AI in Anomaly Detection and Image Compression

A Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Bachelor of Technology

by

Kaushal Kishore
(111601008)

under the guidance of

Dr. Chandra Shekar

INDIAN INSTITUTE
OF TECHNOLOGY
PALAKKAD

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

*This is to certify that the work contained in this thesis entitled "**AI in Anomaly Detection and Image Compression**" is a bonafide work of **Kaushal Kishore** (**Roll No. 111601008**), carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Palakkad under my supervision and that it has not been submitted elsewhere for a degree.*

Dr. Chandra Shekar

Assistant Professor

Department of Computer Science & Engineering

Indian Institute of Technology Palakkad

Acknowledgements

Apart from the efforts of myself, the success of any project depends largely on the encouragement and guidelines of many others. I take this opportunity to express my gratitude to the people who have been instrumental in the successful completion of this project. I would like to show my greatest appreciation to Dr. Chandra Shekar. I can't say thank you enough for his tremendous support and help. I feel motivated and encouraged every time I attend his meeting. Without his encouragement and guidance, this project would not have materialized.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter discusses anomaly detection, its use cases and some major challenges.

1.1 Anomaly Detection

Anomaly detection (also outlier detection) is the identification of rare items, events or observations which raise suspicions by differing significantly from the majority of the data. Typically, the anomalous items will translate to some kind of problem such as bank fraud, a structural defect, medical problems or errors in a text. Anomalies also known as outliers, novelties, noise, deviations and exceptions.

Three broad categories of anomaly detection techniques exist:

1. Unsupervised anomaly detection techniques detect anomalies in an unlabeled test data set under the assumption that the majority of the instances in the data set are normal by looking for instances that seem to fit least to the remainder of the data set.

2. Supervised anomaly detection techniques require a data set that has been labeled as "normal" and "abnormal" and involves training a classifier (the key difference to

many other statistical classification problems is the inherently unbalanced nature of outlier detection).

3. Semi-supervised anomaly detection techniques construct a model representing normal behavior from a given normal training data set, and then test the likelihood of a test instance to be generated by the learnt model.

We will restrict ourselves to unsupervised anomaly detection and semi-supervised anomaly detection problem.

1.2 Use Cases

The ability to detect anomalies has significant relevance, and anomalies often provides critical and actionable information in various application domains.

Identification of potential outliers is important for the following reasons: [1]

1. An outlier may indicate bad data. For example, the data may have been coded incorrectly, or an experiment may not have been run correctly. If it can be determined that an outlying point is in fact erroneous, then the outlying value should be deleted from the analysis (or corrected if possible).

2. In some cases, it may not be possible to determine if an outlying point is bad data. Outliers may be due to random variation or may indicate something scientifically interesting. In any event, we typically do not want to simply delete the outlying observation.

For example, anomalies in credit card transactions could signify fraudulent use of credit cards. An anomalous spot in an astronomy image could indicate the discovery of a new star. An unusual computer network traffic pattern could stand for unauthorised access. These applications demand anomaly detection algorithms with high detection accuracy and fast execution.

1.3 Challenges

1.3.1 Masking and Swamping

Masking and swamping is the biggest problem affecting any anomaly detection algorithm.

Masking is the existence of too many anomalies concealing their own presence. It happens when anomaly clusters become large and dense. For example, if we are testing for a single outlier when there are in fact more outliers, these additional outliers may influence the value of the test statistic enough so that no points are declared as outliers.

On the other hand, swamping refers to situations where normal instances are wrongly identifying as anomalies. It happens when the number of normal instances increases, or they become more scattered. For example, if we are testing for two or more outliers when there is in fact only a single outlier, both points may be declared outliers.

Masking is one reason that trying to apply a single outlier test sequentially can fail. For example, if there are multiple outliers, masking may cause the outlier test for the first outlier to return a conclusion of no outliers. So the testing is not performed for any additional outliers.

1.3.2 Concept drift

In the case of streaming data, the anomaly context can change over time. For example, consider a user's behaviour change from one system to another. The anomaly detection algorithm should adapt to this change in the behaviour of the external agent. This deviation of the normal behaviour time to time is called concept drift. Any online anomaly detection algorithm must have a way to deal with this.

1.3.3 Miscellaneous

Apart from the above conceptual challenges, here are some general challenges pointed out by Vatsal et al. [2]

High dimensional, heterogeneous data: The data collected could contains measurements of metrics like cpu usage, memory, bandwidth, temperature, in addition to categorical data such as day of the week, geographic location, OS type. This makes finding an accurate generative model for the data challenging. The metrics might be captured in different units, hence algorithms that are unit-agnostic are preferable. The algorithm needs to scale to high dimensional data.

Scarce labels: Most of the data are unlabeled. Generating labels is time and effort intensive and requires domain knowledge. Hence supervised methods are a non-starter, and even tuning too many hyper-parameters of unsupervised algorithms could be challenging.

Irrelevant attributes: Often an anomaly manifests itself in a relatively small number of attributes among the large number being monitored. For instance, a single machine in a large datacenter might be compromised and behave abnormally.

1.4 Organization of The Report

This chapter [1] provides a background for the topics covered in this report. We provided a description of anomaly detection problem and discussed some use cases. Then we discussed some challenges to anomaly detection problem: masking, swamping and concept drift. In the next chapter [2] we will discuss a very efficient ensemble method Isolation Forest for anomaly detection. In chapter [3] we will discuss another ensemble method PIDForest which has been recently developed. The major drawback of the above mentioned algorithms is that they are used in offline setting without dealing with concept drift. Most of the anomaly detection algorithm is offline and fail to address the problem of concept drift. In chapter [4] we will present some methods to address these issues. In chapter [5] we will review our work did till mid-sem. And finally in chapter [6], we conclude with some future works.

