementioned hyper-parameter space of OCSVM and IForest to establish subsequence clustering baselines: ▲OCSVM and ▲IForest.

6.6.3 Results and Analysis

We report the F1 score on the datasets with different outlier ratios or the numbers of involved outlier types in Figure 6.5 and tabulate the results of real-world datasets in Table 6.4.

Synthetic Datasets.

Figure 6.5 summarizes the benchmark result on 35 synthetic datasets with F1 score. Specifically Figure 6.5a to 6.5e concludes the F1 score with respect to outlier ratio on 20 univariate sequential data and figure 6.5f shows the average F1 score with respect to number of involved outlier type on 15 multivariate synthetic datasets. We make the following observations.

First, classical algorithms generally outperform deep learning based methods on all of the synthetic datasets. Specifically, AR outperforms all other algorithms in detecting contextual and shapelet outliers; OCSVM and IForest outperform the rests in global outliers and multiple outliers on multivariate setting; and discord analysis algorithms perform the best in seasonal and trend outlier tasks.

Second, detecting contextual outliers is challenging for most of the algorithms. Among all of the algorithms, only AR is able to achieve good performance. A possible reason is AR adopts contextual points to perform self regression and modeling the normalies in the context window, which benefits detecting contextual outlier.

Third, prediction-based algorithms which are designed to detect point-wise outliers are also applicable to some of the pattern-wise outliers. For example, AR outperforms all of other algorithms when detecting shapelet outlier. The reason behind this is that we adopt square sine as the anomalous shapelet to increase the difficulty. However, since the seasonality and trend of shapelet outliers remain identical to normalies, the right angle part of the synthetic outlier will be deemed as contextual outliers by AR and therefore yield an superior performance.

[h]

Fourth, local z-normalization adopted by MP may damage the performance for identifying trend outliers with different directions on zero-centered sequence when the window size is not properly set. As shown in Figure 6.4, with the window size that smaller than the range of outlier, the value range of the trend outlier will be similar to normal subsequences after applying the local z-normalization on the two trend outliers. Moreover, the original trend shift of the two outliers are transferred to their

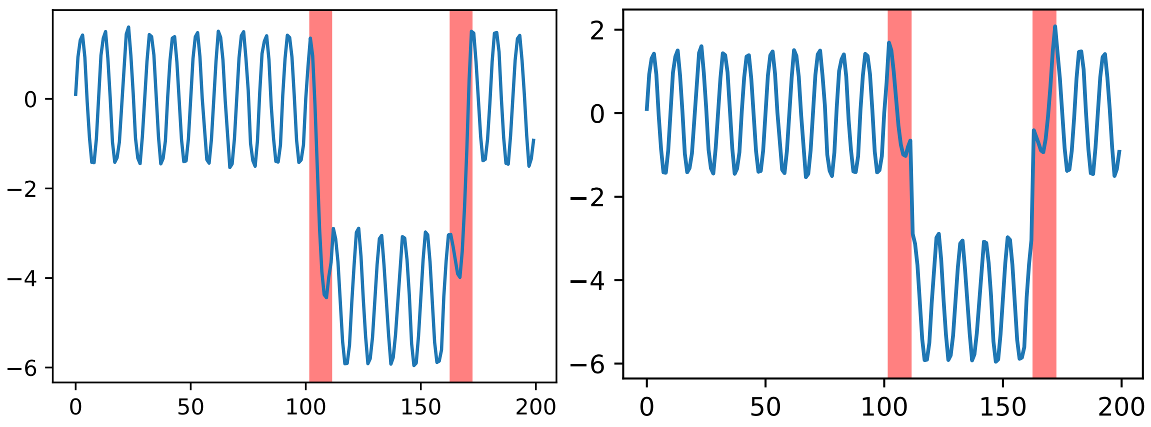
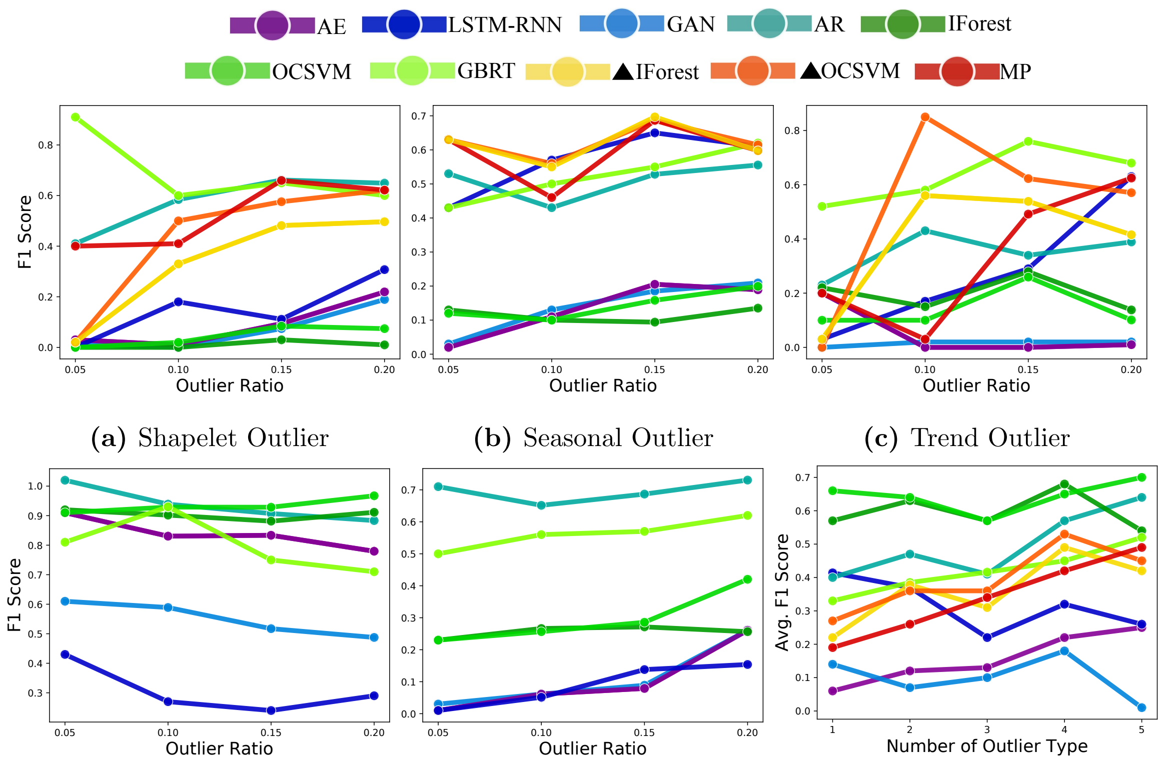


