Artificial Intelligence for Image Processing and Anomaly Detection

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 "**Artificial Intelligence for Image Processing and Anomaly Detection**" 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

Phase-1: Before Lockdown

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

Full Report: `https://github.com/KishoreKaushal/btp-report-phase2`

Chapter 1

AI in Image Compression

In this chapter I will breifly summarise my works till March 17, 2020. The full version of this report can be found here: midsem-report.pdf

1.1 Compression

A data compression algorithm transforms the data to occupy a less space. The original data is encoded by a program called encoder, to a compressed representation using a fewer number of bits. Decoder is responsible for decompressing the compressed representation.

There are two kinds of compression technique: lossy compression and lossless compression. The compression technique where the decompressed data is exactly same as original data is called as lossless compression otherwise it is known as lossy compression technique because some information is lost during coding-encoding phase.

Two well-known codecs for image compression are JPEG and PNG. PNG is lossless and JPEG is lossy.

1.2 Compression using PCA

Principal components analysis (PCA) is one of a family of techniques for taking high-dimensional data, and using the dependencies between the variables to represent it in a more tractable, lower-dimensional form, without losing too much information. PCA is one of the simplest and most robust ways of doing such dimensionality reduction. PCA is mathematically defined as an orthogonal linear transformation that transforms the data to a new coordinate system such that the greatest variance by some scalar projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on.

Let W be a $d \times d$ matrix whose columns are the principal components of X. The transformation $T = XW$ maps a data vector $x_{(i)}$ from an original space of d variables to a new space of d variables which are uncorrelated over the dataset. However, not all the principal components need to be kept. Keeping only the first L principal components, produced by using only the first L eigenvectors, gives the truncated transformation:

$T_L = XW_L$

where the matrix T_L now has n rows but only L columns. In other words, PCA learns a linear transformation $t = W^T x, x \in \Re^d, t \in \Re^L$, where the columns of $d \times L$ matrix W form an orthogonal basis for the L features (the components of representation t) that are decorrelated. By construction, of all the transformed data matrices with only L columns, this score matrix maximises the variance in the original data that has been preserved, while minimising the total squared reconstruction error $\left\|TW^T - T_L W_L^T\right\|_2^2 or \left\|X - X_L\right\|_2^2$.

The basic steps for computing the PCA is as follows:

1. Standardize the d-dimensional dataset.

2. Construct the covariance matrix.

3. Decompose the covariance matrix into its eigenvectors and eigenvalues.

4. Sort the eigenvalues by decreasing order to rank the corresponding eigenvectors.

5. Select L eigenvectors which correspond to the L largest eigenvalues, where L is the dimensionality of the new feature subspace $L \leq d$.

6. Construct a projection matrix W_L from the "top" L eigenvectors.

7. Transform the d-dimensional input dataset X using the projection matrix W_L to obtain the new L-dimensional feature subspace.

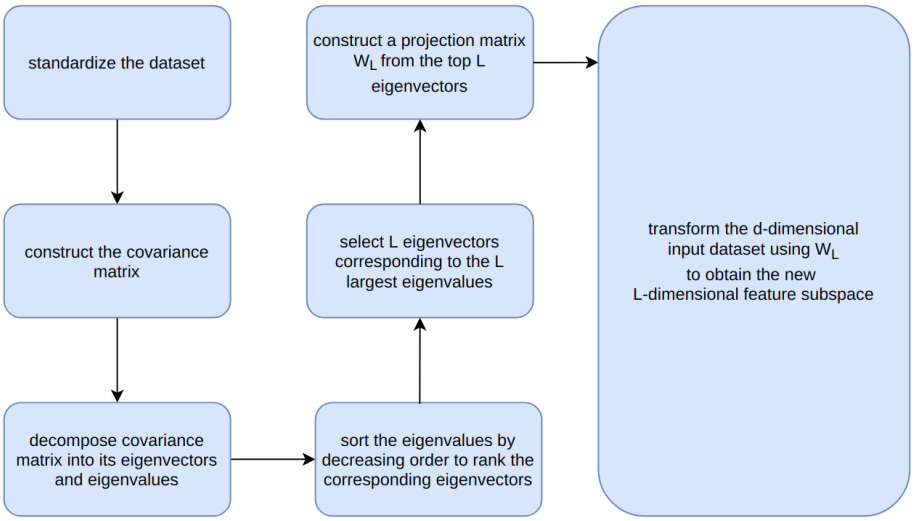


Fig. 1.1: Steps for dimensionality reduction using PCA

1.3 Image compression using PCA

We will follow the steps mentioned in the previous section for dimensionality reduction using PCA. The results of the experiments are shown in the figure 5.2.

You can find the implementation of the image compression using PCA here:

@KishoreKaushal/ImageCompression/

It is clear from this experiment that higher variance leads to more number of principal components and higher is the reconstructed image quality and lower the compression rate.