Code Repository: `https://github.com/KishoreKaushal/AnomalyDetection`

Report Repository: `https://github.com/KishoreKaushal/btp-report`

Chapter 2

Isolation Forest

2.1 Isolation based anomaly detection

Isolation is the process or fact of isolating or being isolated. The authors of [3] proposed an isolation based anomaly detection which takes advantage of two quantitative properties of anomalies:

1. They are the minority consisting of few instances.
2. They have attribute-values that are very different from those of normal instances.

Hence, anomalies are 'few and different' which make them more susceptible to a mechanism we called Isolation. Isolation can be implemented by any means that separates instances. Lui et al. [3] proposed to use a binary tree structure called isolation tree (iTree) which can be constructed effectively to isolate instances. Because of the susceptibility to isolation, anomalies are more likely to be isolated closer to the root of an iTree; whereas normal points are more likely to be isolated at the deeper end of an iTree.

The proposed method, called Isolation Forest (iForest) builds an ensemble builds an ensemble of iTrees for a given data set. Anomalies are those instances which have short average path lengths on the iTrees. There are two training parameters and one evaluation parameter in this method: the training parameters are the number of trees to build and subsampling size. The evaluation parameter is the tree height limit during evaluation.

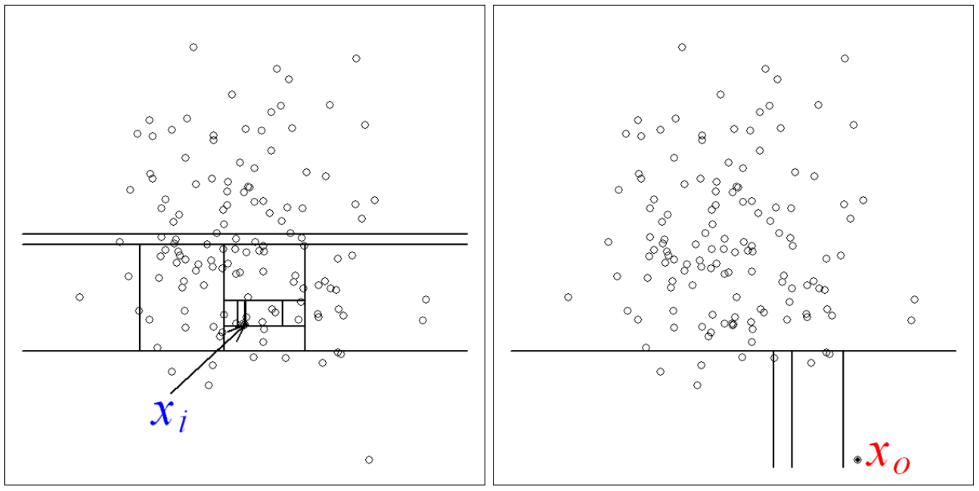


Fig. 2.1: **Left**: a normal point x_i requires twelve random partitions to be isolated; **Right**: an anomaly x_o requires only four partitions to be isolated.
Source: Lui et al. [3]

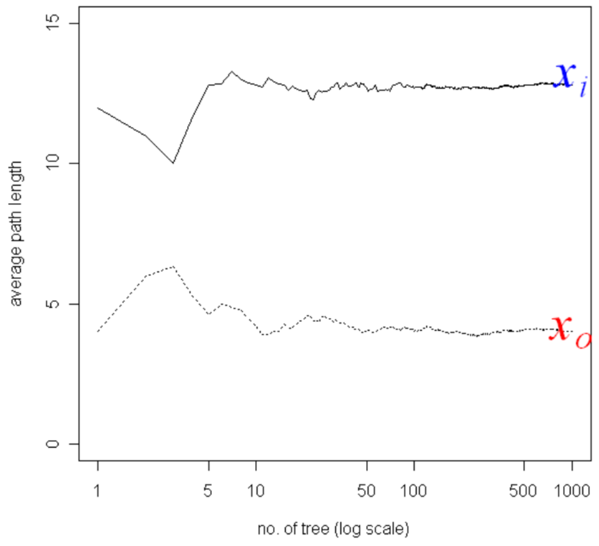


Fig. 2.2: Averaged path lengths of x_i and x_o converge when the number of trees increases.
Source: Lui et al. [3]

2.2 Training Stage

Formally, isolation tree is defined as follows:

Definition: 1 *(Isolation Tree) Let T be a node of an isolation tree. T is either an external-node with no child, or an internal-node with one test and exactly two daughter nodes (T_l, T_r). A test at node T consists of an attribute q and a split value p such that the test $q < p$ determines the traversal of a data point to either T_l or T_r. Let $X = \{x_1, ..., x_n\}$ be the given data set of a d-variate distribution. A sample of instances X'$\subset$ X is used to build an isolation tree*[2]*. We recursively divide X' by randomly selecting an attribute q and a split value p, until either: i) Node has only one instance ii) Or, all data at the node have the same values.*

Definition: 2 *(Isolation Forest) Isolation forest is defined as 4-tuple (X, t, ψ, S) where*

- *X is input data,*
- *t is number of trees,*
- *ψ is subsampling size and*
- *S is the set of isolation trees.*

The elements of set S is constructed[1] *by sampling ψ instances from X without replacement.*

```
Algorithm 1: $iForest(X, t, \psi)$
  Complexity: Time - $O(t\psi^2)$, Space - $O(t\psi)$
  Input: $X$ - input data, $t$ - number of trees, $\psi$ - subsampling size
  Output: List of $iTrees$
1 $Forest \leftarrow$ EmptyList
2 for $i = 1$ to $t$ do
3     $X' \leftarrow sampleWithoutReplacement(X, \psi)$
4     $Forest.append(iTree(X'))$
5 end
6 return $Forest$
```