Figure 6.4 : Before (upper) and after (lower) applying local z-normalization.

neighboring points, which make it hard for MP to identify the true trend outlier.

Lastly, deep learning methods such as RNN and GAN can only handle limited type of outliers. In the Figure 6.5f, the average F1 score of GAN and RNN tend to decrease when more types of outliers are involved. This suggests that the two algorithms might have limited performance on real-world datasets with numerous kinds of outliers or mixed type of outliers.



(a) Shapelet Outlier **(b)** Seasonal Outlier **(c)** Trend Outlier

(d) Global Outliers **(e)** Contextual Outlier **(f)** Multiple Outliers

Figure 6.5 : Summary of benchmark results on univariate (*a*-*e*) and multivariate (*f*) synthetic datasets. ▲IForest and ▲OCSVM are the subsequence clustering with the two algorithms. Figure *a*-*e* report the F1 score with respect to different ratio of outliers within the dataset and figure *f* report the F1 score with different number of outlier types within the data. We report only prediction deviation and majority modeling-based algorithms for the point outliers.

Dataset (Best)	Credit Card			CICIDS			GECCO			SWAN-SF		
Metrics	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
AR	0.113	0.652	0.192	0.016	0.310	0.030	0.392	0.314	0.349	0.421	0.354	0.385
GBRT	**0.113**	**0.657**	**0.193**	0.018	0.351	0.034	0.175	0.140	0.156	0.447	0.375	0.408
LSTM-RNN	0.004	0.110	0.007	**0.024**	**0.383**	**0.046**	0.343	0.275	0.305	0.527	0.221	0.312
IForest	0.098	0.569	0.168	0.010	0.040	0.016	**0.439**	0.353	**0.391**	**0.569**	**0.598**	**0.583**
OCSVM	0.107	0.620	0.183	0.004	0.046	0.007	0.185	**0.743**	0.296	0.474	0.498	0.485
AutoEncoder	0.103	0.598	0.176	0.011	0.042	0.017	0.424	0.340	0.377	0.497	0.522	0.509
▲IForest	0.039	0.226	0.066	0.011	0.168	0.020	0.392	0.315	0.390	0.406	0.425	0.416
▲OCSVM	0.002	0.305	0.004	0.000	0.000	0.000	0.021	0.341	0.040	0.193	0.001	0.001
MatrixProfile	0.006	0.514	0.012	0.007	0.080	0.013	0.046	0.185	0.074	0.167	0.175	0.171

Table 6.4 : Benchmark results on four real-world multivariate sequential data. ▲ represents the subsequence clustering based the algorithm.

Real-world Datasets.

Table 6.4 tabulates the best result for each algorithm on the real-world datasets. In the real-world experiments, we search the contamination ratio for all of the algorithms in $\{0.01, 0.05, 0.1, 0.15, 0.2, 0.25\}$ and select the best precision, recall and F1-score to report for each dataset. Since GAN cannot identify any outliers from all of the four real-world datasets, we exclude the algorithm in the benchmark result. Based on the Table 6.4 we can make two observations as follows.

First, classical algorithms generally outperform deep learning methods. Except for the web attack dataset, all of other datasets are dominated by AR, IForest and OCSVM. Although this is reflected in the synthetic benchmark, it is surprising that GAN cannot identify any of the outliers within the four datasets. A possible explanation is that the outliers in real-world datasets are very complex with very different patterns, which is aligned with the result in multivariate synthetic benchmark in Figure 6.5f that GAN may not be able to detect outliers from dataset with numerous kinds of outliers.

Second, subsequence clustering algorithms are not robust to real-world data when combined with OCSVM. As shown in the table ▲OCSVM has the worst performance among all of the datasets with a huge gap to other algorithms. This is because the OCSVM assumes that all of the normal subsequences can be mapped into the same cluster in hyperspace, which may be not true in real-world datasets. Specifically, we observe that OCSVM with subsequence segmentation costs more than ten times of training time compared with vanilla OCSVM. This suggests that it is very challenging to find a hyperspace to cluster all normal subsequences into one class and therefore the training iteration will never stop if no maximum is set.

6.7 Conclusion

In this work, we develop an instance of data-centric automated machine learning system, TODS, for detecting outliers in time series.

