A Novel Active Anomaly Discovery Method and its Applications in Additive Manufacturing

This paper was downloaded from TechRxiv (<https://www.techrxiv.org>).

LICENSE

CC BY 4.0

SUBMISSION DATE / POSTED DATE

24-09-2021 / 03-03-2022

CITATION

Shen, Bo; Kong, Zhenyu (2021): A Novel Active Anomaly Discovery Method and its Applications in Additive Manufacturing. TechRxiv. Preprint. <https://doi.org/10.36227/techrxiv.16674412.v2>

DOI

[10.36227/techrxiv.16674412.v2](https://doi.org/10.36227/techrxiv.16674412.v2)

A Novel Active Anomaly Discovery Method and its Applications in Additive Manufacturing

Bo Shen^a and Zhenyu (James) Kong^{a*}

^aGrado Department of Industrial and Systems Engineering, Virginia Tech, Blacksburg, US

*Corresponding Author

Abstract

Anomaly detection aims to identify the true anomalies from a given set of data instances. Unsupervised anomaly detection algorithms are applied to an unlabeled dataset by producing a ranked list based on anomaly scores. Unfortunately, due to the inherent limitations, many of the top-ranked instances by unsupervised algorithms are not anomalies or not interesting from an application perspective, which leads to high false-positive rates. Active anomaly discovery (AAD) is proposed to overcome this deficiency, which sequentially selects instances to get the labeling information. The labeling information is used to update the anomaly detection algorithm to improve the detection accuracy. However, labeling is often costly. Therefore, the way to balance detection accuracy and labeling cost is essential. Along this line, this paper proposes a novel AAD method to achieve the goal. Our approach is based on the state-of-the-art unsupervised anomaly detection algorithm, namely, Isolation Forest, as the baseline anomaly detector to extract features. Thereafter, the sparsity of the extracted features is utilized to adjust the anomaly detector so that it can focus on more important features for anomaly detection. To enforce the sparsity of the features and subsequent improvement of the detection accuracy, a new algorithm based on online gradient descent, namely, Sparse Approximated Linear Anomaly Discovery (SALAD), is proposed with its theoretical Regret analysis. Extensive experiments on both open-source and additive manufacturing datasets demonstrate that the proposed algorithm significantly outperforms the state-of-the-art algorithms for anomaly detection.

Keywords: Isolation Forest; Sparsity; Active Anomaly Discovery; Measurement Feedback; Online Gradient Descent; Additive Manufacturing.

1 Introduction

Anomaly detection has been an active research area for several decades and plays increasingly important roles in various applications, for example, process monitoring in additive manufacturing [Shen et al. \(2021a,b\)](#), cyber-attack in smart manufacturing [Ten et al. \(2011\)](#), and healthcare analytics in IoT [Ukil et al. \(2016\)](#). The goal of anomaly detection is to identify anomalies, which have patterns in data that do not conform to a well-defined notion of normal behavior [Chandola et al. \(2009\)](#). An *anomaly* is a data instance generated by

a process different from generating the “normal” data instances, referred to as *nominal* instances. As a learning task, anomaly detection can be categorized into supervised, semi-supervised, or unsupervised [Liu et al. \(2008\)](#); [An and Cho \(2015\)](#); [Görnitz et al. \(2013\)](#). Supervised anomaly detection is the same thing as binary classifications, which is trained on both nominal and anomaly instances. In semi-supervised anomaly detection, anomaly detectors are only trained on nominal instances. In real-world applications, a labeled dataset is not available for training since labeling is time-consuming or costly. As a viable solution, unsupervised anomaly detection algorithms (e.g., [Breunig et al. \(2000\)](#); [Knorr and Ng \(1998\)](#); [Liu et al. \(2008\)](#)) can help address this problem to some extent.

The unsupervised anomaly detection algorithms work on an unlabeled dataset directly to provide an anomaly score list where the anomalies are supposed to have a higher rank than the nominal data. The top-ranked instances are identified as candidate anomalies. Since labeling is not needed for the algorithm itself, the time and cost-saving are the most appealing aspect of this solution. However, unsupervised anomaly detection algorithms generally do not perform well due to their deficiency of the high false-positive rate, which significantly constrains their broad applications.

Figure 1: High-level overview of the active anomaly discovery framework.

The high false-positive rate of the unsupervised anomaly detection motivates a new research direction, namely, active anomaly discovery (AAD) [Das et al. \(2017\)](#); [Siddiqui et al. \(2018\)](#); [Das et al. \(2019, 2020\)](#). As shown in Figure 1, active anomaly detector elaborately selects one data instances to label. Then, labeling is obtained through inspections or measurements, which incurs the cost and is thus needed to be minimized or under a certain budget. Based on the label feedback, the weight for active anomaly detector is updated to

improve the anomaly detection performance. The goal of AAD is to sequentially select some data instances to measure and then label (termed “query” in this paper), and maximize the number of detected anomalies under the given number of measurements constrained by the budget.

Among all AAD methods, IF-AAD [Das et al. \(2017\)](#) and Generalized Linear Anomaly Detector (GLAD) [Siddiqui et al. \(2018\)](#) are the state-of-the-art methods. Both apply Isolation Forest [Liu et al. \(2008\)](#) as a feature extraction method to learn a feature function $\phi(\cdot)$, which maps the data instances to high dimensional binary features. At each step, the IF-AAD approach [Das et al. \(2017\)](#) solves an optimization problem based on all prior measurement feedback, which results in new weights for the model components. The optimization in this approach involves a number of components associated with hyperparameters. In addition, this optimization problem is non-convex and requires a relatively complex “alternating” convex optimization approach with its own hyperparameters. Both hyperparameters should be optimized for a good performance of IF-AAD, leading to another challenging problem that needs to be solved. This issue makes IF-AAD difficult to be applied in practice.

Rather, this motivates the GLAD method proposed in [Siddiqui et al. \(2018\)](#), which uses a simpler and more efficient approach with a minimal number of parameters. Specifically, they formulated the AAD problem within the framework of online convex optimization. At each step of GLAD: 1) the model weight is updated through online mirror descent [Hazan \(2019\)](#), and 2) Select an instance with the highest anomaly score from all unlabeled instances based on the new weight for investigation. This step yields an efficient and relatively simple approach to incorporating feedback compared to the prior state-of-the-art [Das et al. \(2017\)](#). GLAD has been proven to be one of the best anomaly detection algorithms for large-scale datasets based on [Das et al. \(2019\)](#)¹.

Although [Siddiqui et al. \(2018\)](#) pointed out that the set of features from $\phi(\cdot)$ is very sparse for any particular instance, they did not explore the sparsity further. In this paper, the feature function $\phi(\cdot)$ is proven to be sparse theoretically and verified empirically. We then hypothesized that utilizing the sparsity of extracted features could improve the GLAD algorithm for two reasons. First, model sparsity can make the anomaly detection algorithm

¹see performance comparison in Appendix B of [Das et al. \(2019\)](#)

more focused on the essential features (edges in the forest); Second, model sparsity can significantly reduce the model complexity, which makes learning more tractable, since there are only a small number of training instances. A new algorithm based on gradient descent, namely, Sparse Approximated Linear Anomaly Discovery (SALAD), is proposed in this paper. The sparsity is achieved by a sparse approximated solution after the gradient descent update at each online round. The key of SALAD is to control the error between the exact solution and the approximated one, which captures the tradeoff between the sparsity of the model and the regret performance of the algorithm.

This paper aims to solve two practical problems in additive manufacturing (AM) using our proposed SALAD: (1) detect a specific type of anomalies from an unlabeled dataset, and (2) measurement allocation under a certain budget. In Sections 5.3 and 5.4, polymer Liu et al. (2019) and metal AM Wang et al. (2021) are used to demonstrate the superior performance of the proposed SALAD in solving Problems (1) and (2), respectively.

The remainder of this paper is organized as follows. The baseline anomaly detector, namely, Isolation Forest, is introduced in Section 2. The research framework of this paper is provided in Section 3. The sparsity analysis of feature function from Isolation Forest is presented in Section 4.1, followed by the proposed Sparse Approximated Linear Anomaly Discovery in Section 4.2. The numerical and real-world case studies are provided in Section 5 to test and validate the proposed algorithm. Finally, the conclusions and future work are discussed in Section 6.