Algorithm 2: $iTree(X)$

```
Complexity: Time - $O(\psi^2)$, Space - $O(\psi)$
Input: $X$ - input data
Output: an $iTree$
q ← $RandomChoice(X.attributes)$
p ← $RandomNumber(X[\text{splitAttr}].min(), X[\text{splittAttr}].max())$
tree ← Node { left ← None, right ← None, size ← $X.size$
        splitAttr ← q, splitVal ← p}
if $X.size > 1$ and $X[splitAttr].numUnique() > 1$ then
    $X_l \leftarrow X.where(q < p)$
    $X_r \leftarrow X.where(q \geq p)$
    tree.left ← $iTree(X_l)$
    tree.right ← $iTree(X_r)$
end
return tree
```

2.3 Evaluation Stage

Algorithm 3: $PathLength(x, T, hlim, e)$

```
Complexity: Time - $O(t\psi)$, Space - $O(1)$
Input: $x$ - input instance, $T$ - an iTree, $hlim$ - height limit, $e$ - current path
       length to be initialized to zero when called first time
Output: path length of $x$
if (T.right is None) and (T.left is none) and ($e \geq hlim$) then
    return $e + c(T.size)$                    // c(...)  is defined in Equation 2.1
end
$a \leftarrow T.splitAttr$
if $x[a] < T.splitVal$ then
    return $PathLength(x, T.left, hlim, e + 1)$
end
else
    return $PathLength(x, T.right, hlim, e + 1)$
end
```

In the evaluation stage[3], a single path length $h(x)$ is derived by counting the number of edges e from the root node to an external node as instance x traverses through an iTree. When the traversal reaches a predefined height limit $hlim$, the return value is e plus an adjustment $c(size)$. This adjustment accounts for estimating an average path length of a random sub-tree which could be constructed using data of $size$ beyond the tree height limit. When $h(x)$ is obtained for each tree of the ensemble, an anomaly score is computed. The anomaly score and the adjustment $c(size)$ is defined in the next section.

2.4 Anomaly Score

The difficulty in deriving an anomaly score from $h(x)$ is that while the maximum possible height of iTree grows in the order of ψ, the average height grows in the order of $log\,\psi$. When required to visualize or compare path lengths from models of different subsampling sizes, normalization of $h(x)$ by any of the above terms either is not bounded or cannot be directly compared. Thus, a normalized anomaly score is needed for the aforementioned purposes.

Since iTrees have an equivalent structure to Binary Search Tree, the estimation of average $h(x)$ for external node terminations is the same as that of the unsuccessful searches in BST.

Section 10.3.3 of [4] gives the average path length of unsuccessful searches in BST as:

$$c(\psi) = \begin{cases} 2H(\psi-1) - 2(\psi-1)/n & \text{for } \psi > 2, \\ 1 & \text{for } \psi = 2, \\ 0 & \text{otherwise.} \end{cases} \tag{2.1}$$

where $H(i) \approx \ln(i) + 0.5772156649$ (euler's constant), is the harmonic number. As $c(\psi)$ is the average of $h(x)$ given ψ, we use it to normalise $h(x)$. The anomaly score s of an instance x is defined as:

$$s(x,\psi) = 2^{\frac{-E[h(x)]}{c(\psi)}} \tag{2.2}$$

where $E[h(x)]$ is the average of $h(x)$ from a collection of iTrees.

Anomaly score $s(x,\psi)$ is interpreted as follows:

1. if $s \approx 1$, instance is abnormal
2. if $s \approx 0$, instance is nominal
3. if $s \approx 0.5$, no distinct anomaly

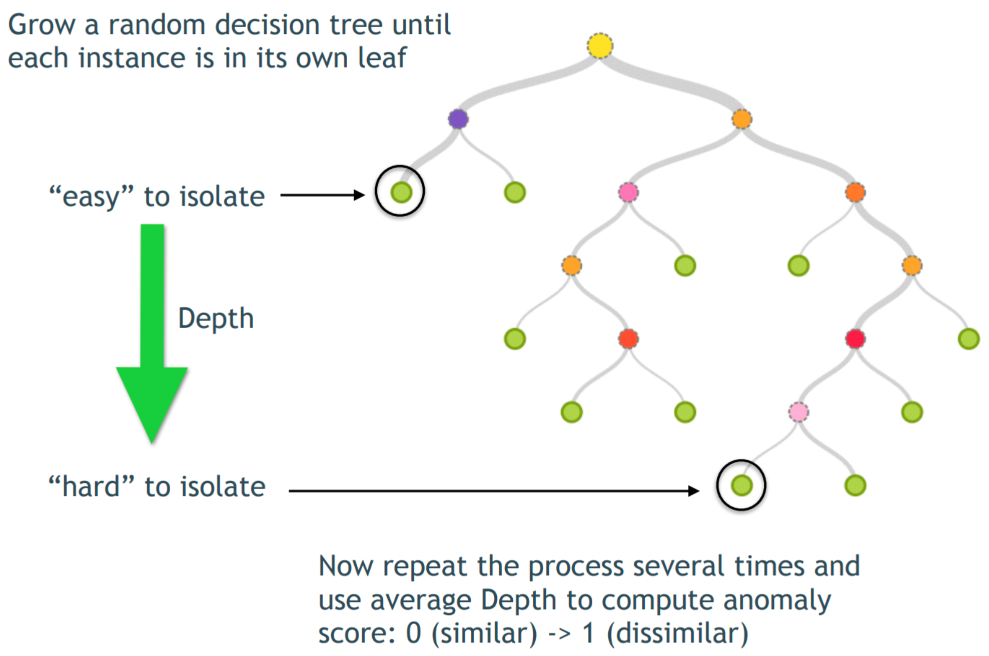

Grow a random decision tree until each instance is in its own leaf
"easy" to isolate
Depth
"hard" to isolate
Now repeat the process several times and use average Depth to compute anomaly score: 0 (similar) -> 1 (dissimilar)