The clear context definitions in the point- and pattern-wise behaviors make the proposed taxonomy ideal for synthesizing outliers. Based on the taxonomy, we present a general synthetic criterion with 35 corresponding synthetic datasets and identify 4 multivariate real-world datasets from different domains. We then benchmark 9 algorithms using these datasets and empirically show that classical algorithms are generally and surprisingly be superior in both synthetic and real-world datasets. We hope this insight gleaned from our benchmark experiments could motivate future algorithm design. To facilitate the reproducibility and fast experimental pipeline in time series outlier detection, we have made all the datasets, scripts, and algorithm implementations publicly available, and we will actively maintain this project. In the future, we will enrich our benchmark with more datasets and polish the definition of outliers with more delicate synthetic criteria. We will also benchmark more state-of-the-art algorithms and leverage this platform to design more effective algorithms to tackle different types of outliers.

Chapter 7

Conclusions and Future Research Directions

AutoML has been shown to be an tools that makes ML accessible to the general public and improve the productivity of machine learning engineers. However, as recent studies show the bottleneck of the AutoML and the recent data challenges suggest the critical role of the data preparation, we propose a data-centric AutoML framework to compliment the shortcoming of existing model-centric AutoML framework. Our research focuses on bringing sample-wise model customization, data preparation and knowledge acquisition into the AutoML. The contribution of this research is two folded. First, we show the feasibility of the data-centric AutoML by introduced fundamental algorithms and strategies for tackle the challenges of automated model customization, knowledge acquisition and data preparation. Second, we show the capability of data-centric AutoML by adopting the proposed framework into real-world problems such as semi-supervised learning and outlier detection which are equipped with challenging settings, sparse label information. Third, we instantiate an instance of data-centric AutoML system to tackle challenging time series outlier detection problem and further benchmark the existing algorithm with data-centric AutoML and identify the illusion of progress in this research area. Putting all these together, our research could help build more powerful AutoML systems, facilitating their deployment in our daily life to better serve human beings.

Despite the efforts we have put into this thesis, there is still room for improvement.With the demonstrated success of data-centric AutoML in real-world situations, there is potential for further exploration of its applicability to various scenarios. For future work, we are interested in the follwing directions:

- **Human-centric machine learning.** AutoML aims to make machine learning accessible to individuals who lack comprehensive knowledge of the field, thus emphasizing the importance of enabling human interaction with ML solutions. To achieve human-centric machine learning, two significant challenges must be addressed: automated exploration and exploitation of domain expertise, and the development of explainable AutoML and ML solutions. Addressing the issue of automated domain expertise exploration and exploitation can enhance the resulting solution by better mining the domain knowledge from humans. While active learning methods can explore domain expertise by generating good queries for users, there is no current effort to evaluate the queried knowledge and develop specific ways to exploit it to maximize model performance. For explainable AutoML, although there exists various explanation methods for individual models [185, 186], there exists very limited studies for explaining AutoML engine. As AutoML can be complex, enabling users to trust and understand the models is particularly important for AutoML to participate in high-stakes applications such as financial and medical analysis. Additionally, understanding the decision-making mechanism of models allows humans to provide critical feedback, which can further improve ML models and AutoML procedures.

- **Efficient data-centric policies.** Data-centric AutoML may suffer from large-scale datasets due to the large search space and complex data-centric operations. It is particularly important to address the problem for broader applications. One intuitive way is to directly work on system-wise improvement (e.g., better memory usage). Apart from that, we may also alleviate the problem from algorithm perspective by developing transferable data-centric policy. Given the fact that there exists limited number of data-centric operations and datasets from similar domains may equip with similar attribute, developing pretrain strategy for generating a transferable data-centric policy toward different applications and datasets could be a promising direction.

- **Unified benchmark criteria.** In many challenging real-world applications, it is often very difficult to evaluate the validity of existing solutions due to the complexity of the problem and the resulting datasets. Therefore, it is particularly critical to conduct benchmarks to study existing algorithms. However, given the different assumptions of individual algorithms, the data preparation process may vary a lot, and a unified preprocessed input for all algorithms may lead to biased benchmark results. To address this issue, data-centric AutoML could be a possible approach to identify the most suitable input for individual algorithms and generate the results within a given computational budget. Formally define the problem and develop an benchmark criteria upon data-centric AutoML may be a promising directions toward effective comparison between ML algorithms.

Bibliography

[1] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI magazine*, 2008.

[2] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, "Deep learning for computer vision: A brief review," *Computational intelligence and neuroscience*, 2018.

[3] V. Borisov, T. Leemann, K. Seßler, J. Haug, M. Pawelczyk, and G. Kasneci, "Deep neural networks and tabular data: A survey," *arXiv preprint arXiv:2110.01889*, 2021.

[4] F. Xia, K. Sun, S. Yu, A. Aziz, L. Wan, S. Pan, and H. Liu, "Graph learning: A survey," *IEEE Transactions on Artificial Intelligence*, 2021.

[5] Z. Jiang, X. Han, C. Fan, Z. Liu, N. Zou, A. Mostafavi, and X. Hu, "Fmp: Toward fair graph message passing against topology bias," *arXiv preprint arXiv:2202.04187*, 2022.

[6] J. F. Torres, D. Hadjout, A. Sebaa, F. Martínez-Álvarez, and A. Troncoso, "Deep learning for time series forecasting: a survey," *Big Data*, 2021.

[7] N. K. Logothetis and D. L. Sheinberg, "Visual object recognition," *Annual review of neuroscience*, 1996.

[8] M. Riesenhuber and T. Poggio, "Models of object recognition," *Nature neuroscience*, 2000.

[9] G. Wang, Z. P. Bhat, Z. Jiang, Y.-W. Chen, D. Zha, A. C. Reyes, A. Niktash, G. Ulkar, E. Okman, X. Cai, *et al.*, "Bed: A real-time object detection system for edge devices," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, 2022.

[10] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez, "Recommender systems survey," *Knowledge-based systems*, 2013.