Remark 1. *Our work is actually active learning Settles (2009) for anomaly detection. In active learning, there is a query strategy to select instances that provide the most useful information (quantified by the acquisition function) for learning. In this work, a simple greedy instance selection mechanism is used based on empirical results. Our emphasis is how to best adapt the anomaly detector in response to feedback. In future work, a query strategy that can tradeoff exploration and exploitation may further improve our method.*

2 Theoretical Background of Isolation Forest

In this Section, Isolation Forest proposed by Liu et al. (2008, 2012) will be introduced as our baseline anomaly detector. Given a dataset $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_m\}$, where each instance

$\mathbf{x}_i \in \mathbb{R}^d$ contains d features $\{\mathbf{x}_{i1}, \dots, \mathbf{x}_{id}\}$. Isolation Forest comprises an ensemble of K trees denoted by $\mathbf{T} = \{T_1, \dots, T_K\}$, where T_k is k th tree in the forest. Each tree is constructed in a randomized manner as illustrated in Figure 2, and outlined in Algorithm 1. At first, a sub-sample dataset $\mathbf{X}_{S_k} = \{\mathbf{x}_i, i \in S_k\}$ is used to construct tree T_k , where $S_k \subset \{1, \dots, m\}$ is the index set with sub-sample size $|S_k| = \psi \leq m$. Each tree T_k is built from the root to leaves by randomly partitioning the data instance at each node via selecting a feature and a threshold both at uniformly random. For example, let $U = S_k$ be the index set for the instances at the root node (node N_0 in Figure 2). $f \in \{1, \dots, d\}$ is randomly selected as the index for the feature and a threshold p_f is sampled uniformly from $[f_{min}, f_{max}]$, where $f_{min} = \min\{\mathbf{x}_{if}, i \in U = S_k\}$ and $f_{max} = \max\{\mathbf{x}_{if}, i \in U = S_k\}$. Now, let $U = \{i \in S_k, \mathbf{x}_{if} \geq p_f\}$ is the index set for instances that go through edge e_1 to node N_1 while the remaining instances go through edge e_2 to node N_2 , as shown in Figure 2. The same procedure is performed for the instances nodes N_1 and N_2 . The trees are grown until each instance is isolated in a leaf or reaches the tree height limit H , where the tree height limit is the maximum number of edges that an instance can travel.

Figure 2: Illustration of the construction of isolation tree T_k .

Isolation Forest is based on the idea that anomaly instances are few and well-separated from clusters of nominal instances in the feature space. Because of this, anomaly instances (represented as a red circle in Figure 2) very quickly reach leaf nodes through random partitioning. On the other hand, nominal instances (represented as a green circle in Figure

Algorithm 1: Generating randomized trees in Isolation Forest

Input: $\mathbf{X}$; sub-sample size: ψ ; number of trees: K ; $\mathbf{T} = \emptyset$; Tree height limit: H .

Output: Isolation Forest $\mathbf{T} = \{T_1, \dots, T_K\}$

```
1 for  $k=1, 2, \dots, K$  do
2   Build tree  $T_k$ : let  $S_k$  = a sub-sample of  $\psi$  instances from  $\mathbf{X}$ 
3   Start with all instances in  $S_k$  at the root node
4   Let  $U \subset S_k$  be the index set of instances at the current node
5   if  $|U| == 1$  or Reach height limit  $H$  then
6     | return
7   else
8     | Let  $f$  be a index of the feature sampled at random
9     | Let  $f_{min} = \min\{\mathbf{x}_{if}, i \in U\}$  and  $f_{max} = \max\{\mathbf{x}_{if}, i \in U\}$ 
10    | Let  $p_f$  = value sampled unif. random in  $[f_{min}, f_{max}]$ 
11    | Partition  $U$  into two parts on the basis of  $p_f$  and recurse on both partitions
12   $\mathbf{T} = \mathbf{T} \cup T_k$ 
```

2), which form dense clusters, require many more steps to reach leaf nodes finally. Therefore, the number of edges traversed by an instance from the root node to the leaf, also known as the isolation depth, is shorter (on average) for anomaly instances than for nominal instances. For example, the isolation depth of anomaly instance (red circle in Figure 2) is 2. The *anomaly score* assigned to an instance is simply the average isolation depth across K trees in the forest.

Isolation Forest can be described as a particular way of setting the weights of a tree-based anomaly detector through a feature function $\phi(\cdot) : \mathbb{R}^d \rightarrow \{0, 1\}^D$, where D is the number of edges in the forest. Specifically, the feature function $\phi(\cdot)$ can be constructed as follows: for each edge e in the forest with K trees,

$$\phi_e(\mathbf{x}) = \begin{cases} 1, & \text{if instance } \mathbf{x} \text{ goes through the edge } e \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

For example, the red circle $\mathbf{x}$ in Figure 2, $\phi_{e_1}(\mathbf{x}) = \phi_{e_2}(\mathbf{x}) = 1$ while $\phi_{e_3}(\mathbf{x}) = 0$. By doing so, $\phi(\mathbf{x})$ is a very sparse feature vector where most features are zero given that each

instance can only travel a small number of edges. Similarly, a weight $\mathbf{w}_e$ is assigned to each edge to construct an anomaly score. Now let ϕ and $\mathbf{w}$ be vectors that concatenate all of the features and weights across the forest consistently for different instances, and $\mathbf{w}_e = 1$ initially for all e . The linear anomaly score function

$$\text{score}(\mathbf{x}, \mathbf{w}) = -\phi(\mathbf{x}) \cdot \mathbf{w} \quad (2)$$

corresponds exactly to the (unnormalized) Isolation Forest *anomaly score*. The main purpose of making scores negative is to ensure that higher scores indicate more anomalous and lower scores indicate more nominal.

Remark 2. *In the literature of unsupervised anomaly detection, Isolation Forest has been proven to be one of the most robust methods Domingues et al. (2018); Emmott et al. (2013); Zhao et al. (2019)², which is the top-performer for different datasets with different sizes and feature spaces. This is the reason why Isolation Forest is used as the baseline learner.*

3 Research Framework

In the setting of this paper, m data instances $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_m\}$ are considered as a dataset. $\mathbf{x}_i \in \mathbb{R}^d$ is a data instance that is not given with any labels initially. But, it can be measured with a certain cost and then labeled as either anomaly or nominal class, namely, y_i belonging to $\{-1, +1\}$. A query is defined as the procedure to select the data instance and take a measurement for labeling. Instances labeled with $+1$ represent the *anomaly* class $\mathbf{A}$. The label of -1 represents the *nominal* class $\mathbf{N}$. Generally, the nominal instances account for an overwhelming fraction of the data, that is, $|\mathbf{N}| \gg |\mathbf{A}|$. Besides, $\mathcal{L}$ and $\mathcal{U}$ denote the set of labeled and unlabeled instances, respectively.

Our objective is to develop a sampling procedure to sequentially query the unlabeled data instances, which maximizes the number of anomalies to be detected during the querying procedure (i.e., let the set of labeled instances $\mathcal{L}$ contain as many anomalies as possible) under a given amount of budget (i.e., a fixed number of measurements). Figure 3 summarizes the framework of our method, which is a two-step procedure. The output from this procedure is the measurements that need to be taken to maximize the probability of capturing anomalies.

²The Best-in-class performance that generalizes: <https://pyod.readthedocs.io/en/latest/benchmark.html>

Figure 3: Overview of the framework of the proposed method in this paper.

- Step 1 (Sections 2 and 4.1): train Isolation Forest on data $\mathbf{X}$ (for example, the 2D images from the metal AM application in Section 5.4) as the baseline anomaly detector to obtain its feature function $\phi(\cdot)$. The feature function $\phi(\cdot)$ can be used to calculate the anomaly score for any single instance.
 - The anomaly score is utilized to identify top candidates for queries by taking measurements since anomalies are supposed to have high anomaly scores.
 - The foundation of Step 2 is the sparsity of feature function $\phi(\cdot)$, which is proved in Section 4.1.
- Step 2 (Section 4.2): A new measurement is executed to provide the label for the queried instance (for example, 3D scanner data from the metal AM application). Then the model weight $\mathbf{w}$ is updated through the proposed SALAD in Section 4.2 with a sparsity constraint to adapt the anomaly detector. Specifically, the following loop proceeds until the budget (the maximum number of measurements) is reached:
 - (1) Based on the current anomaly scores calculated from model weight $\mathbf{w}$, select one top-ranked instance from the unlabeled dataset $\mathcal{U}$;
 - (2) A new measurement is taken on the queried instance (i.e., provide the label);
 - (3) Update the model weight $\mathbf{w}$ based on the new labeled instance, where $\mathbf{w}$ is constrained to be sparse.