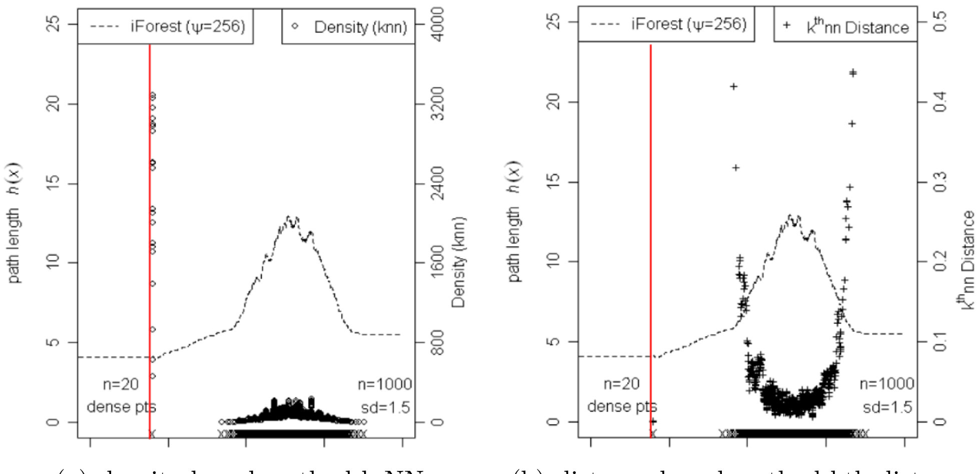
Fig. 2.3: Isolation Forest
Source: slideshare.com

In practical use cases, one has to set a threshold deciding the result. And finding a good threshold is a very difficult task. Most of the times the anomalous points' score overlaps with the nominal points' score.

2.5 Isolation vs. Distance and Density

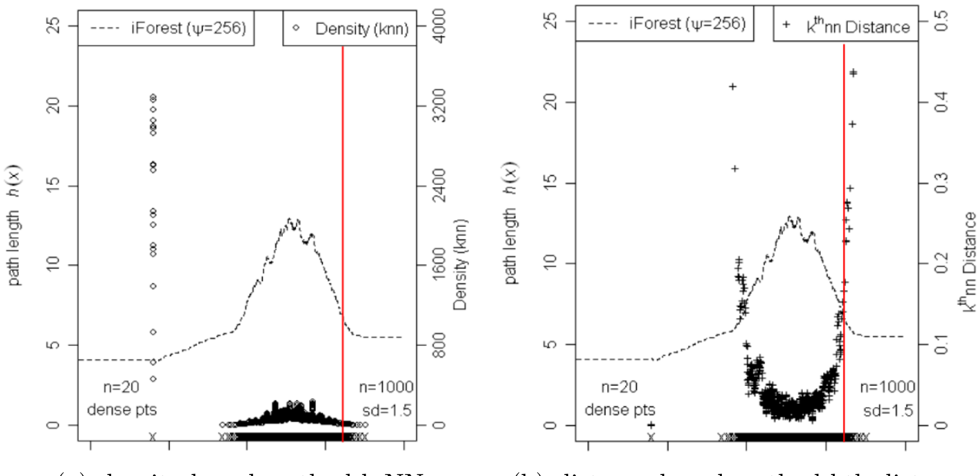
Using basic density measures, the assumption is that 'Normal points occur in dense regions, while anomalies occur in sparse regions'. Using basic distance measures, the basic assumption is that ‘Normal point is close to its neighbours and anomaly is far from its neighbours’.

There are violations of these assumptions: i) high density and short distance do not always imply normal instances ii) low density and long distance do not always imply anomalies.



(a) density based method k-NN (b) distance based method kth-distance

Fig. 2.4: High density and short distance do not always imply normal instances.
Source: Lui et al. [3]



(a) density based method k-NN (b) distance based method kth-distance

Fig. 2.5: Low density and long distance do not always imply anomalies.
Source: Lui et al. [3]

Isolation Forest compares favourably to distance and density based methods in terms of accuracy and processing time. Distance and density based suffers immensly in terms of accuracy and processing time because of curse of dimensionalty.

Distance based methods also suffers from masking and swamping effect. In isolation forest, masking and swamping effects can be managed by adjusting the *hlim* parameter during evaluation. Refer section 4.5, 5.3 and 5.4 of Lui et al. [3].

Codes for Isolation Forest and Isolation Tree can be found at this repository.

2.6 Conclusion

To conclude, isolation forest is one of the most efficient and accurate methods for anomaly detection. It outperforms various density and distance based methods like ORCA, DOLPHIN, LOF, ROF, etc. and their variants in terms of accuracy and performance. Still there are some shortcomings of the isolation forest that we will discuss in next chapter.

Chapter 3

PIDForest

In chapter 1 we discussed some of the challenges that an anomaly detection algorithm face. In the previous chapter we discussed an isolation based ensemble method which is a very good method in terms of complexity and accuracy. Isolation forest tried to address the problems related to masking and swamping. In this chapter we will point out some major issues with isolation forest and discuss another ensemble method, which improves upon those issues.

3.1 Issues with Isolation Forest

Random Split: iForest repeatedly samples a set X' of ψ points from X and builds a random tree with those points as leaves. The tree is built by choosing a random co-ordinate q, and a random value p in its range about which to split. Since iForest chooses which coordinate we split on as well as the breakpoint at random. Thus to be isolated at small depth frequently, picking splits at random must have a good chance of isolating an anomalous point. Although, there are other variants like extended isolation forest which improves on this, but there is no significant improvements.

High Dimensional: As the number of co-ordinates or attributes increases, probability of choosing a sequence of attributes for split which gives rise to most of the anomalies will be very less. Hence, it is very likely that anomalous points won't be isolated near the root and false negative cases will increase. (In anomaly detection problem, anomaly is the true class.)

Presence of non-ordinal categorical attributes: A very big limitation of iForest is that it only works for those datasets where all the features are real-values or ordinal.