[11] R. Burke, "Hybrid recommender systems: Survey and experiments," *User modeling and user-adapted interaction*, 2002.

[12] F. Z. Xing, E. Cambria, and R. E. Welsch, "Natural language based financial forecasting: a survey," *Artificial Intelligence Review*, 2018.

[13] R. C. Cavalcante, R. C. Brasileiro, V. L. Souza, J. P. Nobrega, and A. L. Oliveira, "Computational intelligence and financial markets: A survey and future directions," *Expert Systems with Applications*, 2016.

[14] A. A. Cook, G. Mısırlı, and Z. Fan, "Anomaly detection for iot time-series data: A survey," *IEEE Internet of Things Journal*, 2019.

[15] J. E. Zhang, D. Wu, and B. Boulet, "Time series anomaly detection for smart grids: A survey," in *2021 IEEE Electrical Power and Energy Conference (EPEC)*, 2021.

[16] Z. Jiang, K. Zhou, M. Zhang, R. Chen, X. Hu, and S.-H. Choi, "Adaptive risk-aware bidding with budget constraint in display advertising," *arXiv preprint*

arXiv:2212.12533, 2022.

[17] S. K. Karmaker, M. M. Hassan, M. J. Smith, L. Xu, C. Zhai, and K. Veeramachaneni, "Automl to date and beyond: Challenges and opportunities," *ACM Computing Surveys (CSUR)*, 2021.

[18] Q. Yao, M. Wang, Y. Chen, W. Dai, Y.-F. Li, W.-W. Tu, Q. Yang, and Y. Yu, "Taking human out of learning applications: A survey on automated machine learning," *arXiv preprint arXiv:1810.13306*, 2018.

[19] R. Elshawi, M. Maher, and S. Sakr, "Automated machine learning: State-of-the-art and open challenges," *arXiv preprint arXiv:1906.02287*, 2019.

[20] X. Shi, Y. D. Wong, C. Chai, and M. Z.-F. Li, "An automated machine learning (automl) method of risk prediction for decision-making of autonomous vehicles," *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[21] A. Singh, N. Thakur, and A. Sharma, "A review of supervised machine learning algorithms," in *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, 2016.

[22] J. Brownlee, *Data preparation for machine learning: data cleaning, feature selection, and data transforms in Python.* 2020.

[23] S. Han, X. Hu, H. Huang, M. Jiang, and Y. Zhao, "ADBench: Anomaly detection benchmark," in *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

[24] K.-H. Lai, D. Zha, J. Xu, Y. Zhao, G. Wang, and X. Hu, "Revisiting time series outlier detection: Definitions and benchmarks," in *Thirty-fifth Conference*

on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1), 2021.

[25] W. Zhou, T. D. Roy, and I. Skrypnyk, “The kdd cup 2019 report,” *ACM SIGKDD Explorations Newsletter*, 2020.

[26] Z. Luo, J. Huang, K. Hu, X. Li, and P. Zhang, “Accuair: Winning solution to air quality prediction for kdd cup 2018,” in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019.

[27] X. He, K. Zhao, and X. Chu, “Automl: A survey of the state-of-the-art,” *Knowledge-Based Systems*, 2021.

[28] J. Snoek, H. Larochelle, and R. P. Adams, “Practical bayesian optimization of machine learning algorithms,” *Advances in neural information processing systems*, 2012.

[29] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, “Auto-weka: Combined selection and hyperparameter optimization of classification algorithms,” in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013.

[30] R. S. Olson and J. H. Moore, “Tpot: A tree-based pipeline optimization tool for automating machine learning,” in *Workshop on automatic machine learning*, 2016.

[31] J. Liang, E. Meyerson, B. Hodjat, D. Fink, K. Mutch, and R. Miikkulainen, “Evolutionary neural automl for deep learning,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019.

[32] Y. Heffetz, R. Vainshtein, G. Katz, and L. Rokach, “Deepline: Automl tool for pipelines generation using deep reinforcement learning and hierarchical actions filtering,” in *KDD*, 2020.

[33] I. Drori, Y. Krishnamurthy, R. Rampin, R. d. P. Lourenco, J. P. Ono, K. Cho, C. Silva, and J. Freire, “Alphad3m: Machine learning pipeline synthesis,” *arXiv preprint arXiv:2111.02508*, 2021.

[34] G. Tesauro *et al.*, “Temporal difference learning and td-gammon,” *Communications of the ACM*, 1995.

[35] M. L. Puterman, “Markov decision processes,” *Handbooks in operations research and management science*, 1990.

[36] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, 2015.

[37] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015.

[38] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.

[39] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, “High-dimensional continuous control using generalized advantage estimation,” *arXiv preprint arXiv:1506.02438*, 2015.

[40] Y. Li, X. Huang, J. Li, M. Du, and N. Zou, “Specae: Spectral autoencoder for anomaly detection in attributed networks,” in *CIKM*, 2019.

[41] J. You, B. Liu, Z. Ying, V. Pande, and J. Leskovec, “Graph convolutional policy network for goal-directed molecular graph generation,” in *NeurIPS*, 2018.

[42] X. Hu, L. Tang, J. Tang, and H. Liu, “Exploiting social relations for sentiment analysis in microblogging,” in *WSDM*, 2013.

[43] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, “Neural collaborative filtering,” in *WWW*, 2017.

[44] W. L. Hamilton, R. Ying, and J. Leskovec, “Representation learning on graphs: Methods and applications,” 2017.

[45] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *IEEE*, 1998.

[46] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” in *ICLR*, 2017.

[47] M. Defferrard, X. Bresson, and P. Vandergheynst, “Convolutional neural networks on graphs with fast localized spectral filtering,” in *NeurIPS*, 2016.