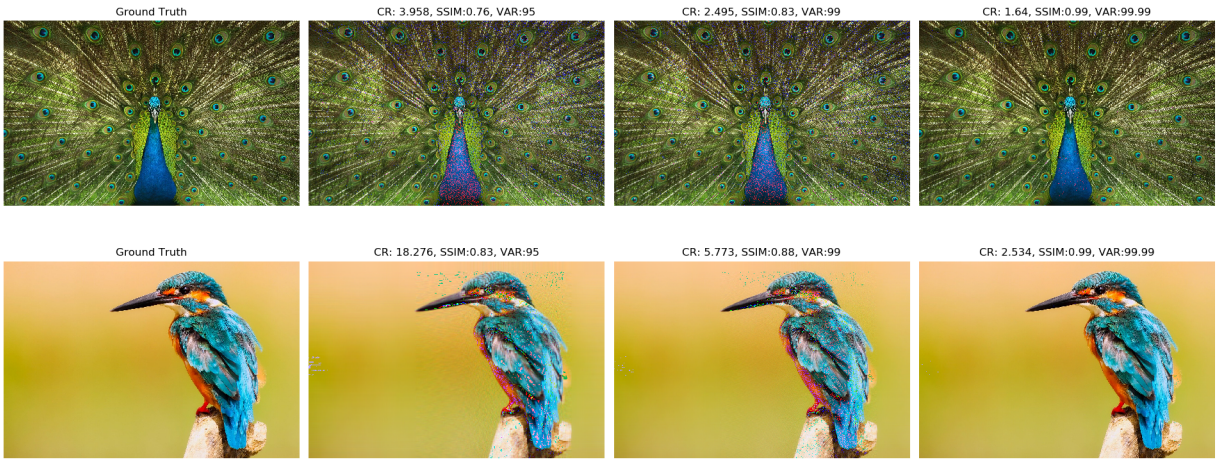


Fig. 1.2: CR = Compression Ratio, SSIM = Structural Similarity Index, VAR = Variance. Results of compression using the PCA method. Higher variance leads to more number of principal components and higher is the reconstructed image quality and lower the compression rate.

Infact, the first L principal components is selected to get atleast the given number of variance.

1.4 JPEG Compression

I will summarise the JPEG compression algorithm using the flowchart shown in the next figure. You can refer to my previous report for a detailed analysis.

The image is first split in $8x8$ blocks. Then, the image in RGB space is transformed to YCbCr space, followed by a downsampling and a forward discrete cosine transform (DCT). The final blocks are then quantized with an quantization matrix which is adjusted accoirding to the quality factor. Finally, the quantized block of values are arranged in form of a row-vector and entropy encoded.

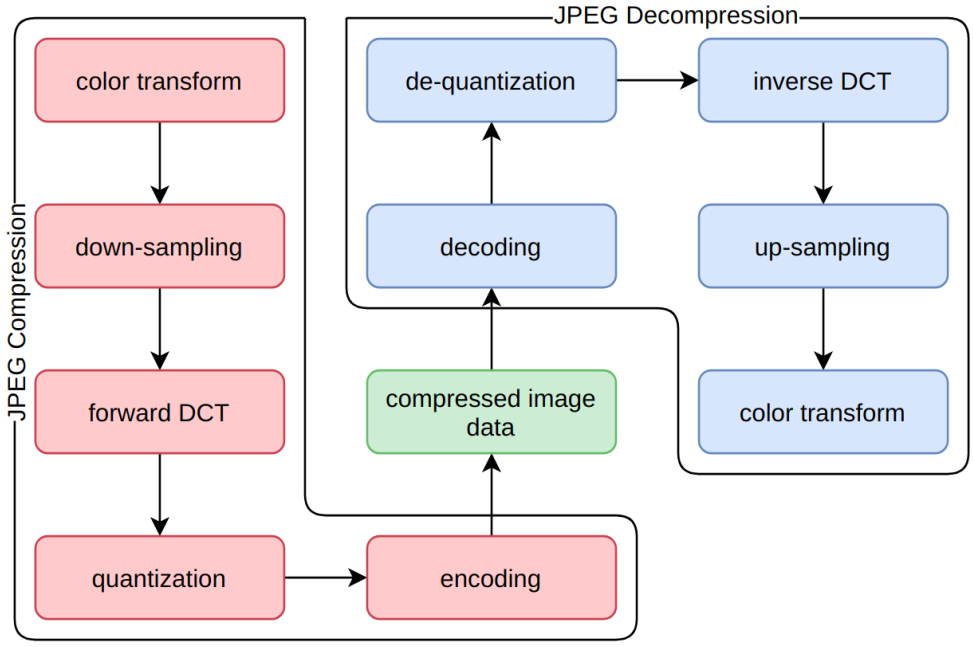


Fig. 1.3: JPEG Schematic

1.5 Perceptual Image Comparison

Prakash et al. [1] introduced a powerful CNN tailored to the specific task of semantic image understanding to achieve higher visual quality in lossy compression. A modest increase in complexity is incorporated into the encoder which allows a standard, off-the-shelf jpeg decoder to be used. While JPEG encoding may be optimized for generic images, the process is ultimately unaware of the specific content of the image to be compressed. This technique makes JPEG content-aware by designing and training a model to identify multiple semantic regions in a given image.