3.2 Partial Indentification

We will briefly discuss some concepts present in section 2 of Vatsal et al. [2].

Notations: Let T denote a dataset of n points in d dimensions. Given indices $S \subseteq [d]$ and $x \in R$, let x_S denote the projection of x onto coordinates in S. All the logarithms are to base 2.

3.2.1 Boolean Setting

In Boolean setting $T \subseteq \{0,1\}^d$ and assume that T has no duplicates.

Definition: 3 *(ID for a point)*

$$\begin{gathered} id = \{S \mid S \subseteq [d], x \in T \ \ and \ \forall y \in T \setminus x,\ x_S \neq y_S\} \\ ID(x,T) = arg \min_{S \subseteq [d]} |\{S \mid S \subseteq [d], x \in T \ \ and \ \forall y \in T \setminus x,\ x_S \neq y_S\}| \end{gathered} \tag{3.1}$$

$$idLength(x,T) = |ID(x,T)| \tag{3.2}$$

Definition: 4 *(Imposters)*

$$Imp(x,T,S) = \{y \in T \mid x \in T, S \subseteq [d] \ \ and \ x_S = y_S\} \tag{3.3}$$

Definition: 5 *(Partial ID)*

$$PID(x,T) = arg \min_{S \subseteq [d]} (|S| + log_2(|Imp(x,T,S)|)), \tag{3.4}$$

$$pidLength(x,T) = \min_{S \subseteq [d]} (|S| + log_2(|Imp(x,T,S)|)). \tag{3.5}$$

Geometric view of pidLength: A subcube C of $\{0,1\}^d$ is the set of points obtained by fixing some subset $S \subseteq [d]$ coordinates to values in 0, 1. (Refer section 1 of [5]) The sparsity of T in a subcube C is $\rho_{0,1}(T,C) = \frac{|C|}{|C \cap T|}$. The notation $C \ni x$ means that C contains x, hence $\min_{C \ni x}$ is the minimum over all C that contain x. Anomalies are points that lie in relatively sparse subcubes. Low scores come with a natural witness: a sparse subcube $PID(x,T)$ containing relatively few points from T.

Definition: 6 *(Anomaly Score)*

$$s(x,T) = 2^{-pidLength(x,T)} \tag{3.6}$$

3.2.2 Continuous Setting

Without loss of generality assume that $T \subseteq [0,1]^d$. Length of an interval $I = [a,b], 0 \leq a \leq b \leq 1$ is $len(I) = (b-a)$. A subcube C is specified by a subset of co-ordinates S and intervals $I_j, \forall j \in S$. It consists of all points such that $x_j \in I_j, \forall j \in S$.

Definition: 7 *(Volume of a Subcube)*

$$vol(C) = \prod_i len(I_i) \ where\ C = \prod_j I_j \ and\ I_k = [0,1] \ for\ k \notin S. \tag{3.7}$$

Definition: 8 *(Sparsity of T in C)*

$$\rho(T,C) = \frac{vol(C)}{|C \cap T|} \tag{3.8}$$

Definition: 9 *(PIDScore of x in T)*

$$PIDScore(x,T) = \max_{C \ni x} \rho(T,C),$$
$$PID(x,T) = arg \max_{C \ni x} \rho(T,C). \tag{3.9}$$

Refer section 2.2 of Vatsal et al. [2] to see the analogy to the Boolean case.

3.2.3 Other attributes

To handle attributes over a domain D, we need to specify what subsets of D are intervals and how we measure their length. For discrete attributes, it is natural to define $len(I) = \frac{|I|}{|D|}$. For unordered discrete values, the right definition of interval could be singleton sets, like $country = Brazil$ or certain subsets, like $continent = Americas$. The right choice will depend on the dataset.

3.3 PIDForest Algorithm

Like with iForest, the PIDForest algorithm builds an ensemble of decision trees, each tree is built using a sample of the data set and partitions the space into subcubes. However, the way the trees are constructed and the criteria by which a point is declared anomalous are very different.

Vatsal et al. [2] provided a rough idea on how a PIDForest is to be constructed without emphasising much on the pseudo-code. In this report we will fill the gaps and provide pseudo-codes for various data structures to be implemented for PIDForest.

Each node of a tree corresponds to a subcube C, the children of C represent a disjoint partition of C along some axis $i \in [d]$ (iTree[2] always splits C into two, here finer partition is allowed). The goal is to have large variance in the sparsity (ρ) of the subcubes. Ultimately, the leaves with large sparsity values will point to regions with anomalies.

For each tree, we pick a random sample $P \subseteq T$ of m points, and use that subset to build the tree. Each node v in the tree corresponds to subcube $C(v)$, and a set of points

$P(v) = C(v) \cap P$. For the root, $C(v) = [0,1]^d$ and $P(v) = P$. At each internal node, we pick a coordinate $j \in [d]$, and breakpoints $t_1 \leq ... \leq t_{k-1}$ which partition I_j into k intervals, and split C into k subcubes.

How to choose the partition? We want to partition the cube into some sparse regions and some dense regions. This idea is formulized in section 3 of Vatsal et al. [2] and the objective functions turns out to be:

$$arg \max_{\{C^i\}_{i\in k}} Var(P, \{C^i\}_{i\in k}) \tag{3.10}$$

Maximizing the variance has the advantage that it turns out to equivalent to a well-studied problem about histograms, and admits a very efficient streaming algorithm. Here we are going to use $(1+\epsilon)-$factor approximation algorithm for histogram construction given by Guha et al. [6].