[48] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, *et al.*, “Relational inductive biases, deep learning, and graph networks,” *arXiv preprint arXiv:1806.01261*, 2018.

[49] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, “Graph neural networks: A review of methods and applications,” *arXiv preprint*

arXiv:1812.08434, 2018.

[50] J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec, “Graphrnn: Generating realistic graphs with deep auto-regressive models,” *arXiv preprint arXiv:1802.08773*, 2018.

[51] X. Bresson and T. Laurent, “Residual gated graph convnets,” *arXiv preprint arXiv:1711.07553*, 2017.

[52] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, “Graph attention networks,” in *ICLR*, 2018.

[53] D. Zha, K.-H. Lai, K. Zhou, and X. Hu, “Experience replay optimization,” 2019.

[54] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” *Nature*, 2015.

[55] W. L. Hamilton, Z. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” in *NeurIPS*, 2017.

[56] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, “Weisfeiler-lehman graph kernels,” *JMLR*, 2011.

[57] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, “The graph neural network model,” *IEEE Transactions on Neural Networks*, 2009.

[58] H. Gao and S. Ji, “Graph u-nets,” *ICML*, 2019.

[59] J. Chen, T. Ma, and C. Xiao, "Fastgcn: Fast learning with graph convolutional networks via importance sampling," in *ICLR*, 2018.

[60] X. Huang, Q. Song, Y. Li, and X. Hu, "Graph recurrent networks with attributed random walks," in *KDD*, 2019.

[61] K. Zhou, Q. Song, X. Huang, D. Zha, N. Zou, and X. Hu, "Multi-channel graph convolutional networks," *arXiv preprint arXiv:1912.08306*, 2019.

[62] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[63] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," *arXiv preprint arXiv:1802.03268*, 2018.

[64] H. Jin, Q. Song, and X. Hu, "Auto-keras: Efficient neural architecture search with network morphism," *arXiv preprint arXiv:1806.10282*, 2018.

[65] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, *et al.*, "Evolving deep neural networks," in *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, 2019.

[66] Y. Li, D. Zha, P. Venugopal, N. Zou, and X. Hu, "Pyodds: An end-to-end outlier detection system with automated machine learning," in *WWW*, 2020.

[67] Y. Gao, H. Yang, P. Zhang, C. Zhou, and Y. Hu, "Graphnas: Graph neural architecture search with reinforcement learning," *arXiv preprint arXiv:1904.09981*, 2019.

[68] K. Zhou, Q. Song, X. Huang, and X. Hu, "Auto-gnn: Neural architecture search of graph neural networks," *arXiv preprint arXiv:1909.03184*, 2019.

[69] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint*, 2013.

[70] D. Zha, K.-H. Lai, Y. Cao, S. Huang, R. Wei, J. Guo, and X. Hu, "Rlcard: A toolkit for reinforcement learning in card games," *arXiv preprint arXiv:1910.04376*, 2019.

[71] K.-H. Lai, D. Zha, Y. Li, and X. Hu, "Dual policy distillation," *arXiv preprint arXiv:2006.04061*, 2020.

[72] Z. Xu, H. van Hasselt, and D. Silver, "Meta-gradient reinforcement learning," in *NeurIPS*, 2018.

[73] C. J. C. H. Watkins, *Learning from delayed rewards.* PhD thesis, King's College, 1989.

[74] G. Li, M. Müller, A. K. Thabet, and B. Ghanem, "Can gcns go as deep as cnns?," *arXiv preprint*, 2019.

[75] W. Huang, T. Zhang, Y. Rong, and J. Huang, "Adaptive sampling towards fast graph representation learning," in *NeurIPS*, 2018.

[76] H. Gao, Z. Wang, and S. Ji, "Large-scale learnable graph convolutional networks," in *KDD*, 2018.

[77] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *NeurIPS*,

2013.

[78] B. Perozzi, R. Al-Rfou, and S. Skiena, “Deepwalk: online learning of social representations,” in *KDD*, 2014.

[79] A. Grover and J. Leskovec, “node2vec: Scalable feature learning for networks,” in *KDD*, 2016.

[80] W. Hamilton, Z. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” in *NeurIPS*, 2017.

[81] S. Das, W.-K. Wong, T. Dietterich, A. Fern, and A. Emmott, “Incorporating expert feedback into active anomaly discovery,” in *ICDM*, 2016.

[82] S. Das, W.-K. Wong, A. Fern, T. G. Dietterich, and M. A. Siddiqui, “Incorporating feedback into tree-based anomaly detection,” *arXiv preprint arXiv:1708.09441*, 2017.

[83] M. A. Siddiqui, A. Fern, T. G. Dietterich, R. Wright, A. Theriault, and D. W. Archer, “Feedback-guided anomaly discovery via online optimization,” in *KDD*, 2018.

[84] H. Lamba and L. Akoglu, “Learning on-the-job to re-rank anomalies from top-1 feedback,” in *SDM*, 2019.

[85] B. Settles, “Active learning literature survey,” tech. rep., University of Wisconsin-Madison Department of Computer Sciences, 2009.

[86] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, “Lof: identifying density-based local outliers,” in *SIGMOD*, 2000.

[87] S. Ramaswamy, R. Rastogi, and K. Shim, "Efficient algorithms for mining outliers from large data sets," in *SIGMOD*, 2000.

[88] F. Angiulli and C. Pizzuti, "Fast outlier detection in high dimensional spaces," in *ECML PKDD*, 2002.

[89] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *ICDM*, 2008.

[90] J. Chen, S. Sathe, C. Aggarwal, and D. Turaga, "Outlier detection with autoencoder ensembles," in *SDM*, 2017.

