Outlier Detection

Group name: PeakyCoders
5 authors, including:

Name	Surname	ID
Yağız	Gezmen	116200043
Kübra	Tahaoğlu	116202068
Burak	Güler	11575012
Devrim	Hot	115200020
Berker	Arslan	117200033

Project Source Code

Project Link-> **Outlier_Detection**

CMPE 211 MIDTERM PROJECT

Istanbul Bilgi University
Istanbul,Turkey

Abstract

Primary goal is io implement two populer outlier detection methods in to java source code.

1 Introduction

Anomalies are data patterns that have different data characteristics compared to normal instances. The detection of anomalies has significant relevance and often provides critical actionable information in various application domains. For example, possible anomalies in launching satellite operations probably causes it to self destruct. Minimal change in weather could make a smart car's system act weirdly which will lower it's performance.

These applications demand anomaly detection algorithms with high detection performance and fast execution.

Most existing model-based approaches to anomaly detection construct a profile of normal instances, then identify instances that do not conform to the normal profile as anomalies. Notable examples such as statistical methods , classification-based methods [1], and clustering-based methods [2] all use this general approach. Two major drawbacks of this approach are: (i) the anomaly detector is optimized to profile normal instances, but not optimized to detect anomalies—as a consequence, the results of anomaly detection might not be as good as expected, causing too many false alarms (having normal instances identified as anomalies) or too few anomalies being detected; (ii) many existing methods are constrained to low dimensional data and small data size because of their high computational complexity.

This paper proposes two ways of detection methods which are:

(I)Statistical outlier detection, analysis the data for abstract anomalies which are +-3 sigma away from local mean.

(II)Isolation Forest:

The proposed method, called Isolation Forest or iForest, builds an ensemble of iTrees for a given data set, then anomalies are those instances which have short average path lengths on the iTrees. There are only two variables in this method: the number of trees to build and the sub-sampling size. We show that iForest's detection performance converges quickly with a very small number of trees, and it only requires a small sub-sampling size to achieve high detection performance with high efficiency.

Apart from the key difference of isolation versus profiling, iForest is distinguished from existing model-based , distance-based and density-based methods in the follow ways:

- The isolation characteristic of iTrees enables them to build partial models and exploit sub-sampling to an extent that is not feasible in existing methods. Since a large part of an iTree that isolates normal points is not needed for anomaly detection; it does not need to be constructed. A small sample size produces better iTrees because the swamping and masking effects are reduced.

- iForest utilizes no distance or density measures to detect anomalies. This eliminates major computational cost of distance calculation in all distance-based methods and density-based methods.

- iForest has a linear time complexity with a low constant and a low memory requirement. To our best knowledge, the best-performing existing method achieves only approximate linear time complexity with high memory usage .

- iForest has the capacity to scale up to handle extremely large data size and high-dimensional problems with a large number of irrelevant attributes.

This paper is organised as follows: In Section 2 we describe how did we apply statistical methods with used algorithms. In Section 3, we describe Hand-made Isolation forest method with used algorithms. Section 4, we describe which and why we used specific data-structures. Section 5 provides results of given data analysis with both methods. Section 6 provides a discussion on unexpected difficulties, how could we improvise our results and what could be done for feature work. Section 7 concludes this paper.

2 Statistical Detection

Statistical detection method is a simple model based application which takes a data, calculate it's moving average, calculates local variance due to local mean and applies three-sigma rule to locate anomalies.

Algorithm 1 : *Statistical*(X)

Inputs: X - input data
Output: a set of anomalies

1: read data and store it in a array->X'=read(X)
2: calculate moving average and store it in a array->Y=movingAvg(X')
3:calculate local variance and store it in a array->Y'=localVariance(X',Y)
4:Store anomalyies $A \leftarrow threeSigma(Y',X')$
5: return A

Algorithm 2 : *threeSigma (Y,X)*

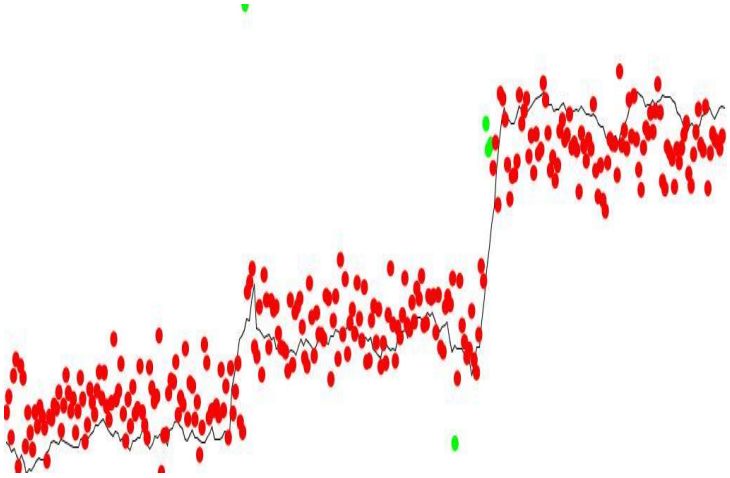
Inputs: X - input data,Y-local variance
Output: a set of anomalies