The above two-step procedure can be understood as follows. In Step 1, the Isolation Forest is trained as anomaly detector without label information as the initialization of our method.

In Step 2, the anomaly detector is updated to make better decision, in response to the label feedback by adjusting the model weight $\mathbf{w}$.

4 Methodology

As illustrated in Figure 3, the sparsity analysis of unsupervised anomaly detection method in Step 1, namely, the Isolation Forest, is introduced in Section 4.1. Based on Isolation Forest, the proposed SALAD in Step 2 is presented in Section 4.2, which can adapt the Isolation Forest anomaly detector.

4.1 Sparsity Analysis of Feature Function $\phi(\cdot)$

This subsection discusses and proves the sparsity of feature function $\phi(\cdot)$ from the Isolation Forest. Once the dataset $\mathbf{X}$ is trained using Algorithm 1, the feature function $\phi : \mathbb{R}^d \rightarrow \{0, 1\}^D$ can be obtained through the process described in equation (1). The dimension of the binary feature, namely, D , is closely related to the tree height limit H . Typically, D is increased as increased H . The following lemma proves that the binary features from the Isolation Forest feature function $\phi(\cdot)$ are indeed sparse under two conditions.

Lemma 1. *Assume that there is no empty leaf in the forest trained by Algorithm 1 on m instances, where there are K trees in the forest and $\phi(\cdot)$ is constructed through equation (1).*

- (i) *If the tree height limit H is small (i.e., not all instances are isolated in a leaf), then for any $\mathbf{x} \in \mathbf{X}$*

$$\|\phi(\mathbf{x})\|_0 \leq H \cdot K; \quad (3)$$

- (ii) *If the tree height limit H is large (i.e., all instances are isolated in a leaf) and $\psi = m$, then*

$$\frac{\lfloor \log m \rfloor}{2(m-1)} \leq r \leq \frac{m+2}{4m}, \quad (4)$$

where m is the total number of instances in the dataset, $r = \frac{\sum_{i=1}^m \|\phi(\mathbf{x}_i)\|_0}{mD}$ denotes the ratio of nonzero elements that is calculated from the extracted feature set $\{\phi(\mathbf{x}_1), \dots, \phi(\mathbf{x}_m)\}$, and $D = 2(m-1)K$.

Proof. **Under Condition (i):** If the tree height limit is set to H , then the number of edges that each instance can travel in each tree is upper bounded by H . Since there are K trees in the forest, then $\|\phi(\mathbf{x})\|_0 \leq H \cdot K$.

Under Condition (ii): The dimension of feature function $\phi(\cdot)$, namely, D , is first considered. In the case where all instances are isolated in a leaf in a tree, the number of edges in each tree increases by 2 when a new instance is added to the dataset. When $m = 2$, $D = 2K$. Therefore, the dimension of feature function D is $\phi(\cdot)$ is $2(m - 1)K$.

The upper bound in inequality (4) is proved by induction. When $m = 2$, the $\sum_{i=1}^m \|\phi(\mathbf{x}_i)\|_0 = (1+1)K$; When $m = 3$, the $\sum_{i=1}^m \|\phi(\mathbf{x}_i)\|_0 = (1+2+2)K$; When $m = 4$, the $\sum_{i=1}^m \|\phi(\mathbf{x}_i)\|_0 \leq (1 + 2 + 3 + 3)K$, where “=” is achieved when the new instance will increase the tree depth by 1 when it is added to the dataset. Therefore, $\sum_{i=1}^m \|\phi(\mathbf{x}_i)\|_0 \leq \frac{(m+2)(m-1)K}{2}$ is valid, when the size of dataset $m \leq 4$. If a new instance is added to the dataset with size m (i.e., it becomes $m + 1$), the maximum number of edges that the new instance can travel is m . Therefore,

$$\begin{aligned} \|\phi(\mathbf{x}_i)\|_0 &\leq \frac{(m+2)(m-1)K}{2} + (1+m)K \\ &= \frac{(m+3)mK}{2}, \end{aligned}$$

where the second term on the right-hand side of the first inequality comes from the effect of adding one instance. By induction, $\sum_{i=1}^m \|\phi(\mathbf{x}_i)\|_0 \leq \frac{(m+2)(m-1)K}{2}$ is valid for all $m \geq 2$. Thus,

$$\begin{aligned} r &\leq \frac{(m+2)(m-1)K}{2mD} \\ &= \frac{m+2}{4m}. \end{aligned}$$

The lower bound in inequality (4) is also proved by induction. $\sum_{i=1}^m \|\phi(\mathbf{x}_i)\|_0 \geq \lfloor \log m \rfloor mK$ is valid when $m \leq 3$. If a new instance is added the dataset with size m where $2^{\lfloor \log m \rfloor} \leq m \leq 2^{\lfloor \log m \rfloor + 1} - 1$, the minimum number of edges that the new instance can travel is $\lfloor \log m \rfloor$ because all instances at least travel $\lfloor \log m \rfloor$ edges (this is the case where the lower bound is tight when the dataset size is $2^{\lfloor \log m \rfloor}$). Therefore,

$$\begin{aligned} \sum_{i=1}^{m+1} \|\phi(\mathbf{x}_i)\|_0 &\geq \lfloor \log m \rfloor mK + \lfloor \log m \rfloor \\ &= \lfloor \log(m+1) \rfloor (m+1)K \end{aligned}$$

By induction, $\sum_{i=1}^m \|\phi(\mathbf{x}_i)\|_0 \geq \lfloor \log m \rfloor mK$ is valid for all $m \geq 2$. As a result,

$$\begin{aligned} r &\geq \frac{\lfloor \log m \rfloor mK}{mD} \\ &= \frac{\lfloor \log m \rfloor}{2(m-1)}. \end{aligned}$$

□

Note that the empty leaf only shows up when there are duplicated instances in dataset $\mathbf{X}$ since these instances can not be split in a node. However, the duplicated instances can be removed so that the assumption of no empty leaf is satisfied. The above lemma discusses the sparsity of feature function $\phi(\cdot)$ under two independent conditions. Conditions (i) and (ii) correspond to the cases of small and large tree height limits, respectively. Under Condition (i), inequality (3) shows the sparsity of $\phi(\mathbf{x})$ from any instance is upper bounded by $H \cdot K$. Under Condition (ii), all instances are isolated in a leaf. The inequality (4) shows the lower and upper bound of *the ratio of nonzero elements*, namely, r , that is calculated from the extracted feature set $\{\phi(\mathbf{x}_1), \dots, \phi(\mathbf{x}_m)\}$. As the dataset size m goes to infinity, the lower bound converges to zero while the upper bound converges to $1/4$. Both conditions are needed since they cover all situations for Isolation Forest.

Remark 3 (Sparsity of model weight $\mathbf{w}$). *There are two reasons to enforce the sparsity of model weight $\mathbf{w}$. First, by proving the feature function $\phi(\cdot)$ is sparse, we can conclude that the number of edges in the forest, which are important to anomaly detection, is small since the entire feature matrix $\phi(\mathbf{X})$ is sparse. The objective of enforcing the sparsity of model weight $\mathbf{w}$ is to make the anomaly detection algorithm more focused on the essential edges. For example, edge e_3 is more important for anomaly detection compared with the edge e_4 in Figure 2. Without model sparsity, each edge is treated equally, which may slow down the learning process of algorithm. Second, the dimension of model weight $\mathbf{w}$ is very high. For example, let number of training instances $\psi = 2^8$ and number of trees $K = 100$, the dimension of $\mathbf{w}$ (i.e., D) is 51,000 if H is large based on Lemma 1. Given that we only have a small budget to query instances for labeling, enforcing the sparsity of model weight $\mathbf{w}$ makes learning more tractable by reducing the model complexity.*

4.2 Sparse Approximated Linear Anomaly Discovery

In this subsection, Step 2 in Figure 3 is presented within the framework of online optimization Shalev-Shwartz et al. (2011a) with a new proposed algorithm based on online gradient descent, which can tradeoff between the model sparsity and model accuracy.