[91] G. Pang, L. Cao, L. Chen, D. Lian, and H. Liu, "Sparse modeling-based sequential ensemble learning for effective outlier detection in high-dimensional numeric data," in *AAAI*, 2018.

[92] L. Akoglu, H. Tong, J. Vreeken, and C. Faloutsos, "Fast and reliable anomaly detection in categorical data," in *CIKM*, 2012.

[93] M. Gupta, J. Gao, C. C. Aggarwal, and J. Han, "Outlier detection for temporal data: A survey," *IEEE Transactions on Knowledge and Data Engineering*, 2013.

[94] L. Akoglu, H. Tong, and D. Koutra, "Graph based anomaly detection and description: a survey," *Data mining and knowledge discovery*, 2015.

[95] Y. Zhao, Z. Nasrullah, and Z. Li, "Pyod: A python toolbox for scalable outlier detection," *arXiv preprint arXiv:1901.01588*, 2019.

[96] X. J. Zhu, "Semi-supervised learning literature survey," tech. rep., University of Wisconsin-Madison Department of Computer Sciences, 2005.

[97] D. Zha and C. Li, "Multi-label dataless text classification with topic modeling," *Knowledge and Information Systems*, 2019.

[98] Y. Zhao and M. K. Hryniewicki, “Xgbod: improving supervised outlier detection with unsupervised representation learning,” in *IJCNN*, 2018.

[99] A. Tamersoy, K. Roundy, and D. H. Chau, “Guilt by association: large scale malware detection by mining file-relation graphs,” in *KDD*, 2014.

[100] G. Pang, L. Cao, L. Chen, and H. Liu, “Learning representations of ultrahigh-dimensional data for random distance-based outlier detection,” in *KDD*, 2018.

[101] N. Görnitz, M. Kloft, K. Rieck, and U. Brefeld, “Toward supervised anomaly detection,” *Journal of Artificial Intelligence Research*, 2013.

[102] K. Veeramachaneni, I. Arnaldo, V. Korrapati, C. Bassias, and K. Li, “Aiˆ 2: training a big data machine to defend,” in *BigDataSecurity*, 2016.

[103] Y. Li, Z. Chen, D. Zha, K. Zhou, H. Jin, H. Chen, and X. Hu, “Autood: Automated outlier detection via curiosity-guided search and self-imitation learning,” *arXiv preprint arXiv:2006.11321*, 2020.

[104] L. Ruff, R. A. Vandermeulen, N. Görnitz, A. Binder, E. Müller, K.-R. Müller, and M. Kloft, “Deep semi-supervised anomaly detection,” *arXiv preprint arXiv:1906.02694*, 2019.

[105] D. A. Cohn, Z. Ghahramani, and M. I. Jordan, “Active learning with statistical models,” *Journal of artificial intelligence research*, 1996.

[106] H. T. Nguyen and A. Smeulders, “Active learning using pre-clustering,” in *ICML*, 2004.

[107] J. He and J. G. Carbonell, “Nearest-neighbor-based active learning for rare category detection,” in *NeurIPS*, 2008.

[108] D. Zhou, J. He, H. Yang, and W. Fan, "Sparc: Self-paced network representation for few-shot rare category characterization," in *KDD*, 2018.

[109] S. Das, M. R. Islam, N. K. Jayakodi, and J. R. Doppa, "Active anomaly detection via ensembles," *arXiv preprint arXiv:1809.06477*, 2018.

[110] K. Ding, J. Li, and H. Liu, "Interactive anomaly detection on attributed networks," in *WSDM*, 2019.

[111] K.-H. Lai, D. Zha, K. Zhou, and X. Hu, "Policy-gnn: Aggregation optimization for graph neural networks," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020.

[112] L. Duong, H. Afshar, D. Estival, G. Pink, P. Cohen, and M. Johnson, "Active learning for deep semantic parsing," in *ACL*, 2018.

[113] G. Dulac-Arnold, R. Evans, H. van Hasselt, P. Sunehag, T. Lillicrap, J. Hunt, T. Mann, T. Weber, T. Degris, and B. Coppin, "Deep reinforcement learning in large discrete action spaces," *arXiv preprint arXiv:1512.07679*, 2015.

[114] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* 2018.

[115] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *ICML*, 2018.

[116] V. Vercruyssen, M. Wannes, V. Gust, M. Koen, B. Ruben, and D. Jesse, "Semi-supervised anomaly detection with an application to water analytics," in *ICDM*, 2018.

[117] T. Chen, X. Liu, B. Xia, W. Wang, and Y. Lai, "Unsupervised anomaly detection of industrial robots using sliding-window convolutional variational autoencoder," *IEEE Access*, 2020.

[118] D. Kim, H. Yang, M. Chung, S. Cho, H. Kim, M. Kim, K. Kim, and E. Kim, "Squeezed convolutional variational autoencoder for unsupervised anomaly detection in edge device industrial internet of things," in *2018 International Conference on Information and Computer Technologies (icict)*, 2018.

[119] K. Al Jallad, M. Aljnidi, and M. S. Desouki, "Anomaly detection optimization using big data and deep learning to reduce false-positive," *Journal of Big Data*, 2020.

[120] L. Ruff, R. A. Vandermeulen, N. Görnitz, A. Binder, E. Müller, K.-R. Müller, and M. Kloft, "Deep semi-supervised anomaly detection," in *International Conference on Learning Representations*, 2020.

[121] G. Pang, C. Shen, H. Jin, and A. v. d. Hengel, "Deep weakly-supervised anomaly detection," *arXiv preprint arXiv:1910.13601*, 2019.