The idea here is to locate multiple regions of interest (ROI) within a single image and noting the fact that it's not an object detection problem and hence the precision of the boundary doesn't matter. Also, the model needs to learn a single class-invariant feature map by learning separate feature maps for each of a set of object classes and then summing over the top features.

We then discussed methods for object localization like multi-structure region of interest

and CAM which allow us to detect the region of interest in the image. We can then use this to train our omodel to generate a better quantization matrix for a particular task.

1.6 Conclusion

In this chapter we breifly discussed our works before till Mar 17, 2020.

Phase-2: After Lockdown

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

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

Chapter 2

Introduction

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

2.1 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.

2.2 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: [2]

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.

2.3 Challenges

2.3.1 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.

2.3.2 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.

2.3.3 Miscellaneous

Apart from the above conceptual challenges, here are some general challenges pointed out by Vatsal et al. [3]

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 data-center might be compromised and behave abnormally.

2.4 Organization of The Report

This chapter 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 3 we will discuss a very efficient ensemble method Isolation Forest for anomaly detection. In chapter 4 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 chapter 5 we will present some methods to address these issues.

Chapter 3

Isolation Forest

3.1 Isolation based anomaly detection

Isolation is the process or fact of isolating or being isolated. The authors of [4] 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. [4] 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.

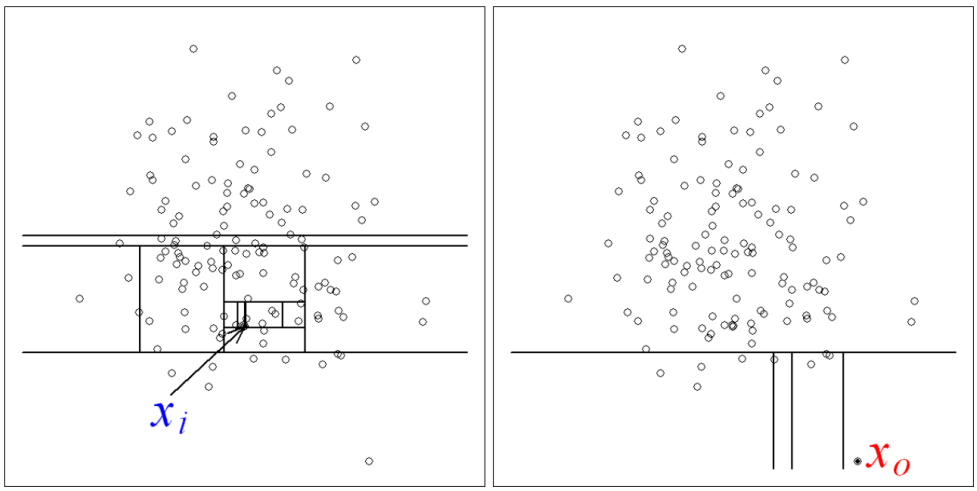


Fig. 3.1: **Left**: a normal point x_i requires twelve random partitions to be isolated; **Right**: an anomaly x_o requires only four partitions to be isolated.
Source: Lui et al. [4]

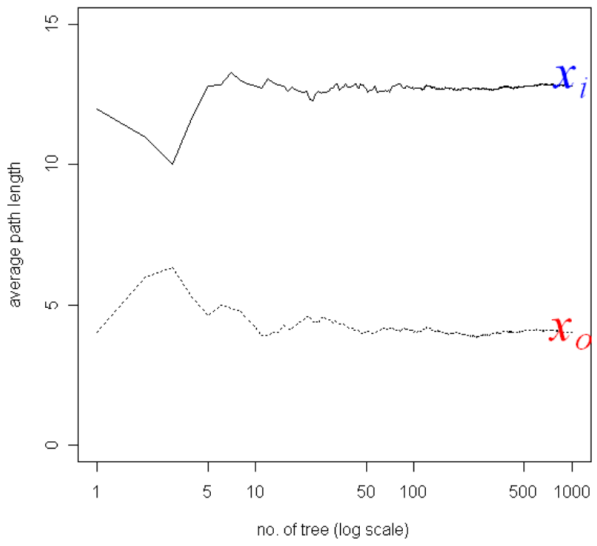


Fig. 3.2: Averaged path lengths of x_i and x_o converge when the number of trees increases.
Source: Lui et al. [4]

3.2 Isolation forest algorithm

Formally, we can define isolation forest 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, ψ - subsampling size
Output: List of $iTrees$

```
Forest ← EmptyList
for i = 1 to t do
    X' ← sampleWithoutReplacement(X, ψ)
    Forest.append(iTree(X'))
end
return Forest
```

Algorithm 2: $iTree(X)$

Complexity: Time - $O(\psi^2)$, Space - $O(\psi)$

Input: X - input data

Output: an $iTree$

```
1  q ← RandomChoice(X.attributes)
2  p ← RandomNumber(X[splitAttr].min(), X[splittAttr].max())
3  tree ← Node { left ← None, right ← None, size ← X.size
4                splitAttr ← q, splitVal ← p}
5  if X.size > 1 and X[splitAttr].numUnique() > 1 then
6  |  X_l ← X.where(q < p)
7  |  X_r ← X.where(q ≥ p)
8  |  tree.left ← iTree(X_l)
9  |  tree.right ← iTree(X_r)
10 end
11 return tree
```