4.2.1 Connection with Online Optimization

In our problem, the anomaly score defined in equation (2) is assigned to an instance $\mathbf{x}$, where larger scores correspond to more anomalous since anomaly instances are shallower leaf nodes (i.e., there are fewer nonzeros elements in $\phi(\mathbf{x})$ for an anomaly instance $\mathbf{x}$). At each iteration t , an anomaly score list over the unlabeled instances can be obtained through $\mathbf{w}_t$ (Step 2(1) in Figure 3). Based on the anomaly score list, the top-ranked instance is selected for query (i.e., taking measurement/experiment to label the queried instance), which consumes time or cost. After that, measurement feedback y_t on the top-ranked instance is received (Step 2(2) in Figure 3), where $y_t = +1$ if the instance is an anomaly and $y_t = -1$ if it is nominal. To transfer the measurement feedback y_t to a function for the online optimization framework, *Linear loss* is used in this paper due to its efficiency and effectiveness Siddiqui et al. (2018). The linear loss is given by

$$f_t(\mathbf{w}_t) = -y_t \phi(\mathbf{x}) \cdot \mathbf{w}_t \quad (5)$$

which is a linear function of $\mathbf{w}_t$. Intuitively, if $y_t = +1$, then our current cost vector should suffer a smaller loss than the case when $y_t = -1$, since the ranking based on $\mathbf{w}_t$ correctly put an anomaly at the top. When anomaly or nominal feedback is received, the model weight $\mathbf{w}_{t+1}$ is updated accordingly (Step 2(3) in Figure 3).

The procedure in Step 2 of Figure 3 falls under the framework of online optimization. Online optimization is an iterative game against a potential adversary where our moves are vectors from some constrained sets $\mathcal{S}$. The game proceeds as follows: (1) The adversary selects a function $f_t : \mathcal{S} \rightarrow \mathbb{R}$; (2) Suffer a loss $f_t(\mathbf{w}_t)$; (3) Select a vector $\mathbf{w}_{t+1} \in \mathcal{S}$. In our problem, the adversary is the greedy policy to select the top-ranked instance in Step 2(1). Typically, the performance of online optimization is measured by the accumulated Regret for the best $\mathbf{w}^*$ that is the minimizer by assuming f_t 's are known in advance. Specifically,

the accumulated Regret for a T -round game is defined as

$$\text{Regret}_T = \sum_{t=1}^T f_t(\mathbf{w}_t) - \min_{\mathbf{w} \in \mathcal{S}} \sum_{t=1}^T f_t(\mathbf{w}). \quad (6)$$

The objective of our problem is to minimize the accumulated Regret by choosing the appropriate $\mathbf{w}_t (t = 1, \dots, T)$ sequentially.

4.2.2 Sparse Approximated Linear Anomaly Discovery Algorithm

To minimize equation (6), a wide variety of online optimization algorithms are available to use in the literature. In particular, online gradient descent techniques [Kivinen et al. \(2004\)](#); [Zhang \(2004\)](#); [Shalev-Shwartz et al. \(2011b\)](#) stand out as an attractive combination of predictive accuracy and computational efficiency. Specifically, the iterative procedure to minimize equation (6), where model weight (i.e., decision variable) $\mathbf{w}_t$ is sequentially obtained from $t = 1$ to T , can be summarized as follows

$$\mathbf{w}_1 \rightarrow f_1(\mathbf{w}_1) \mapsto \mathbf{w}_2 \rightarrow f_2(\mathbf{w}_2) \mapsto \dots \mapsto \mathbf{w}_T \rightarrow f_T(\mathbf{w}_T).$$

At each iteration t , $\mathbf{w}_t$ is determined from the last iteration (represented by $f_{t-1}(\mathbf{w}_{t-1}) \mapsto \mathbf{w}_t$), where f_t is unseen yet. Once f_t is obtained (represented by $\mathbf{w}_t \rightarrow f_t(\mathbf{w}_t)$), it can be used to update new model weight $\mathbf{w}_{t+1}$ (represented by $f_t(\mathbf{w}_t) \mapsto \mathbf{w}_{t+1}$). The proposed algorithm, namely, SALAD, is motivated by [Zhai et al. \(2018\)](#) and presented in Algorithm 2.

As discussed in Section 4.1, it is necessary to add sparsity constraint in model weight $\mathbf{w}$. In the literature, there are three main techniques to achieve sparsity: (i) imposing an ℓ_1 -norm constraint [Duchi et al. \(2008\)](#), (ii) adding an ℓ_1 -norm regularization to the loss function [Langford et al. \(2009\)](#), or (iii) using ℓ_0 -based truncation [Wang et al. \(2013\)](#). Our proposed sparsity strategy differs from the above-mentioned approaches. Specifically, the following sparse approximated problem is proposed to achieve sparsity of $\mathbf{w}$ while minimizing equation (6)

$$\begin{aligned} \min_{\mathbf{w}} \quad & \|\mathbf{w}\|_0 \\ \text{s.t.} \quad & \|\mathbf{w} - \boldsymbol{\theta}_t\| \leq \varepsilon_t, \|\mathbf{w}\|_0 \geq \kappa. \end{aligned} \quad (7)$$

It aims to find the most sparse vector $\mathbf{w}_t$, which is constrained to be close to $\boldsymbol{\theta}_t$ and has a reasonable number of nonzero features specified as κ . The truncation error ε_t in the

constraints controls how close between $\mathbf{w}_t$ and $\boldsymbol{\theta}_t$, thus affects the sparsity of $\mathbf{w}_t$. The parameter κ in the constraint of optimization problem (7) is an integer, which helps avoid any improper situations where the value of ε_t is set too large. Problem (7) can be efficiently solved to optimal using a simple greedy approach. Specifically, it follows: first, let $\mathbf{w}_t$ be a copy of $\boldsymbol{\theta}_t$; then repeat setting the nonzero element with the smallest absolute value in $\mathbf{w}_t$ to zero until no more elements can be set to zero without violating the constraints $\|\mathbf{w} - \boldsymbol{\theta}_t\| \leq \varepsilon_t$ and $\|\mathbf{w}\|_0 \geq \kappa$.

Algorithm 2: Sparse Approximated Linear Anomaly Discovery

Input: $\mathbf{X}$; feature function: $\phi(\cdot)$; learning rate: η_t ; weight prior: $\mathbf{w}_0 = \mathbf{1}$; weight approximation error: $\varepsilon_t \geq 0$; sparsity: κ ; bounded area: $R > 0$

Output: Selected instances and their labels obtained from the corresponding measurements $\{\mathbf{x}_t, y_t\}_{t=1}^T$; sequence of model weights $\{\mathbf{w}_t\}_{t=1}^T$

```

1  $\boldsymbol{\theta}_1 = \mathbf{w}_0$ 
2 for  $t=1, 2, \dots, T$  do
3   Update  $\mathbf{w}_t$  by solving the problem in equation (7)
4    $\mathbf{w}_t = \frac{\mathbf{w}_t \cdot R}{\max\{R, \|\mathbf{w}_t\|\}}$  // Projection to a bounded  $\mathcal{S}$ 
5    $\mathbf{x}_t = \arg \max_{\mathbf{x} \in \mathcal{U}} -\phi(\mathbf{x}) \cdot \mathbf{w}_t$  // Select the instance with the highest anomaly score
6   Get measurement feedback  $y_t \in \{-1, +1\}$  on  $\mathbf{x}_t$  // Perform measurement for the
   instance with highest anomaly score to get the label information.
7    $\mathcal{U} = \mathcal{U} - \{\mathbf{x}_t\}$  // Remove the new labeled instance from the unlabeled dataset
8    $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \partial f_t(\mathbf{w}_t)$  based on  $f_t$  defined in equation (5) // Gradient Descent
```

Line 4 is to project $\mathbf{w}_t$ to a bounded area $\mathcal{S} = \{\mathbf{w} \in \mathbb{R}^D : \|\mathbf{w}\| \leq R\}$ so that the linear loss is also bounded; otherwise, $\min_{\mathbf{w} \in \mathcal{S}} \sum_{t=1}^T f_t(\mathbf{w})$ could be infinity. In Line 5, based on the current model weight $\mathbf{w}_t$, the algorithm selects one top-ranked instance $\mathbf{x}_t$ from the unlabeled dataset, followed by its costly measurement to get the label information y_t (Line 6). After selecting $\mathbf{x}_t$ and receiving y_t , loss function $f_t(\mathbf{w}_t)$, which is a part of the objective function in equation (6), can be obtained through equation (5). Based on loss function $f_t(\mathbf{w}_t)$, the $\boldsymbol{\theta}_{t+1}$ is updated in Line 8 through the standard gradient descent update. After T iterations, $\sum_{t=1}^T f_t(\mathbf{w}_t)$ in equation (6) is achieved.