[122] G. Pang, C. Shen, L. Cao, and A. V. D. Hengel, "Deep learning for anomaly detection: A review," *ACM Computing Surveys (CSUR)*, 2021.

[123] G. Pang, C. Shen, and A. van den Hengel, "Deep anomaly detection with deviation networks," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019.

[124] G. Pang, A. van den Hengel, C. Shen, and L. Cao, "Toward deep supervised anomaly detection: Reinforcement learning from partially labeled anomaly

data," in *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, 2021.

[125] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

[126] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," in *International Conference on Learning Representations*, 2018.

[127] A. Dabouei, S. Soleymani, F. Taherkhani, and N. M. Nasrabadi, "Supermix: Supervising the mixing data augmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[128] D. Ienco, R. G. Pensa, and R. Meo, "A semisupervised approach to the detection and characterization of outliers in categorical data," *IEEE transactions on neural networks and learning systems*, 2016.

[129] O. Chapelle, B. Scholkopf, and A. Zien, "Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]," *IEEE Transactions on Neural Networks*, 2009.

[130] Y. Ouali, C. Hudelot, and M. Tami, "An overview of deep semi-supervised learning," *arXiv preprint arXiv:2006.05278*, 2020.

[131] S. Y. Feng, V. Gangal, J. Wei, S. Chandar, S. Vosoughi, T. Mitamura, and E. Hovy, "A survey of data augmentation approaches for nlp," *arXiv preprint arXiv:2105.03075*, 2021.

[132] Q. Wen, L. Sun, F. Yang, X. Song, J. Gao, X. Wang, and H. Xu, "Time series data augmentation for deep learning: A survey," *arXiv preprint arXiv:2002.12478*, 2020.

[133] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of big data*, 2019.

[134] X.-Y. Liu and Z.-H. Zhou, "The influence of class imbalance on cost-sensitive learning: An empirical study," in *ICDM*, 2006.

[135] Z. Cao, K. You, M. Long, J. Wang, and Q. Yang, "Learning to transfer examples for partial domain adaptation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

[136] M. Li, X. Zhang, C. Thrampoulidis, J. Chen, and S. Oymak, "Autobalance: Optimized loss functions for imbalanced data," *NeurIPS*, 2021.

[137] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, 2002.

[138] H. He, Y. Bai, E. A. Garcia, and S. Li, "Adasyn: Adaptive synthetic sampling approach for imbalanced learning," in *IJCNN*, 2008.

[139] G. Kovács, "Smote-variants: A python implementation of 85 minority oversampling techniques," *Neurocomputing*, 2019.

[140] C. Bunkhumpornpat, K. Sinapiromsaran, and C. Lursinsap, "Dbsmote: density-based synthetic minority over-sampling technique," *Applied Intelligence*, 2012.

[141] H. Han, W.-Y. Wang, and B.-H. Mao, "Borderline-smote: a new over-sampling method in imbalanced data sets learning," in *ICIC*, 2005.

[142] D. Zha, K.-H. Lai, Q. Tan, S. Ding, N. Zou, and X. B. Hu, "Towards automated imbalanced learning with deep hierarchical reinforcement learning," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, 2022.

[143] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," *arXiv preprint arXiv:1710.09412*, 2017.

[144] A. Kabra, A. Chopra, N. Puri, P. Badjatiya, S. Verma, P. Gupta, *et al.*, "Mixboost: Synthetic oversampling with boosted mixup for handling extreme imbalance," *arXiv preprint arXiv:2009.01571*, 2020.

[145] T. Wu and J. Ortiz, "Rlad: Time series anomaly detection through reinforcement learning and active learning," *arXiv preprint arXiv:2104.00543*, 2021.

[146] D. Zha, K.-H. Lai, M. Wan, and X. Hu, "Meta-aad: Active anomaly detection with deep reinforcement learning," in *2020 IEEE International Conference on Data Mining (ICDM)*, 2020.

[147] L. Perini, V. Vercruyssen, and J. Davis, "Quantifying the confidence of anomaly detectors in their example-wise predictions," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2020.

[148] J. Vanschoren, J. N. Van Rijn, B. Bischl, and L. Torgo, "Openml: networked science in machine learning," *ACM SIGKDD Explorations Newsletter*, 2014.

[149] S. Rayana, "Odds library," 2016.

[150] H. M. Nguyen, E. W. Cooper, and K. Kamei, “Borderline over-sampling for imbalanced data classification,” *International Journal of Knowledge Engineering and Soft Data Paradigms*, 2011.

[151] A. Nguyen, J. Yosinski, and J. Clune, “Deep neural networks are easily fooled: High confidence predictions for unrecognizable images,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015.

[152] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, “Measuring and relieving the over-smoothing problem for graph neural networks from the topological view,” in *Proceedings of the AAAI conference on artificial intelligence*, 2020.

[153] Q. P. He and J. Wang, “Fault detection using the k-nearest neighbor rule for semiconductor manufacturing processes,” *IEEE transactions on semiconductor manufacturing*, 2007.

[154] L. MacEachern and G. Vazhbakht, “Configurable fpga-based outlier detection for time series data,” in *2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2020.

[155] J. Liang and R. Du, “Model-based fault detection and diagnosis of hvac systems using support vector machine method,” *International Journal of refrigeration*, 2007.

[156] T. P. Carvalho, F. A. Soares, R. Vita, R. d. P. Francisco, J. P. Basto, and S. G. Alcalá, “A systematic literature review of machine learning methods applied to predictive maintenance,” *Computers & Industrial Engineering*, 2019.

[157] R. K. Mobley, *An introduction to predictive maintenance.* 2002.