3.3 Evaluation

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

```
1  if (T.right is None) and (T.left is none) and (e ≥ hlim) then
2  |  return e + c(T.size)                    // c(...) is defined in Equation 3.1
3  end
4  a ← T.splitAttr
5  if x[a] < T.splitVal then
6  |  return PathLength(x, T.left, hlim, e + 1)
7  end
8  else
9  |  return PathLength(x, T.right, hlim, e + 1)
10 end
```

In the evaluation stage, 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.

3.4 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 [5] gives the average path length of unsuccessful searches in BST as:

$$c(\psi) = \begin{cases} 2H(\psi-1) - 2(\psi-1)/n & \text{for } \psi > 2, \\ 1 & \text{for } \psi = 2, \\ 0 & \text{otherwise.} \end{cases} \tag{3.1}$$

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:

$$s(x, \psi) = 2^{\frac{-E[h(x)]}{c(\psi)}} \tag{3.2}$$

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

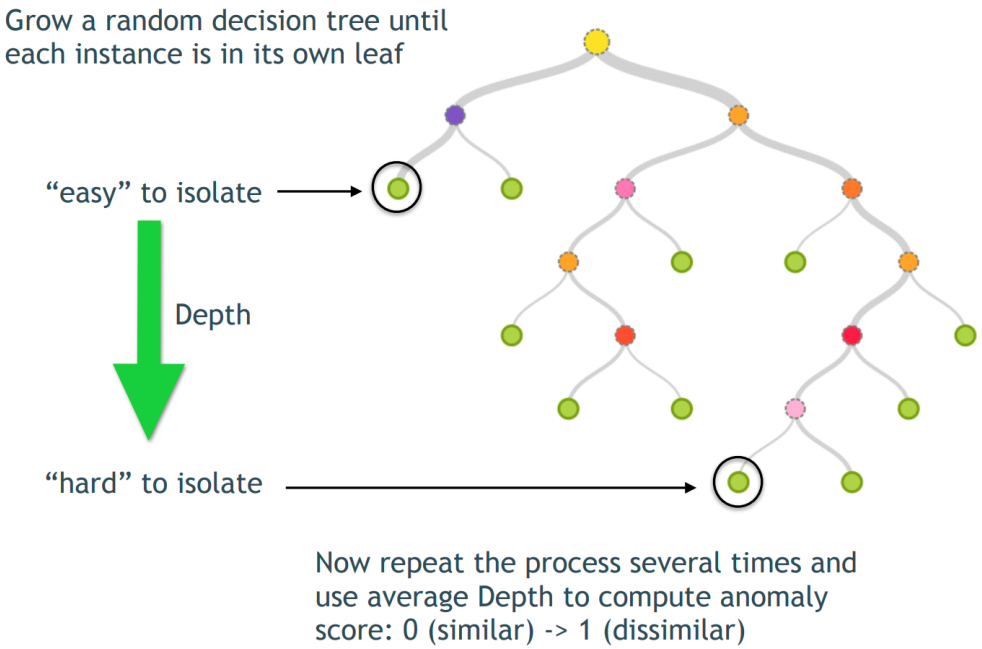


Fig. 3.3: Isolation forest
Source: slideshare.com

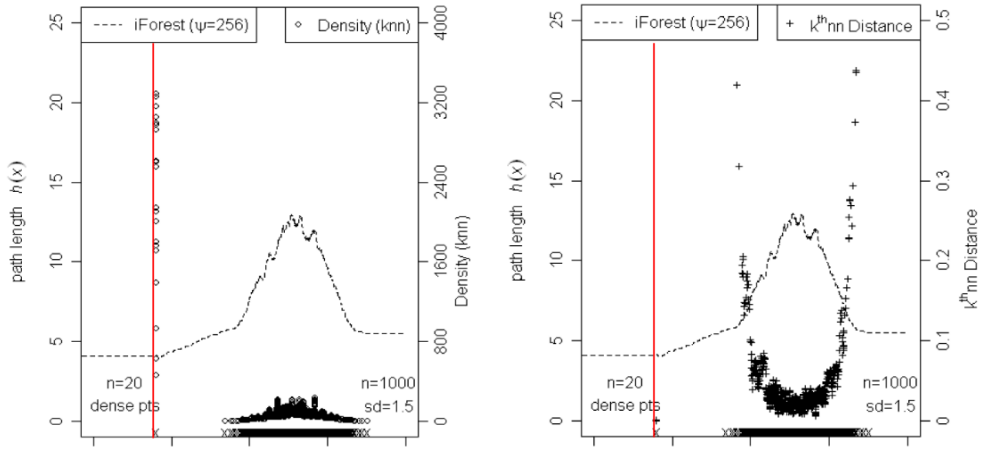
In practical use cases, one has to set a threshold deciding the result. And finding a good threshold is a very difficult task. Most of the times the anomalous points' score overlaps with the nominal points' score.