Next, the Regret analysis is discussed, where the Regret is defined in equation (6) with

$\mathcal{S} = \{\mathbf{w} \in \mathbb{R}^D : \|\mathbf{w}\| \leq R\}$. The following theorem shows that the Regret of SALAD algorithm 2.

Theorem 1. *SALAD algorithm 2 with learning rate $\{\eta_t = \frac{\sqrt{2}R}{\sqrt{\sum_{s=1}^t \partial f_t^2(\mathbf{w}_t)}}, t \in 1, \dots, T\}$ and approximation error $\{\varepsilon_t, t \in 1, \dots, T\}$ guarantees the following for all $T \geq 1$*

$$\text{Regret}_T \leq \underbrace{2\sqrt{2}R \sqrt{\sum_{t=1}^T \|\phi(\mathbf{x}_t)\|_0}}_{\text{Original Convex Regret}} + \underbrace{\sum_{t=1}^T \varepsilon_t \sqrt{\|\phi(\mathbf{x}_t)\|_0}}_{\text{Sparse Approximation Regret}}. \quad (8)$$

Proof.

$$\begin{aligned} \text{Regret}_T &= \sum_{t=1}^T f_t(\mathbf{w}_t) - \min_{\mathbf{w} \in \mathcal{S}} \sum_{t=1}^T f_t(\mathbf{w}) \\ &= \sum_{t=1}^T f_t(\mathbf{w}_t) - f_t(\boldsymbol{\theta}_t) + \sum_{t=1}^T f_t(\boldsymbol{\theta}_t) - \min_{\mathbf{w} \in \mathcal{S}} \sum_{t=1}^T f_t(\mathbf{w}) \\ &= \sum_{t=1}^T y_t \phi(\mathbf{x}_t) \cdot (\boldsymbol{\theta}_t - \mathbf{w}_t) + \sum_{t=1}^T f_t(\boldsymbol{\theta}_t) - \min_{\mathbf{w} \in \mathcal{S}} \sum_{t=1}^T f_t(\mathbf{w}) \\ &\leq \sum_{t=1}^T \|\phi(\mathbf{x}_t)\| \cdot \|\boldsymbol{\theta}_t - \mathbf{w}_t\| + 2\sqrt{2}R \sqrt{\sum_{t=1}^T \|\phi(\mathbf{x}_t)\|^2} \\ &\leq \sum_{t=1}^T \|\phi(\mathbf{x}_t)\| \cdot \varepsilon_t + 2\sqrt{2}R \sqrt{\sum_{t=1}^T \|\phi(\mathbf{x}_t)\|^2} \\ &\leq \sum_{t=1}^T \sqrt{\|\phi(\mathbf{x}_t)\|_0} \cdot \varepsilon_t + 2\sqrt{2}R \sqrt{\sum_{t=1}^T \|\phi(\mathbf{x}_t)\|_0}. \end{aligned}$$

In the first inequality, $\sum_{t=1}^T y_t \phi(\mathbf{x}_t) \cdot (\boldsymbol{\theta}_t - \mathbf{w}_t) \leq \|\phi(\mathbf{x}_t)\| \cdot \|\boldsymbol{\theta}_t - \mathbf{w}_t\|$ is due to the Cauchy inequality and $y_t = \{+1, -1\}$. $\sum_{t=1}^T f_t(\boldsymbol{\theta}_t) - \min_{\mathbf{w} \in \mathcal{S}} \sum_{t=1}^T f_t(\mathbf{w}) \leq 2\sqrt{2}R \sqrt{\sum_{t=1}^T \|\phi(\mathbf{x}_t)\|^2}$ is based on the result in Streeter and McMahan (2010); McMahan (2012) and the fact that $\partial f_t(\mathbf{w}_t) = \partial f_t(\boldsymbol{\theta}_t) = -y_t \phi(\mathbf{x}_t)$. The second inequality comes from the constraint in problem (7). The third inequality is due to the fact that $\phi(\mathbf{x}_t)$ is a binary vector. $\square$

The Regret of our proposed SALAD algorithm shown in the right hand side of equation (8) can be decomposed into two parts. The first part is the original convex Regret $\mathcal{O}(\sqrt{T})$, which is the Regret without involving the sparse approximation problem in (7). The original convex Regret has a desirable asymptotic property, namely, *no-regret* (also called *sub-linear* convergence rate). That is, $\lim_{T \rightarrow +\infty} \frac{R(T)}{T} = 0$. The second part is the sparse

approximation Regret, which comes from the sparse approximation problem in (7) in Line 3. If $\varepsilon_t = \mathcal{O}(1/\sqrt{T})$, the sparse approximation Regret also has a *sub-linear* convergence rate since $\sum_{t=1}^T \varepsilon_t \sqrt{\|\phi(\mathbf{x}_t)\|_0}$ is $\mathcal{O}(\sqrt{T})$. However, when ε_t is set too small, the model weight $\mathbf{w}$ is not sparse enough to promote the sparsity in $\phi(\mathbf{x}_t)$. When ε_t is set too big, the model weight $\mathbf{w}$ is too sparse to generalize well on unseen instances.

Remark 4. *The proposed algorithm 2 leads to regret bounds that are stronger than those of standard online gradient descent Shalev-Shwartz et al. (2011a) for general online convex optimization problems, which rely on the Lipschitz continuity of feature function $\phi(\cdot)$. The standard online gradient descent will yield the regret bound $\sum_{t=1}^T G \cdot \varepsilon_t + 2\sqrt{2}RG\sqrt{T}$, where $G = \max_{\mathbf{x}} \|\phi(\mathbf{x})\|_2$. It shows that the Regret of the proposed algorithm 2 is tighter than those standard online gradient descent. Especially, our regret bound is quantified by the cardinality of $\phi(\cdot)$, which is proven to be sparse in Lemma 1.*

Parameter setting of Algorithm 1 and 2 for case studies in Section 5: In Algorithm 1, sub-sample size ψ , number of trees K , and tree height limit H need to be determined. As suggested in the literature Liu et al. (2008, 2012); Das et al. (2017), $\psi = 2^8$, $K = 100$. H is set to a large number so that all instances are isolated in a leaf Siddiqui et al. (2018). In our proposed SALAD Algorithm 2, the learning rate η_t is set to the value 1 since it worked well throughout our range of experiments. Based on our empirical results, $\kappa = 4,000$ and $\varepsilon_t = \sqrt{0.8} \cdot \|\boldsymbol{\theta}_t\|$ are found to be very effective in our case studies. R can be set up as any positive value since our anomaly score in equation (2) is a linear function, which is not affected by any scale operation.

5 Case Studies

To evaluate the performance of the proposed SALAD, numerical and real-world case studies are conducted in this section. In Section 5.1, the sparsity of feature function $\phi(\cdot)$ and the convergence of SALAD algorithm 2 are investigated with numerically simulated data. In Sections 5.2, 5.3 and 5.4, real-world applications in open-source datasets, a polymer AM dataset, and a metal AM dataset are used to demonstrate the performance of the proposed SALAD. For all analyses in Sections 5.2, 5.3 and 5.4, our proposed SALAD is assessed against the following benchmark methods:

- (1) Unsupervised baseline: All instances are sorted in decreasing order of anomaly score in equation (2) computed with the Isolation Forest algorithm given $\mathbf{w} = \mathbf{1}$. This algorithm ignores the measurement feedback, and thus the ranking is constant across iterations. This baseline captures the performance of an unsupervised anomaly detection algorithm that does not incorporate measurement feedback.
- (2) GLAD: This corresponds to the motivating paper Siddiqui et al. (2018), where the anomaly score is computed using equation (2) based on Isolation Forest. The algorithm incorporates the measurement feedback by adjusting the model weight $\mathbf{w}$ via online mirror descent with non-negative constraints. The instance with the highest anomaly score based on the current model weight is selected to evaluate and get the label at each iteration.