[158] P. J. Rousseeuw and A. M. Leroy, *Robust regression and outlier detection.* 2005.

[159] L. Bontemps, J. McDermott, N.-A. Le-Khac, *et al.*, "Collective anomaly detection based on long short-term memory recurrent neural networks," in *International Conference on Future Data and Security Engineering*, 2016.

[160] M. Sakurada and T. Yairi, "Anomaly detection using autoencoders with nonlinear dimensionality reduction," in *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, 2014.

[161] S. Zolhavarieh, S. Aghabozorgi, and Y. W. Teh, "A review of subsequence time series clustering," *The Scientific World Journal*, 2014.

[162] C.-C. M. Yeh, Y. Zhu, L. Ulanova, N. Begum, Y. Ding, H. A. Dau, D. F. Silva, A. Mueen, and E. Keogh, "Matrix profile i: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets," in *2016 IEEE 16th international conference on data mining (ICDM)*, 2016.

[163] R. Wu and E. Keogh, "Current time series anomaly detection benchmarks are flawed and are creating the illusion of progress," *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[164] M. Milutinovic, B. Schoenfeld, D. Martinez-Garcia, S. Ray, S. Shah, and D. Yan, "On evaluation of automl systems," in *ICML Workshop*, 2020.

[165] K.-H. Lai, D. Zha, G. Wang, J. Xu, Y. Zhao, D. Kumar, Y. Chen, P. Zumkhawaka, M. Wan, D. Martinez, *et al.*, "Tods: An automated time series outlier detection system," *arXiv preprint arXiv:2009.09822*, 2020.

[166] S. Elsayed, D. Thyssens, A. Rashed, L. Schmidt-Thieme, and H. S. Jomaa, "Do we really need deep learning models for time series forecasting?," *arXiv preprint arXiv:2101.02118*, 2021.

[167] B. Schölkopf, A. J. Smola, F. Bach, *et al.*, *Learning with kernels: support vector machines, regularization, optimization, and beyond.* 2002.

[168] I. Steinwart, D. Hush, and C. Scovel, "A classification framework for anomaly detection.," *Journal of Machine Learning Research*, 2005.

[169] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, J. C. Platt, *et al.*, "Support vector method for novelty detection.," in *NIPS*, 1999.

[170] R. Zhang, S. Zhang, Y. Lan, and J. Jiang, "Network anomaly detection using one class support vector machine," in *Proceedings of the International Multi-Conference of Engineers and Computer Scientists*, vol. 1, 2008.

[171] J. Ma and S. Perkins, "Time-series novelty detection using one-class support vector machines," in *Proceedings of the International Joint Conference on Neural Networks, 2003.*, 2003.

[172] Z. Ding and M. Fei, "An anomaly detection approach based on isolation forest algorithm for streaming data using sliding window," *IFAC Proceedings Volumes*, 2013.

[173] Y. Liu, Z. Li, C. Zhou, Y. Jiang, J. Sun, M. Wang, and X. He, "Generative adversarial active learning for unsupervised outlier detection," *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[174] P. Senin, “Dynamic time warping algorithm review,” *Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA*, 2008.

[175] Y. Zhu, C.-C. M. Yeh, Z. Zimmerman, K. Kamgar, and E. Keogh, “Matrix profile xi: Scrimp++: time series motif discovery at interactive speeds,” in *2018 IEEE International Conference on Data Mining (ICDM)*, 2018.

[176] M. Barandas, D. Folgado, L. Fernandes, S. Santos, M. Abreu, P. Bota, H. Liu, T. Schultz, and H. Gamboa, “Tsfel: Time series feature extraction library,” *SoftwareX*, 2020.

[177] X. Hou and L. Zhang, “Saliency detection: A spectral residual approach,” in *2007 IEEE Conference on computer vision and pattern recognition*, 2007.

[178] J. Mei, Y. De Castro, Y. Goude, and G. Hébrail, “Nonnegative matrix factorization for time series recovery from a few temporal aggregates,” in *International Conference on Machine Learning*, 2017.

[179] H.-F. Yu, N. Rao, and I. S. Dhillon, “Temporal regularized matrix factorization for high-dimensional time series prediction,” in *Advances in neural information processing systems*, 2016.

[180] V. C. Cheung, K. Devarajan, G. Severini, A. Turolla, and P. Bonato, “Decomposing time series data by a non-negative matrix factorization algorithm with temporally constrained coefficients,” in *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2015.

[181] D. Li, D. Chen, B. Jin, L. Shi, J. Goh, and S.-K. Ng, “Mad-gan: Multivariate

anomaly detection for time series data with generative adversarial networks," in *International Conference on Artificial Neural Networks*, 2019.

[182] Y. Zhao and M. K. Hryniewicki, "Dcso: dynamic combination of detector scores for outlier ensembles," *arXiv preprint arXiv:1911.10418*, 2019.

[183] J. Rapin, P. Bennet, E. Centeno, D. Haziza, A. Moreau, and O. Teytaud, "Open source evolutionary structured optimization," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, 2020.

[184] J. Demšar, B. Zupan, G. Leban, and T. Curk, "Orange: From experimental machine learning to interactive data mining," in *PKDD*, 2004.

[185] Y.-N. Chuang, G. Wang, F. Yang, Q. Zhou, P. Tripathi, X. Cai, and X. Hu, "Cortx: Contrastive framework for real-time explanation," *arXiv preprint arXiv:2303.02794*, 2023.

[186] G. Wang, Y.-N. Chuang, M. Du, F. Yang, Q. Zhou, P. Tripathi, X. Cai, and X. Hu, "Accelerating shapley explanation via contributive cooperator selection," in *International Conference on Machine Learning*, 2022.