3.5 Comparing isolation with distance and density measure

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

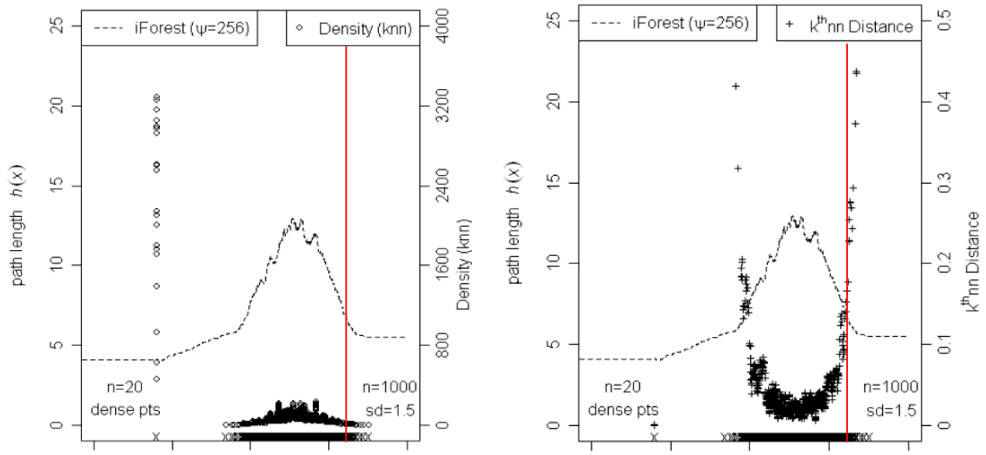
assumption is that 'Normal point is close to its neighbours and anomaly is far from its neighbours'.

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. 3.4: High density and short distance do not always imply normal instances.
Source: Lui et al. [4]



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

Fig. 3.5: Low density and long distance do not always imply anomalies.
Source: Lui et al. [4]

Isolation forest compares favourably to distance and density based methods in terms of

accuracy and processing time. Distance and density based suffers immensely in terms of accuracy and processing time because of curse of dimensionality.

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. [4].

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

3.6 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 4

PIDForest

In chapter 2 we discussed some 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.

4.1 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 isolation forest 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 isolation forest is that it only works for those datasets where all the features are real-values or ordinal.

4.2 Partial Identification

We will briefly discuss some concepts present in section 2 of Vatsal et al. [3].

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.

4.2.1 Boolean Setting

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

Definition: 3 *(ID for a point)*

$$\begin{aligned} id &= \{S \mid S \subseteq [d], x \in T \ and \ \forall y \in T \setminus x,\ x_S \neq y_S\} \\ ID(x,T) &= arg\min_{S \subseteq [d]} |\{S \mid S \subseteq [d], x \in T \ and \ \forall y \in T \setminus x,\ x_S \neq y_S\}| \end{aligned} \tag{4.1}$$

$$idLength(x,T) = |ID(x,T)| \tag{4.2}$$

Definition: 4 *(Impostor)*

$$Imp(x,T,S) = \{y \in T \mid x \in T, S \subseteq [d] \ and \ x_S = y_S\} \tag{4.3}$$

Definition: 5 *(Partial ID)*

$$PID(x,T) = arg\min_{S\subseteq[d]}(|S|+log_2(|Imp(x,T,S)|)), \tag{4.4}$$

$$pidLength(x,T) = \min_{S\subseteq[d]}(|S|+log_2(|Imp(x,T,S)|)). \tag{4.5}$$

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 [6]) 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: sparse subcube $PID(x,T)$ containing relatively few points from T.

Definition: 6 *(Anomaly Score)*

$$s(x,T) = 2^{-pidLength(x,T)} \tag{4.6}$$

4.2.2 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)*

$$vol(C) = \prod_i len(I_i) \text{ where } C = \prod_j I_j \text{ and } I_k = [0,1] \text{ for } k \notin S. \tag{4.7}$$

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

$$\rho(T,C) = \frac{vol(C)}{|C\cap T|} \tag{4.8}$$

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

$$PIDScore(x, T) = \max_{C \ni x} \rho(T, C),$$
$$PID(x, T) = arg \max_{C \ni x} \rho(T, C). \tag{4.9}$$

Refer section 2.2 of Vatsal et al. [3] to see the analogy to the Boolean case.

4.2.3 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, and it requires input from domain expert.

4.3 PIDForest Algorithm

Like with isolation forest, 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. [3] 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. [3] and the objective functions turns out to be:

$$arg \max_{\{C^i\}_{i \in k}} Var(P, \{C^i\}_{i \in k}) \tag{4.10}$$

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. [7].