Algorithm 4: $PIDForest(X, t, \psi, h, k, \epsilon)$

```
Complexity: Time - $O(td\psi log(\psi))$
Input: $X$ - input data, $t$ - number of trees, $\psi$ - subsampling size, $h$ - max depth,
  $k$ - max partitions, $\epsilon$ - for histogram construction
Output: List of $PIDTree$
Forest ← {
          trees : EmptyList,
          start : EmptyDictionary,
          end : EmptyDictionary
    }
for $attr$ in $X.attributes$ do
    $Forest.start[attr] = X[attr].min() - \delta$    // set $\delta$ to 1e-4 for precision issues
    $Forest.end[attr] = X[attr].max() + \delta$
end
for $i = 1$ to $t$ do
    $X' \leftarrow sampleWithoutReplacement(X, \psi)$
    $Forest.trees.append(PIDTree(X', 0, Forest.start, Forest.end, h, k, \epsilon))$
end
return $Forest$
```

Algorithm 5: $PIDTree(X, e, start, end, h, k, \epsilon)$

```
Complexity: Time - $O(d\psi log(\psi))$
Input: $X$ - input data, $e$ - current depth, [$start$, $end$] - interval for each attribute,
       $h$ - max depth, $k$ - max partitions, $\epsilon$ - for histogram construction
Output: a $PIDTree$
1  $tree \leftarrow \{$
2          child : EmptyList,
3          depth : e,
4          sparsity : (-1)
5      }
6  $tree.cube = Cube(start, end, \&tree)$
   // &x stands for a reference of x
7  $tree.pointset = Pointset(filter(X, tree.cube), \&tree)$
8  if $tree.depth < h$ and $|tree.pointset| > 1$ then
       // if not a leaf node then split
9      $tree.child \leftarrow findSplit(...)$
10 end
11 else
12     $tree.sparsity \leftarrow tree.cube.logvolume - log(|tree.pointset.X|)$
13 end
14 return $tree$
```

Algorithm 6: $Cube(start, end, node)$

```
Input: [$start$, $end$] - interval for each attribute, $node$ - ref. of containing PIDTree
Output: a $Cube$
1  $cube \leftarrow \{$
2          node : node
3          start : start,
4          end : end,
5          logvolume : 0                      // log of volume of subcube
6          child : EmptyList
7      }
8  for $attr$ in $cube.start.keys()$ do
       // recall that cube.start is a dictionary whose keys are attributes
9      $cube.logvolume += log(cube.start.end[attr] - cube.start.start[attr])$
10 end
11 return $cube$
```

Algorithm 7: *Pointset*$(X, node)$

```
Input: $X$ - input data, $node$ - to which node this set belongs to
Output: a $Pointset$
$pointset \leftarrow$ {
            X : X,
            node : node,
            val : EmptyDictionary,
            count : EmptyDictionary,
            gap : EmptyDictionary
      }
// val, count and gap is used for converting to histogram problem
// refer Appendix A of [2]
for $attr$ in $X.attributes$ do
    // np.unique is a function from python numpy library
    $v, c \leftarrow$ np.unique(X[attr], return_counts=True)
    $pointset.val[attr] \leftarrow v$
    $pointset.count[attr] \leftarrow c$
    $pointset.gap[attr] \leftarrow [0]$
    if $|v| > 1$ then
        // sum of all elements of pointset.gap for a attr is equal to
        // start[attr] - end[attr]
        $g \leftarrow$ List of 0's of size $|v|$
        $g[0] = (v[0] + v[1])/2 - pointset.node.cube.start[attr]$
        // negative indexing is same as python programming language
        $g[-1] = pointset.node.cube.end[col] - (v[-1] + v[-2])/2$
        for $i$ in range(1,$|v|$ - 1) do
            $g[i] = (v[i+1] - v[i-1])/2$
        end
        $pointset.gap[attr] \leftarrow g$
    end
end
return $pointset$
```

Algorithm 5 uses $findSplit(...)$ method to partition the current subcube into disjoint child subcubes. $findSplit(...)$ uses the $AHIST-S(...)$ procedure given by Guha et. al [6] which is $(1+\epsilon)-$factor approximation of V-optimal histogram construction. Due to complexity of the $findSplit(...)$ function we have not provided it's pseudo-code. You can find my implementation of the PIDForest and AHIST-S at this code repository.

Producing an anomaly score for each point is fairly straightforward. Say we want to compute a score for $y \in [0,1]^d$. Each tree in the forest maps y to a leaf node v and gives it a score $PIDTree.sparsity$. We take the 70-80% percentile score as our final score (any robust analog of the max will do).

3.4 Comparison with isolation forest

PIDForest improves upon the issues with isolation forest. Instead of choosing an attribute for partitioning a node into disjoint sets, PIDForest uses AHIST-S method to select an attribute with higher variance. Instead of choosing a random splitting value which is used to partition the set into two, PIDForest used AHIST-S method to get the optimal buckets which is used to do finer partition not limited to 2.

As discussed in section 3.1 isolation forest struggles to give decent results for high dimensional data, on the other hand PIDForest doesn't chooses the splitting attribute with uniform probability, it gives more preference to the attribute which has more variance, therefore doesn't suffer much from higher dimension.

PIDForest is more complex data structure in comparison to Isolation Forest.

3.5 Conclusion

PIDForest is arguably one of the best off-the-shelf algorithms for anomaly detection on a large, heterogenous dataset. It inherits many of the desirable features of Isolation Forests,while also improving on it in important ways. This ends the breif discussion on

PIDForest algorithm, you can refer to Vatsal et al. [2] for more details.

In chapter 1 I discussed some of the major challenges face by any anomaly detection algorithm. Masking and swamping is well handled by the isolation forest and pidforest. Concept drift is a major challenge in online anomaly detection. Because of small time and space complexity of these methods, these methods can be easily used for online anomaly detection, i.e, we can retrain the model as the new data comes but this is a very poor way to handle the concept drift. Another problem with these ensemble methods is poor handling of categorical attributes. We have already seen that isolation forest can't handle non-ordinal data, whereas PIDForest requires external input from domain expert.

In the next chapter I will present some modifications to these existing methods and present a general framework in which these methods can be improved to handle categorical data, concept drift and online anomaly detection.

Chapter 4

Contributions

We concluded previous chapter section 3.5 by mentioning these issues:

- Online Anomaly detection
- Handling categorical attributes
- Concept Drift

In this chapter I will present some enhancements that can be made to improve the performance of isolation forest and pidforest.

4.1 Feedback guided anomaly detection

Anomaly detectors are often used to produce a ranked list of statistical anomalies, which are examined by human analysts in order to extract the actual anomalies of interest.

This can be exceedingly difficult and time consuming when most high-ranking anomalies are false positives and not interesting from an application perspective.

Siddiqui et al. [7] address this problem and gives a general framework of how we can convert unsupervised anomaly detection to a semi-supervised anomaly detection problem in which a feedback is given by a domain expert which is used to improve the accuracy of the anomaly detection model. Feedback guided anomaly discovery can be model in online convex optimization (OCO) framework.

4.1.1 Online convex optimization

OCO is formulated as an iterative game against a potentially adversarial environment where our moves are vectors from a convex set S. At discrete time steps t the game proceeds as follows:

1. We select a vector $w_t \in S$.
2. The environment selects a convex function $f_t : S \rightarrow R$.
3. We suffer a loss $f_t(w_t)$.

The goal is to select a sequence of vectors with small accumulated loss over time. Given, a T-step game episode where we play $(w_1, w_2, \dot{;} w_T)$ against $(f_1, f_2, \dot{;} f_T)$ the total T step regret is equal to:

$$Regret_T = \sum_{t=1}^{T} f_t(w_t) - \min_{w^* \in S} \sum_{t=1}^{T} f_t(w^*) \tag{4.1}$$

Refer chapter 2 of [8] for more details.

4.1.2 Modelling in OCO framework

Query-guided anomaly discovery can also be viewed as a game where on each round we output an anomaly ranking over the data instances and we get feedback on the top-ranked instance. We wish to minimize the number of times we receive ”nominal” as the feedback response.

To put this problem in OCO framework Siqqiqui et al. [7] have put some reasonable restrictions on the form of the anomaly detectors that we will consider. Only family of generalized linear anomaly detectors(GLADs) which are defined by i) a feature function $\phi : D \rightarrow R^n$, which maps data instances to n-dimenstional vectors and ii) n-dimenstional weight vector w are considered. In this context anomaly score for an instance x is defined to be $SCORE(x; w) = -\phi \cdot w$ with larger score corresponding to more anomalous instances.

Given, a GLAD parameterization of an anomaly detector, we can now connect query-guided anomaly discovery to OCO.

On each feedback round we select a vector w_t for the detector, which specifies an anomaly ranking over instances. We recieve feedback y_t on the top ranked instance, where $y_t = +1$ if the instance is alien and $y_t = -1$ if it is nominal.

There are three choices of loss function given in the Siddqui et al. [7]: i) linear loss ii) log-likelihood loss iii) logistic loss.

With the experiments, I came to conclusion that overall linear loss performs better than other two in terms of performance, computational complexity, and accuracy. Hence, throughout this report I will stick with only linear loss.

Definition: 10 *(Linear loss) Let x_t be the top-ranked instance in D under the ranking given by w_t. The linear loss is given by*

$$f_t(w_t) = -y_t SCORE(x_t; w_t) = y_t w_t \cdot \phi(x_t) \tag{4.2}$$

Algorithm 1 of Siqqiqui et al. [7] gives a general framework in which OCO can be applied on anomaly detector methods for query-guided anomaly discovery. In the next section we will model the isolation forest and pidforest in OCO framework.

4.2 Feedback guided isolation forest

The isolation forest assigns an anomaly score to an instance x based on its average isolation depth across the randomized forest, ref [3]. In particular, the score is (a normalized version of) the negative of this average depth.

We need to define a GLAD model that replicates isolation forest.

Define $\phi_e(x)$ be a binary feature that is 1 if instance x goes through the edge and 0 otherwise. **Define** w_e be the weight of each edge. **Define** ϕ be a vector that concatenate all of the features across the forest in a consistent order. **Define** w be a vector that concatenate all of the weights across the forest in a consistent order.

Now the modified model for isolation forest is given by the following algorithms:

```
Algorithm 8: $feedbackITree(X)$
  Complexity: Time - $O(\psi^2)$, Space - $O(\psi)$
  Input: $X$ - input data
  Output: a $feedbackITree$
  q ← $RandomChoice(X.attributes)$
  p ← $RandomNumber(X$[splitAttr]$.min(), X$[splittAttr]$.max())$
  ftree ← Node { left ← None, right ← None,
                 size ← $X.size$, splitAttr ← q,
                 splitVal ← p, $w$ ← 1, $\theta$ ← 1} ;          // $w, \theta$ for mirror descent
  if $X.size > 1$ and $X[splitAttr].numUnique() > 1$ then
      $X_l$ ← $X.where(q < p)$
      $X_r$ ← $X.where(q \geq p)$
      ftree.left ← $feedbackITree(X_l)$
      ftree.right ← $feedbackITree(X_r)$
  end
  return $ftree$
```

```
Algorithm 9: $unadjustedPathLength(x, T, hlim, e)$
  Complexity: Time - $O(t\psi)$, Space - $O(1)$
  Input: $x$ - input instance, $T$ - a $feedbackITree$, $hlim$ - height limit, $e$ - current
         path length to be initialized to zero when called first time
  Output: path length of $x$
  if ($T.right$ is None) and ($T.left$ is none) and ($e \geq hlim$) then
      return $e$                // removed the adjustment, return unadjusted path length
  end
  $a$ ← $T.splitAttr$
  // (e + 1) → e + (weight of the edge)
  if $x[a] < T.splitVal$ then
      return $PathLength(x, T.left, hlim, e + T.w)$
  end
  else
      return $PathLength(x, T.right, hlim, e + T.w)$
  end
```