For the evaluation metric, anomaly discovery curve Ding et al. (2019) is utilized to plot the number of discovered anomalies against the number of iterations (measurements). A perfect result is a line with a slope of one, i.e., all the queries are anomalies. The worst case is a line with a slope of zero, i.e., all the queries are normal. All analyses in Sections 5.2, 5.3 and 5.4 are repeated 10 times to get the mean and standard error for comparison. The codes of SALAD are implemented in Matlab 2019a. The CPU of the computer used in the experiments is an Intel[®] Core[™] Processor i7-6820HQ (Quad Core 2.70 GHz, 3.60 GHz Turbo, 8MB 45W).

5.1 Numerical Study

In this subsection, two cases are considered using simulated data: (a) sparsity analysis of feature function $\phi(\cdot)$ with different sizes of training data (i.e., m), and (b) a numerical optimization performance of Algorithm 2 with different ε_t .

Case (a): The instances used here are simulated independently from $\mathcal{N}(0, 5)$. To study the sparsity of feature function $\phi(\cdot)$ as stated in Lemma 1, different numbers of instances are considered for both Conditions (i) and (ii). Specifically, the number of instances can be set to value $m = 2^j$, where $j = \{4, 5, \dots, 12\}$. For Condition (i), the tree limit height H is set to the value $\lceil \log m \rceil$ as suggested in Liu et al. (2008, 2012). For Condition (ii), the tree limit height H is set to a large number so that all m instances are isolated in a leaf. In

Figure 4: Sparsity analysis of $\phi(\cdot)$ under Conditions (i) and (ii) in Lemma 1.

Figures 4a and 4b, the ratio of nonzero elements is plotted versus the number of instances. The ratio keeps decreasing to zero when the number of instances increases. Furthermore, the lower and upper bounds in equation (4) are also shown in Figure 4b. It shows that the ratio of nonzero elements is much closer to the lower bound, especially when the number of instances is large. As a result, Figure 4 empirically illustrates that the feature function $\phi(\cdot)$ is sparse for both Conditions (i) and (ii) in Lemma 1.

Case (b): The nominal instances are simulated from $\mathcal{N}(0, 5)$ independently, while anomaly instances are simulated from $\mathcal{N}(3, 3)$ independently. In total, there are 400 nominal instances and 100 anomaly instances considered in this experiment. To study the effect of ε_t in equation (7), ε_t can take value from $\{0, 0.2 \cdot \|\theta_t\|, 0.4 \cdot \|\theta_t\|, 0.6 \cdot \|\theta_t\|, 0.8 \cdot \|\theta_t\|\}$. If $\varepsilon_t = 0$, our algorithm degrades to the online gradient descent algorithm used by Streeter and McMahan (2010), which has the *sub-linear* convergence rate. It can be treated as the oracle for comparison since the optimal solution of the second term on the right-hand side of equation (6) is unknown. In this experiment, Line 5 in Algorithm 2 is replaced by a random policy so that the same instance is used for different ε_t at each iteration. In addition, the sparsity κ is set to 4,000.

For performance evaluation, the average loss Zhai et al. (2018) defined as $\frac{1}{t} \sum_{i=1}^t f_i(\mathbf{w}_i)$ is plotted versus the number of iterations in Figure 5a, where $f_i(\mathbf{w}_i)$ is defined in equation (5). The five curves corresponding to five ε have a very similar trend. It can be observed

Figure 5: Convergence performance of SALAD algorithm (a) Average loss versus number of iterations; (b) sparsity of model weight w_t versus number of iterations.

that a larger ε_t incurs a larger average loss. It is reasonable since a larger approximation error results in a larger loss. On the other hand, the sparsity of model weight w_t is plotted versus the number of iterations for different ε_t in Figure 5b. For all ε_t except $\varepsilon_t = 0$, the sparsity keeps decreasing until it reaches $\kappa = 4,000$ when the number of iterations increases. Besides, a larger ε_t corresponds to a smaller value of sparsity.

5.2 Application in Open-source Datasets

In this subsection, SALAD and benchmark methods are applied to open-source datasets for performance comparison. Twenty datasets selected from Campos et al. (2016); Ranjan et al. (2018) and ODDS³ are used in this experiment. The classes were divided into two sets for each dataset, one representing the nominal instances and the other smaller set representing the anomaly instances. The dimension, the number of anomalies, and the percentage of anomalies in each dataset are summarized in Table 1. For each dataset, the budget T is determined by its number of anomalies. A larger number of anomalies corresponds to a larger budget. Table 2 shows the quantitative results for all the datasets, including the mean and standard error of the number of detected anomalies. Overall, GLAD and SALAD have much better performance than the unsupervised baseline method, demonstrating the

³<http://odds.cs.stonybrook.edu/>

Table 1: Statistics of Open-source datasets.

	# Instances	# Dim.	# Anomalies	Anomalies %
Aloi	50,000	27	1,508	3.02
Annth thyroid	7,200	21	347	4.82
Cardioto	2,126	21	86	4.05
Heart	270	13	7	2.59
Kdd	60,632	41	200	0.33
Letter	1,600	32	100	6.25
Mammography	11,183	6	260	2.32
Mnist	7,603	100	700	9.21
Optdigits	5,216	64	150	2.88
Pblock	5,473	10	99	1.81
Process-Miner	18,398	61	124	0.67
Satellite	6,435	36	2,036	31.64
Shuttle	1,013	9	13	1.28
Spambase	4,601	57	280	6.09
Speech	3,686	400	61	1.65
Thyroid	3,772	6	93	2.47
Vowels	1,456	12	50	3.43
Waveform	3,443	21	100	2.90
Wilt	4,839	5	261	5.39
Yeast	1,364	8	64	4.69

advantage of incorporating measurement feedback into the unsupervised analysis. The proposed SALAD can achieve the best mean performance for all twenty datasets, where it has different levels of improvement over GLAD. On average, the number of anomalies detected by SALAD is 1.97 times more than that detected by GLAD. This illustrates that sparsity can improve the ability to detect anomalies. In terms of standard error, even though the proposed SALAD is not the best, it can generally achieve relatively low standard error.

The results in Table 2 only present the performance of different methods when the budget is reached. To explore the performance of different methods during feedback loops,

Table 2: Results of detected anomalies across 10 runs on open-source datasets for different algorithms.

	Unsupervised		GLAD		Proposed	
	Mean	SE	Mean	SE	Mean	SE
Aloi	27.60	3.21	57.80	6.76	122.40	7.50
Annth thyroid	71.00	9.20	300.30	13.94	333.80	11.57
Cardiotto	38.10	3.51	59.80	5.14	72.10	7.65
Heart	6.00	0.00	6.00	0.00	6.80	0.42
Kdd	102.63	3.38	137.00	5.93	145.13	5.51
Letter	32.70	3.50	31.80	4.24	72.30	5.46
Mammography	70.90	8.44	179.80	3.77	194.50	4.72
Mnist	224.60	12.56	601.40	5.83	635.60	14.86
Optdigits	5.70	1.64	143.50	13.57	149.90	0.32
Pblock	42.00	2.05	70.80	1.55	75.10	2.13
Process-Miner	10.90	1.79	21.10	4.65	56.90	8.41
Satellite	1191.70	23.82	1431.20	7.55	1708.80	17.18
Shuttle	1.50	0.53	10.00	3.46	12.80	0.63
Spambase	119.10	6.74	217.80	3.08	247.10	3.51
Speech	4.10	1.45	4.00	1.05	5.40	1.78
Thyroid	66.20	3.39	90.00	0.00	92.90	0.32
Vowels	14.30	2.00	39.00	1.49	48.60	1.07
Waveform	25.10	3.51	19.00	4.69	73.20	2.49
Wilt	4.00	1.05	5.30	1.06	166.50	55.57
Yeast	29.20	1.48	32.30	0.82	34.10	0.88

Figure 6 plots the number of detected anomalies versus the number of iterations for all twenty datasets. The curves are averaged over 10 independent runs of the algorithm, and 95% confidence intervals are also shown. Overall, it illustrates that SALAD never deteriorates the performance of unsupervised baseline (Isolation Forest). In most cases, SALAD significantly increases the number of anomalies discovered over time compared to both Isolation Forest and GLAD.