1: Calculate Standard Dev ->sd = StdDev(Y)
2: Calculate mean ->mean = mean(Y)
3: for i_0 to Y.size
4: calculate variance -> v = |Y_i -mean|
5: apply 3 sigma if v>=sd*3
6: store anomalies A.add(v)
7:end if
8: end for
9:return A

Algorithm 3:*localVariance(X,Y)*

Inputs: X - input data,Y-local mean
Output: a set of anomalies

1: init range->lim,low = X.size - Y.size,0
2: initilaze array for LVariance->localV
3: for low to Y.size
4: $locaV_{low} = X_{low}-Y_i$
5: if low>lim
6: i+1
7:end if
8: end for
9:return locaV



2.1 Analysis results on a given data

Figure 1. Green dots are anomalyies in a given set,line inside points is the moving average of the data set

We also use a method for normalization points to suitable them drawing results.

$$Normalize = \frac{data_i\text{-}mean(data)}{max(data)\text{-}min(data)}$$

Where data(i) is a point in a data set.

3 Hand-made Isolation Forest

This section describes the hand made isolation forest method with algorithms. Hand-Made isolation forest works pretty similar to original Isolation Forest algorithm. The goal is to sub-samplize the given data set to 2^n size which will improvise performance.

After that it creates a terrain with calling isolation tree 't' times. Each isolation tree returns an index of isolation and stores path length in an array .Each return we took the index of isolated index with path length array and store it in a symbol table which is #Name->index #Values->path length array. After the process is over, we take our symbol table and use it to average path lengths of existing index's, and take 3 minimum elements from our new Symbol table which are highly possible to be an anomaly and returns that**.**

n –> 256 by default

t -> 100 by default

3.1 Training Stage Hand-Made Isloation Forest

There are three input parameters to the Hand-Made iForest algorithm. They are data sets, the sub-sampling size 'ψ' and the number of trees 't'. We provided a guide below to select a suitable value for each of the three parameters.

Sub-sampling size 'ψ' controls the training data size. We find that when 'ψ' increases to a desired value, iForest detects reliably and there is no need to increase 'ψ' further because it increases processing time and memory size without any gain in the detection performance. The complexity of the training of an iForest is $O(t\psi \log\psi)$.

Algorithm 1 : $iForest(X,t,\psi)$

Inputs: X - input data, t - number of trees, ψ - sub-sampling size Output: a index

1: Initialize $Forest$
2: set height limit $l = ceiling(\log_2 \psi)$
3: for $i = 1$ to t do
4: $X^0 \leftarrow sample(X,\psi)$
5: $Forest \leftarrow Forest \cup iTree(X^0,0,l)$
6: end for
7: return $Forest$

Algorithm 2 : $rTree\ (X,e,l\ )$

Inputs: X - input data, e - current tree height, l - height limit Output: an iTree

1: if $e \geq l$
2: return -1
3: $|X| \leq 1$ then
return index
4: let Q be a list of attributes in X
5: randomly select an attribute $q \in Q$
6: randomly select a split point p from max and min values of attribute q in X
7: pathlength + 1
8: $Xl \leftarrow filter(X,q<p)$
9: *if index!=-1 : return index*
10: $Xr \leftarrow filter(X,q \geq p)$
11: *if index!=-1 : return index*
12: *return -1*

4 Used Data Structures

This section describes reasons of used structures

We used :

(i) arrays -> for store specific ranged items.

(ii)ArrayList -> for store arrays and create dinamic arrays.

(iii)Streams -> for apply specific algorithms to given set's all items.

(iv)Stacks -> for create non index required item sets.

(v)BST-> for create symbol tabels .

(vi)algs4jar lib->

(i)StdStats.class->

(i)max,min->for find max,min values of an array.

(ii)stddev -> for calculate standart deviation of an given set.