Algorithm 10: $updateWeights(x, T, hlim, \eta, y)$

```
Complexity: Time - $O(t\psi)$, Space - $O(1)$
Input: $x$ - input instance, $T$ - a $feedbackITree$, $hlim$ - height limit, $\eta$ - learning
       rate, $y$ - feedback
if (T.right is None) and (T.left is none) and ($e \geq hlim$) then
    return                                    // no child node, all weights updated
end
$a \leftarrow T.splitAttr$
if $x[a] < T.splitVal$ then
    nextNode $\leftarrow T.left$
end
else
    nextNode $\leftarrow T.right$
end
// mirror descent update
$nextNode.\theta \leftarrow nextNode.\theta - \eta * y$
$nextNode.w \leftarrow nextNode.\theta * (nextNode.\theta \geq 0)$
$updateWeights(x, \&nextNode, hlim, \eta, y)$
```

You can find the complete implementation of the feedback guided isolation forest at my respository: `https://github.com/KishoreKaushal/AnomalyDetection`

4.3 Feedback guided PIDForest

The anomaly score for PIDForest is the maximum sparsity among all the PIDTrees. We need to define a GLAD model that replicates pidforest.

Define $\phi_{leaf}(x)$ be a binary feature for a *leaf* node which is 1 if instance x falls in that leaf. **Define** w_{leaf} be the sparsity of that *leaf* node. **Define** ϕ be a vector that concatenate all of the features across the forest in a consistent order. **Define** w be a vector that concatenate all of the weights across the forest in a consistent order.

Now the modified model for pidforest is given by the following algorithms:

Algorithm 11: $feedbackPIDTree(X, e, start, end, h, k, \epsilon)$

```
Complexity: Time - $O(d\psi log(\psi))$
Input: $X$ - input data, $e$ - current depth, [$start$, $end$] - interval for each attribute,
       $h$ - max depth, $k$ - max partitions, $\epsilon$ - for histogram construction
Output: a $feedbackPIDTree$
1  $ftree \leftarrow \{$
2          child : EmptyList,
3          depth : e,
4          sparsity : (-1),
5          $w$ : (-1)
6          $\theta$ : (-1)
7      }
8  $ftree.cube = Cube(start, end, \&ftree)$
   // &x stands for a reference of x
9  $ftree.pointset = Pointset(filter(X, ftree.cube), \&ftree)$
10 if $ftree.depth < h$ and $|ftree.pointset| > 1$ then
       // if not a leaf node then split
11     $ftree.child \leftarrow findSplit(...)$
12 end
13 else
14     $ftree.sparsity \leftarrow ftree.cube.logvolume - log(|ftree.pointset.X|)$
       // $w, \theta$ will be used for mirror descent algorithm
15     $ftree.\theta \leftarrow ftree.w \leftarrow ftree.sparsity$
16 end
17 return $ftree$
```

Algorithm 12: $updateWeights(x, T, \eta, y)$

```
Complexity: Time - $O(t\psi)$, Space - $O(1)$
Input: $x$ - input instance, $T$ - a $feedbackPIDTree$, $\eta$ - learning rate, $y$ - feedback
1  if $|T.child| == 0$ then
       // mirror descent update
2      $nextNode.\theta \leftarrow nextNode.\theta - \eta * y$
3      $nextNode.w \leftarrow nextNode.\theta * (nextNode.\theta \geq 0)$
4      return
5  end
   // get the nextNode where instance $x$ falls
6  $updateWeights(x, \&nextNode, hlim, \eta, y)$
```

Chapter 5

AI in Image Compression

This chapter contains information about work done before COVID-19 pandemic lockdown.

5.1 Construction

Write ...

5.2 Improved Method

Write...

5.3 Conclusion

In this chapter, we proposed another distributed algorithm for XYZ. This algorithm has both time complexity of $O(n)$ where n is the total number of nodes. In next chapter, we conclude and discuss some of the future aspects.

Chapter 6

Conclusion and Future Work

write results of your thesis and future work.

References

[1] Pham H, *Handbook of Engineering Satistics.* Springer, London, 2006. [Online]. Available: https://doi.org/10.1007/978-1-84628-288-1

[2] P. Gopalan, V. Sharan, and U. Wieder, "Pidforest: Anomaly detection via partial identification," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 15 809–15 819. [Online]. Available: http://papers.nips.cc/paper/9710-pidforest-anomaly-detection-via-partial-identification.pdf

[3] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation-based anomaly detection," *ACM Trans. Knowl. Discov. Data*, vol. 6, no. 1, Mar. 2012. [Online]. Available: https://doi.org/10.1145/2133360.2133363

[4] B. R. Preiss, *Data Structures and Algorithms with Object-Oriented Design Patterns in C++.* USA: John Wiley and Sons, Inc, 1998.

[5] D. ELLIS, "Irredundant families of subcubes," *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 150, no. 2, p. 257–272, 2011.

[6] S. Guha, N. Koudas, and K. Shim, "Approximation and streaming algorithms for histogram construction problems," *ACM Trans. Database Syst.*, vol. 31, no. 1, p. 396–438, Mar. 2006. [Online]. Available: https://doi.org/10.1145/1132863.1132873

[7] M. A. Siddiqui, A. Fern, T. G. Dietterich, R. Wright, A. Theriault, and D. W. Archer, “Feedback-guided anomaly discovery via online optimization,” in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD ’18. New York, NY, USA: Association for Computing Machinery, 2018, p. 2200–2209. [Online]. Available: https://doi.org/10.1145/3219819.3220083

[8] S. Shalev-Shwartz, “Online learning and online convex optimization,” *Found. Trends Mach. Learn.*, vol. 4, no. 2, p. 107–194, Feb. 2012. [Online]. Available: https://doi.org/10.1561/2200000018