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

Complexity: Time - $O(td\psi log(\psi))$

Input: X - input data, t - number of trees, ψ - subsampling size, h - max depth, k - max partitions, ϵ - 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() − δ    // set δ to 1e-4 for precision issues
    Forest.end[attr] = X[attr].max() + δ
end
for i = 1 to t do
    X' ← sampleWithoutReplacement(X, ψ)
    Forest.trees.append(PIDTree(X', 0, Forest.start, Forest.end, h, k, ε))
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, ϵ - for histogram construction

Output: a $PIDTree$

```
tree ← {
            child : EmptyList,
            depth : e,
            sparsity : (-1)
        }
tree.cube = Cube(start, end, &tree)
// &x stands for a reference of x
tree.pointset = Pointset(filter(X, tree.cube), &tree)
if tree.depth < h and |tree.pointset|> 1 then
    // if not a leaf node then split
    tree.child ← findSplit(...)
end
else
    tree.sparsity ← tree.cube.logvolume − log(|tree.pointset.X|)
end
return tree
```

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

Input: [$start$, end] - interval for each attribute, $node$ - ref. of containing PIDTree

Output: a $Cube$

```
cube ← {
            node : node
            start : start,
            end : end,
            logvolume : 0                          // log of volume of subcube
            child : EmptyList
        }
for attr in cube.start.keys() do
    // recall that cube.start is a dictionary whose keys are attributes
    cube.logvolume += log(cube.start.end[attr] − cube.start.start[attr])
end
return cube
```

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

Input: X - input data, *node* - to which node this set belongs to

Output: a *Pointset*

```
pointset ← {
            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 [3]
for attr in X.attributes do
    // np.unique is a function from python numpy library
    v, c ← np.unique(X[attr], return_counts=True)
    pointset.val[attr] ← v
    pointset.count[attr] ← c
    pointset.gap[attr] ← [0]
    if |v|> 1 then
        // sum of all elements of pointset.gap for a attr is equal to
        // start[attr] - end[attr]
        g ← 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] ← 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 [7] which is $(1+\epsilon)-$factor approximation of V-optimal histogram construction. Due to complexity of the $findSplit(...)$ function we have not provided its 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).

4.4 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 4.1 isolation forest struggles to give decent results for high dimensional data, on the other hand PIDForest doesn't choose 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.

4.5 Conclusion

PIDForest is arguably one of the best off-the-shelf algorithms for anomaly detection on a large, heterogeneous 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. [3] for more details.

In chapter 2 I discussed some 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 5

Contributions

We concluded previous chapter section 4.5 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.

5.1 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. [8] 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.

5.1.1 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:

$$Regret_T = \sum_{t=1}^{T} f_t(w_t) - \min_{w^* \in S} \sum_{t=1}^{T} f_t(w^*) \tag{5.1}$$

Refer chapter 2 of [9] for more details.

5.1.2 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. [8] 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-dimensional vectors and ii) n-dimensional 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 receive 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. [8]: 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 only with 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:*

$$f_t(w_t) = -y_t SCORE(x_t; w_t) = y_t w_t \cdot \phi(x_t) \tag{5.2}$$

Algorithm 1 of Siqqiqui et al. [8] 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.

5.2 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 the features across the forest in a consistent order. **Define** w be a vector that concatenate all 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 $feedbackITree$

```
1  q ← RandomChoice(X.attributes)
2  p ← RandomNumber(X[splitAttr].min(), X[splittAttr].max())
3  ftree ← Node { left ← None, right ← None,
4                 size ← X.size, splitAttr ← q,
5                 splitVal ← p, w ← 1, θ ← 1} ;          // w,θ for mirror descent
6  if X.size > 1 and X[splitAttr].numUnique() > 1 then
7      X_l ← X.where(q < p)
8      X_r ← X.where(q ≥ p)
9      ftree.left ← feedbackITree(X_l)
10     ftree.right ← feedbackITree(X_r)
11 end
12 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

```
1  if (T.right is None) and (T.left is none) and (e ≥ hlim) then
2      return e          // removed the adjustment, return unadjusted path length
3  end
4  a ← T.splitAttr
   // (e + 1) → e + (weight of the edge)
5  if x[a] < T.splitVal then
6      return PathLength(x, T.left, hlim, e + T.w)
7  end
8  else
9      return PathLength(x, T.right, hlim, e + T.w)
10 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, η - learning rate, y - feedback