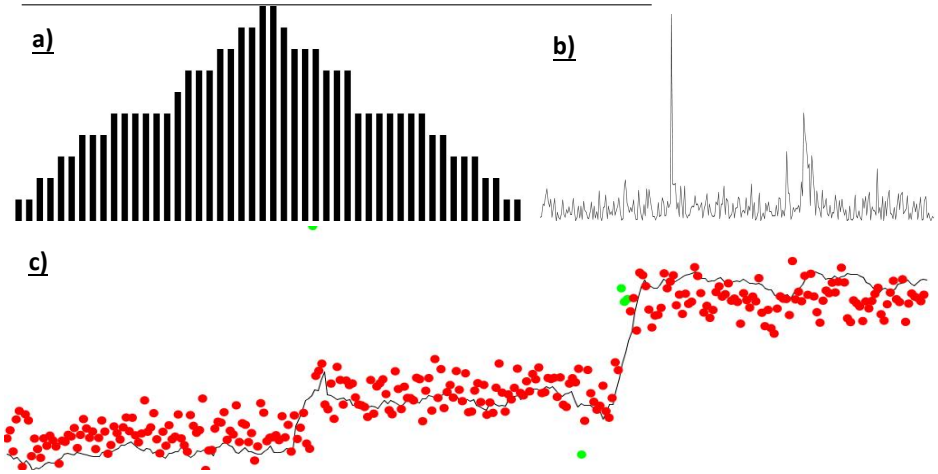
(iii)mean -> for calculate mean of a given set.

(iv)plotPoints,plotBars,plotLines ->for draw results

5 Results

5 .1 Statistic Results

This sections show the image results generated by our Statistical method.



a) **This figure represents histogram graph of the given data set.**

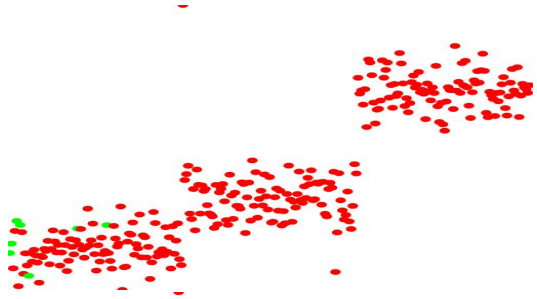
b) **This figure represents differance from local mean of data set's each data point.**

c)**This figure represents analysis of given data set.**

Green dots represents anomalies,Red dots represents normal points,line which is passing inside of red dots is local mean of data set.

5 .2 Isolation Forest Resulst

This sections show the image results generate by our Isolation forest method.



a) **This figure represents results for sub sample size->256**

b) **This figure represents results for sub sample size->64**

6 Discussion

This section we discuss the unexpected difficulties about project phases, what could be done to improvise results and what could be done for future outlier detection.

6.1 Unexpected Difficulities

We thought most difficult part would be drawing images because java's gui isn't efficient but thanks to algs4jar libary it was quite easy to draw results in same canvas area. Because of that, the only difficult part of the statistical method was calculating three sigma for data set. Isolation forest on the other hand, we didn't think it would be that hard to imagine and apply to java. Because of its multi dimensional system it took time to understand its processing principle.

6.2 Improvise Results

We can use AUC[Appendix:A] calculation to test accuracy of our methods.

6.3 Possible Future Technique

As you know,existing outlier detection algorithms were suffering from bigger data sets when they needed to be analysed in near constant time because limited gpu memory isn't enough to process dynamically with real time data according to existing client's data sets. So we thought about a simple question's answer. What if we combine systems of the clients to analyze their generated data sets. There are existing examples for combined systems like bitcoin chains systems or NASA's flight operator systems. So maybe in the near future we can combine something like iForest algorithm with internet of things technology to particularize given multi dimensional data sets to each clients system and after that create a two dimensional set which will process accordingly: each line is analysis of a column and then it will gather them in a main operator which will make final analysis for our last set of data and store them with ID's of bigger data sets with a map location. For example with this kind of a system, we could be able to analyze real time global stock market's data sets in near linear time.

7 Conclussion

This paper proposes description of two popular methods for anomaly detections and provides techniques with simple algorithms to test them with basic programming knowledge. Also shows an example with details which was written in java language.

Appendix

A. AUC CALCULATION FOR ANOMALY DETECTION

AUC is known as the area under receiver operating characteristic curve. In data mining, it is usually used to measure the overall performance of classifiers regardless of the threshold between the true positives and true negatives. In the context of anomaly detection, anomalies are treated as the positive class. To calculate AUC, a simple approach adopted from [Hand and Till 2001] is as follows:

Algorithm 4 : AUC

1: let n_a be the number of true anomalies.
2: let n_n be the number of true normal points.
3: rank all instances according to their anomaly scores in descending order.
4: let S be the sum of rankings of the actual anomalies, $S = \sum_{i=1}^{n_a} r_i$, where r_i is the rank of the i^{th} anomaly in the ranked list.
5: $AUC = \dfrac{S - (n_a^2 + n_a)/2}{n_a n_n}$

Referances

[1] *Large Data Bases*, pages 392–403, San Francisco, CA, USA, 1998. Morgan Kaufmann.

[2] D. M. Rocke and D. L. Woodruff. Identification of outliers in multivariate data. *Journal of the American Statistical Association*, 91(435):1047–1061, 1996.