Figure 6: The total number of detected anomalies versus the number of iterations for all open-source datasets, where the error-bars represent 95% Confidence Interval.

5.3 Application in Polymer Additive Manufacturing

In this section, a polymer AM dataset [Shen et al. \(2020\)](#) is used to explore the performance of the proposed SALAD on detecting anomalies of interest from an unlabeled dataset. In this experiment, a desktop FFF 3D printer, namely, a Hyrel System 30M 3D printer, is used to print a test artifact, a cuboid of dimensions of 2 inches. In reality, the FFF printing process may encounter defects such as voids, overfill, underfill, etc. However, the root causes of

Figure 7: (a) Normal; (b) Underfill caused by improper setting of a cooling fan; (c) Underfill caused by sub-optimal feed rate (red boxes are ROI) [Shen et al. \(2020\)](#).

similar defects may be different, leading to the difficulty of defect mitigation. For example, both inappropriate cooling fan settings for temperature control and sub-optimal feed rate (material extrusion speed) may cause underfill, but with different severities. Differentiating these similar defects is critical. Therefore, sensors are installed for online monitoring of the process of printing. High-quality images of the part surface being printed are collected by two digital microscopes at a sampling frequency of 1 Hz, which are installed at the two sides of the extruder of a 3D printer. Images for the surface quality of normal, underfill caused by improper setting of a cooling fan, and underfill caused by sub-optimal feed rate (shown in Figure 7) are collected during the printing, respectively. For each image, a region of interest (ROI) is utilized, which is cropped from the original image (640×480 pixels) to a smaller region (80×80 pixels) right below the nozzle as shown in Figure 7. The image in ROI is transformed into a vector with 6,400 variables for testing different algorithms. For each variable, it represents a pixel value ranging from 0 to 255. There are 205 images of normal, 153 images of underfill caused by feed rate, and 42 images of underfill caused by the fan, respectively.

The experiment in this subsection has the following procedure.

- **Step 1 (Relabeling):** 42 images of underfill caused by the fan are labeled as $y = +1$ (defects of interest), while 205 images of normal and 153 images of underfill caused by feed rate are labeled as $y = -1$ (not defects of interest).
- **Step 2 (Implementation):** All these 400 images in **Step 1** are treated as unlabeled images in this analysis. Their actual labels (i.e., y) are the ground truth that needs to be measured in Line 6 of our proposed algorithm 2. The objective of our algorithm is to detect images of underfill caused by the fan as many as possible under a given budget.

The budget T is set to 100 (the number of measurements). Figure 8 plots the number

Figure 8: The total number of detected anomalies versus the number of iterations on polymer AM dataset (The error-bars represent 95% Confidence Interval).

of detected anomalies versus the number of measurements (iterations). The curves are averaged over 10 independent runs of the algorithm, and 95% confidence intervals are also shown. The result in Figure 8 implies that the proposed SALAD performs the best since it identifies much more anomalies than the benchmark methods do under the same number of measurements. GLAD improves slightly from the unsupervised baseline. Again, this shows that the advantage of involving sparsity in our proposed algorithm.

In order to gain a deeper understanding of SALAD, the visualization of the polymer AM dataset is provided along with the queries by three algorithms. Our intent is to provide some intuitions on how AAD works. For this purpose, t-SNE [Van der Maaten and Hinton \(2008\)](#),

a popular dimensionality reduction technique, is applied to produce a two-dimensional (2D) representation of the dataset in this experiment. Figure 9 shows a 2D representation of the

Figure 9: Low-dimensional visualization of polymer AM dataset using t-SNE. Plus signs are anomalies, and circles are nominals. A **red** coloring indicates that a true anomaly point was queried. A **green** indicates a nominal point was queried. **Gray** circles correspond to unqueried nominals. To make unqueried anomalies stand out visually, they are indicated with **blue** plus signs.

polymer AM produced by t-SNE, along with the queries made by benchmarks and SALAD. The unsupervised baseline and GLAD tend to discover nominals at the same location shown in the right bottom corner of Figures 9a and 9b. In contrast, our proposed SALAD not only detects more anomalies from the same location but also queries anomalies at more locations as SALAD can detect anomalies shown in the left bottom corner of Figure 9c. Overall, SALAD can query instances more likely to be anomalies and focus on various regions.

5.4 Application in Metal Additive Manufacturing

This subsection uses a metal AM dataset from the EBM process [Townsend et al. \(2016\)](#) to evaluate SALAD on the measurement allocation problem. In the printing process, the machine ARCAM Q10 plus is applied to print samples with dimensions 15 mm×15 mm×25 mm using Ti-6Al-4V powder. The image shown in Figure 10a is the 2D surface pattern of the printed part. Surface morphology measurement and quantification can be obtained through a white light interferometer [Wang et al. \(2021\)](#), which has a small field of view

Figure 10: Metal AM dataset: (a) 2D scan image (yellow square boxes are selected random samples); (b) the corresponding point cloud data.

(spatial resolution $\sim 20 \mu\text{m}$) and takes hours to scan a small surface. The printed part is relatively large ($15 \text{ mm} \times 15 \text{ mm}$ top surface). Therefore, we would like to sample some locations to measure the surface morphology since the measurements are costly, and the cost (the number of measurements) needs to be controlled under the budget. Our proposed algorithm is applied in order to select the locations for S_a (arithmetical mean height of an area) [Thomas \(1981\)](#) measurement in order to identify the surface area with inferior surface finishing. To obtain multiple training samples, 200 images with a size of 60 by 60 are randomly selected without overlap from the image in [Figure 10a](#). Each image is transformed into a vector with 3,600 variables, where each variable represents a pixel value ranging from 0 to 255.

The experiment in this subsection has the following procedure.

- **Step 1 (Labeling):** For 200 images from [Figure 10a](#), the corresponding surface roughness S_a is calculated from the point cloud data in the same area, as shown in [Figure 10b](#). The top-7 images that have the highest S_a are labeled as $y = +1$ (anomalies). The remaining images are nominal instances labeled as $y = -1$.
- **Step 2 (Implementation):** For these 200 images in **Step 1**, their actual labels (i.e., y) are the ground truth that needs to be measured in Line 6 of our proposed algorithm [2](#). The objective of our algorithm is to detect the images that are labeled

as $y = +1$ as quick as possible.

The budget T is set to 45 (the number of measurements). Figure 11a plots the number

Figure 11: (a) The total number of detected anomalies versus the number of iterations on metal AM dataset (The error-bars represent 95% Confidence Interval). (b) Locations of Sa measurements. A **red** box indicates that a true anomaly point was queried. A **green** box indicates a nominal point was queried.

of detected anomalies versus the number of measurements (iterations). The curves are averaged over 10 independent runs of the algorithm, and 95% confidence intervals are also shown. The result in Figure 11a implies that the proposed SALAD performs better than the unsupervised baseline and GLAD. Specifically, the proposed SALAD detects all seven anomalies within 40 measurements. Figure 11b plots the locations of Sa measurements of our algorithm in the 2D surface view. All anomalies are around the letter “R” in the middle area, which are correctly identified by our algorithm. The other locations of measurements have several clusters, which shows that our algorithm makes accurate decisions on which location to measure with a maximum probability to capture the inferior surface finishing.

Same as Section 5.3, the visualization of the metal AM dataset is provided along with the queries by three algorithms. Figure 12 shows a 2D representation of the metal AM produced by t-SNE, along with the queries made by benchmarks and SALAD. The proposed SALAD can detect all seven anomalies in the left bottom corner, as shown in Figure 12c, while

Figure 12: Low-dimensional visualization of metal AM dataset using t-SNE. Plus signs are anomalies, and circles are nominals. A **red** coloring indicates that a true anomaly point was queried. A **green** indicates a nominal point was queried. **Gray** circles correspond to unqueried nominals. To make unqueried anomalies stand out visually, they are indicated with **blue** plus signs.

the unsupervised baseline and GLAD miss two and one anomaly, respectively. Besides, Figure 12c illustrates that SALAD queries more instances around the anomalous area, demonstrating its effectiveness in anomaly discovery.

6 Conclusion

In this article, a new Sparse Approximated Linear Anomaly Discovery algorithm is developed to include sparsity constraint in active anomaly discovery based on isolation forest. The sparsity of feature function from Isolation Forest is studied theoretically and verified empirically to show our motivation to include sparsity. The effectiveness of the proposed algorithm is validated using numerical and real-world case studies. In the numerical study, the empirical convergence shows the tradeoff behavior of the proposed SALAD in terms of regret and model sparsity. Specifically, the proposed SALAD can maintain the model accuracy and sparsity simultaneously. Twenty open-source datasets and two additive manufacturing datasets in real-world applications are applied to evaluate our proposed algorithm. Based on these results, it is evident that SALAD outperforms state-of-the-art algorithms in the literature by including sparsity in the model, especially for datasets with a large number of instances.

In addition, there are still some aspects of SALAD that deserve further investigation. First, unsupervised feature extraction for image data should be further investigated before applying our proposed method. Second, the query strategy in Line 5 of Algorithm 2 to select the top-1 instance is a greedy approach. The query strategy that can tradeoff exploration and exploitation is one of the next steps of research.

References

- An, J. and Cho, S. (2015). Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE*, 2(1):1–18.
- Breunig, M. M., Kriegel, H.-P., Ng, R. T., and Sander, J. (2000). Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 93–104.
- Campos, G. O., Zimek, A., Sander, J., Campello, R. J., Micenková, B., Schubert, E., Assent, I., and Houle, M. E. (2016). On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study. *Data mining and knowledge discovery*, 30(4):891–927.
- Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58.
- Das, S., Islam, M. R., Jayakodi, N. K., and Doppa, J. R. (2019). Active anomaly detection via ensembles: Insights, algorithms, and interpretability. *arXiv preprint arXiv:1901.08930*, 1(1):1–47.
- Das, S., Wong, W.-K., Dietterich, T., Fern, A., and Emmott, A. (2020). Discovering anomalies by incorporating feedback from an expert. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 14(4):1–32.
- Das, S., Wong, W.-K., Fern, A., Dietterich, T. G., and Siddiqui, M. A. (2017). Incorporating feedback into tree-based anomaly detection. *arXiv preprint arXiv:1708.09441*, 1(1):1–9.
- Ding, K., Li, J., and Liu, H. (2019). Interactive anomaly detection on attributed networks. In *Proceedings of the twelfth ACM international conference on web search and data mining*, pages 357–365.
- Domingues, R., Filippone, M., Michiardi, P., and Zouaoui, J. (2018). A comparative evaluation of outlier detection algorithms: Experiments and analyses. *Pattern Recognition*, 74:406–421.
- Duchi, J., Shalev-Shwartz, S., Singer, Y., and Chandra, T. (2008). Efficient projections onto the l_1 -ball for learning in high dimensions. In *Proceedings of the 25th international conference on Machine learning*, pages 272–279.
- Emmott, A. F., Das, S., Dietterich, T., Fern, A., and Wong, W.-K. (2013). Systematic construction of anomaly detection benchmarks from real data. In *Proceedings of the ACM SIGKDD workshop on outlier detection and description*, pages 16–21.

- Görnitz, N., Kloft, M., Rieck, K., and Brefeld, U. (2013). Toward supervised anomaly detection. *Journal of Artificial Intelligence Research*, 46:235–262.
- Hazan, E. (2019). Introduction to online convex optimization. *arXiv preprint arXiv:1909.05207*, 0(0):1–178.
- Kivinen, J., Smola, A. J., and Williamson, R. C. (2004). Online learning with kernels. *IEEE transactions on signal processing*, 52(8):2165–2176.
- Knorr, E. M. and Ng, R. T. (1998). Algorithms for mining distance-based outliers in large datasets. In *VLDB*, volume 98, pages 392–403. Citeseer.
- Langford, J., Li, L., and Zhang, T. (2009). Sparse online learning via truncated gradient. *Journal of Machine Learning Research*, 10(3):777–801.
- Liu, C., Law, A. C. C., Roberson, D., and Kong, Z. J. (2019). Image analysis-based closed loop quality control for additive manufacturing with fused filament fabrication. *Journal of Manufacturing Systems*, 51:75–86.
- Liu, F. T., Ting, K. M., and Zhou, Z.-H. (2008). Isolation forest. In *2008 eighth ieee international conference on data mining*, pages 413–422. IEEE.
- Liu, F. T., Ting, K. M., and Zhou, Z.-H. (2012). Isolation-based anomaly detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 6(1):1–39.
- McMahan, B. (2012). Lecture notes in online learning.
- Ranjan, C., Reddy, M., Mustonen, M., Paynabar, K., and Pourak, K. (2018). Dataset: rare event classification in multivariate time series. *arXiv preprint arXiv:1809.10717*, 0(0):1–7.
- Settles, B. (2009). Active learning literature survey. *Computer Sciences Technical Report*.
- Shalev-Shwartz, S. et al. (2011a). Online learning and online convex optimization. *Foundations and trends in Machine Learning*, 4(2):107–194.
- Shalev-Shwartz, S., Singer, Y., Srebro, N., and Cotter, A. (2011b). Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30.
- Shen, B., Kamath, R. R., and Hahn Choo, Z. J. K. (2021a). Robust tensor decomposition based background/foreground separation in noisy videos and its applications in additive manufacturing. *TechRxiv*, 1(1):1–13.
- Shen, B., Wang, R., Law, A. C. C., Kamath, R., Choo, H., and Kong, Z. (2021b). Super resolution for multi-sources image stream data using smooth and sparse tensor completion and its applications in data acquisition of additive manufacturing. *Technometrics*, 1(1):1–16.
- Shen, B., Xie, W., and Kong, Z. J. (2020). Clustered discriminant regression for high-dimensional data feature extraction and its applications in healthcare and additive manufacturing. *IEEE Transactions on Automation Science and Engineering*, 19(19):1–13.

- Siddiqui, M. A., Fern, A., Dietterich, T. G., Wright, R., Theriault, A., and Archer, D. W. (2018). Feedback-guided anomaly discovery via online optimization. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2200–2209.
- Streeter, M. and McMahan, H. B. (2010). Less regret via online conditioning. *arXiv preprint arXiv:1002.4862*, 0(0):1–11.
- Ten, C.-W., Hong, J., and Liu, C.-C. (2011). Anomaly detection for cybersecurity of the substations. *IEEE Transactions on Smart Grid*, 2(4):865–873.
- Thomas, T. (1981). Characterization of surface roughness. *Precision Engineering*, 3(2):97–104.
- Townsend, A., Senin, N., Blunt, L., Leach, R., and Taylor, J. (2016). Surface texture metrology for metal additive manufacturing: a review. *Precision Engineering*, 46:34–47.
- Ukil, A., Bandyopadhyay, S., Puri, C., and Pal, A. (2016). Iot healthcare analytics: The importance of anomaly detection. In *2016 IEEE 30th international conference on advanced information networking and applications (AINA)*, pages 994–997. IEEE.
- Van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(11):2579–2605.
- Wang, J., Zhao, P., Hoi, S. C., and Jin, R. (2013). Online feature selection and its applications. *IEEE Transactions on knowledge and data engineering*, 26(3):698–710.
- Wang, R., Law, A., Garcia, D., Yang, S., and Kong, Z. (2021). Development of structured light 3d-scanner with high spatial resolution and its applications for additive manufacturing quality assurance. *The International Journal of Advanced Manufacturing Technology*, 0(0):1–24.
- Zhai, T., Koriche, F., Wang, H., and Gao, Y. (2018). Tracking sparse linear classifiers. *IEEE transactions on neural networks and learning systems*, 30(7):2079–2092.
- Zhang, T. (2004). Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on Machine learning*, page 116.
- Zhao, Y., Nasrullah, Z., and Li, Z. (2019). Pyod: A python toolbox for scalable outlier detection. *Journal of Machine Learning Research*, 20(96):1–7.