```
if (T.right is None) and (T.left is none) and (e ≥ hlim) then
    return                                  // no child node, all weights updated
end
a ← T.splitAttr
if x[a] < T.splitVal then
    nextNode ← T.left
end
else
    nextNode ← T.right
end
// mirror descent update
nextNode.θ ← nextNode.θ − η ∗ y
nextNode.w ← nextNode.θ ∗ (nextNode.θ ≥ 0)
updateWeights(x, &nextNode, hlim, η, y)
```

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

5.3 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 the features across the forest in a consistent order. **Define** w be a vector that concatenate all 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, ϵ - for histogram construction

Output: a $feedbackPIDTree$

$ftree \leftarrow$ {
    child : EmptyList,
    depth : e,
    sparsity : (-1),
    w : (-1)
    θ : (-1)
}
$ftree.cube = Cube(start, end, \&ftree)$
// &x stands for a reference of x
$ftree.pointset = Pointset(filter(X, ftree.cube), \&ftree)$
if $ftree.depth < h$ *and* $|ftree.pointset| > 1$ **then**
    // if not a leaf node then split
    $ftree.child \leftarrow findSplit(...)$
end
else
    $ftree.sparsity \leftarrow ftree.cube.logvolume - log(|ftree.pointset.X|)$
    // w, θ will be used for mirror descent algorithm
    $ftree.\theta \leftarrow ftree.w \leftarrow ftree.sparsity$
end
return $ftree$

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

Complexity: Time - $O(t\psi)$, Space - $O(1)$

Input: x - input instance, T - a $feedbackPIDTree$, η - learning rate, y - feedback

if $|T.child| == 0$ **then**
    // mirror descent update
    $nextNode.\theta \leftarrow nextNode.\theta - \eta * y$
    $nextNode.w \leftarrow nextNode.\theta * (nextNode.\theta \geq 0)$
    return
end
// get the nextNode where instance x falls
$updateWeights(x, \&nextNode, hlim, \eta, y)$

By putting these models in OCO framework our algorithm's accuracy increases. Although for a large amount of streaming data the current algorithm is able to handle little-bit of concept drift but as soon the new data diverges too much in value from the data on which these algorithms are trained there are no leaf nodes on which these new instances will fall. In the next section, I propose a general framework in which this can be handled.

5.4 Online anomaly detection

Consider a streaming dataset $D = [x_1, x_2, \ldots\ldots]$ which increases in size as time progresses. We introduce the timestamp of creation for each tree in the forest. It can be implemented by having a *timestamp* field in the tree object. Then we can get the timestamp of creation for i^{th} tree in the forest by querying $forest[i].timestamp$. Or, we can use *Dictionary* data structure indexed by *timestamp* to implement forest. Either way is fine and left as implementation detail.

For the discussion let's denote the initial forest as: $F = [T_{t=1}, \ldots, T_{t=num\_trees}]$ where t refers to timestamp and let the timestamp be discrete numbers. Let W be the window size. Let $D_t \subseteq D$ defined as $D_t = D[t : t+W] = [x_t, x_{t+1}, \ldots, x_{t+W}]$. Let the model update after each δ time steps. The update can be done in following two ways:

1. Retrain the model after every δ time steps by considering the dataset $D_{i*\delta+1}$ where i is the iteration number.

2. Discard α fraction of oldest timestamped trees from the forest and create only those many new trees by considering the dataset $D_{i*\delta+1}$ where i is the iteration number. Note that setting $\alpha = 1$ is same as the first way.

Both of the above ways allow us to handle the concept drift.

5.5 Handling categorical attributes

Vatsal et al. [3] suggested that to handle the categorical dataset we need some inputs from the domain expert to partition the domain set into subsets such that similar elements are grouped together. For example, given the set of countries we may partition by continents. Hence, this is similar to a clustering problem.

In this section we will use the Word2Vec (ref [10], [11]) method from NLP to find the vector embedding of the categorical features. Here is a brief overview:

Word2Vec is a two-layer neural net that processes text by vectorizing words. Its input is a text corpus and its output is a set of vectors: feature vectors that represent words in that corpus. While Word2vec is not a deep neural network, it turns text into a numerical form that deep neural networks can understand.

```
def cat2vec(df, cat_list, num_bins=10, num_cores=-1, embd_size = 4):
    new_df = pd.DataFrame()

    for col in df.columns:
        if col in cat_list: # categorical data
            new_df[col] = col + '-cat-' + df[col].astype(str)
        else: # non-categorical data
            new_df[col] = col + '-bin-' + pd.cut(df[col], num_bins, labels=False).astype(str)

    sentence_list = new_df.values.tolist()

    if num_cores <= 0:
        num_cores = min(MAX_CORES, multiprocessing.cpu_count())

    model = models.Word2Vec(sentence_list, min_count=1,
                            size=embd_size, workers=num_cores)

    cat2vec_dict = dict()
    for cat in cat_list:
        cat2vec_dict[cat] = dict()
        for item in df[cat].unique():
            cat2vec_dict[cat][item] = model.wv[str(cat)+'-cat-'+str(item)]

    return cat2vec_dict
```

Fig. 5.1: cat2vec implementation
Source: @KishoreKaushal/AnomalyDetection

To illustrate, let assume we have 5 APIs running 7 days a week. Some APIs are used more on weekends while some APIs are used more on weekdays. We can use cat2vec method to find the similarity measure between API-API, DAY-DAY, and API-DAY. Here is a sample example:

```
base ~ Documents > git > AnomalyDetection master
$  python cat2vec.py --input=./data/cat2vec4.pickle --catfeat=a,d --embdsize=3 --numbins=6
'----- Computing Distance matrix -----'
'Embeddings of the feature `a`'
{   1: array([ 0.52483803,  0.4028484 , -1.2577517 ], dtype=float32),
    2: array([ 0.54637855,  0.7770928 , -1.5132232 ], dtype=float32),
    3: array([ 0.7945695,  1.9864086, -1.3179451], dtype=float32),
    4: array([ 0.84155643,  2.0910215 , -1.3373861 ], dtype=float32),
    5: array([ 0.94905573,  2.445837  , -1.6234109 ], dtype=float32)}
'Unique value of feature `a`'
[1, 2, 3, 4, 5]
'Distance matrix:'
array([[0.  , 0.45, 1.61, 1.72, 2.12],
       [0.45, 0.  , 1.25, 1.36, 1.72],
       [1.61, 1.25, 0.  , 0.12, 0.57],
       [1.72, 1.36, 0.12, 0.  , 0.47],
       [2.12, 1.72, 0.57, 0.47, 0.  ]])
'----- Computing Distance matrix -----'
'Embeddings of the feature `d`'
{   1: array([ 0.76067257, -0.90797   , -1.366907  ], dtype=float32),
    2: array([2.1120982 , 0.32404324, 1.9892193 ], dtype=float32),
    3: array([2.1604955 , 0.22678772, 1.9892615 ], dtype=float32),
    4: array([1.9442009 , 0.05397448, 1.6681284 ], dtype=float32),
    5: array([2.0478158 , 0.08926401, 1.8346636 ], dtype=float32),
    6: array([ 1.9491963 , -0.02602539,  1.722808  ], dtype=float32),
    7: array([ 0.64328897, -0.6238332 , -0.9972675 ], dtype=float32)}
'Unique value of feature `d`'
[1, 2, 3, 4, 5, 6, 7]
'Distance matrix:'
array([[0.  , 3.82, 3.81, 3.4 , 3.59, 3.43, 0.48],
       [3.82, 0.  , 0.11, 0.45, 0.29, 0.47, 3.46],
       [3.81, 0.11, 0.  , 0.42, 0.24, 0.42, 3.46],
       [3.4 , 0.45, 0.42, 0.  , 0.2 , 0.1 , 3.04],
       [3.59, 0.29, 0.24, 0.2 , 0.  , 0.19, 3.24],
       [3.43, 0.47, 0.42, 0.1 , 0.19, 0.  , 3.08],
       [0.48, 3.46, 3.46, 3.04, 3.24, 3.08, 0.  ]])
base ~ Documents > git > AnomalyDetection master
$
```

Fig. 5.2: cat2vec sample run
Source: @KishoreKaushal/AnomalyDetection

Figure 4.3 is the distance matrix for the sample problem discussed above. We can use the vector embeddings generated by cat2vec with Isolation Forest and distance matrix for eliminating the need of domain expert to handle the categorical dataset.

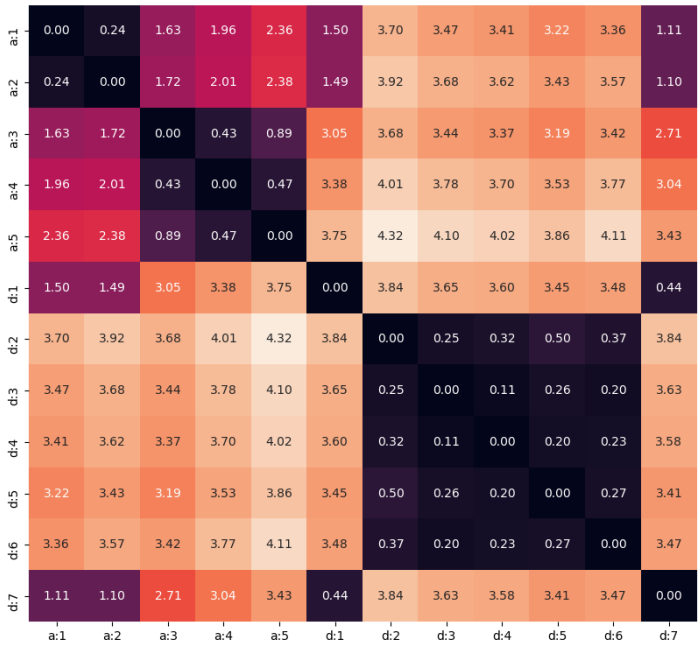


Fig. 5.3: distance matrix
Source: @KishoreKaushal/AnomalyDetection

5.6 Conclusion

This concludes this report on anomaly detection. We discussed two state-of-the-art algorithms. We discussed issues with them and tried to improve upon them on this chapter. You can find the implementations of these methods and algorithms at my repository. I am happy to boast that (at present) my code is the only good implementation of PIDForest and feedback guided anomaly discovery available on the internet.

In the next chapter we will briefly discuss my works on AI in Image Compression, which unfortunately I have to stop in the middle as a consequence of COVID-19 pandemic, because of lack of resources like stable internet and GPU for computation at my home during the lockdown.

Chapter 6

Conclusion and Future Work

References

[1] A. Prakash, N. Moran, S. Garber, A. DiLillo, and J. Storer, "Semantic perceptual image compression using deep convolution networks," 2017.

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

[3] 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

[4] 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

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

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

[7] 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

[8] 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

[9] 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

[10] T. Mikolov, K. Chen, G. S. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013. [Online]. Available: http://arxiv.org/abs/1301.3781

[11] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 3111–3119. [Online]. Available: http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf