Outlier Detection Algorithm Suite in MATLAB

Project Report submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology

in

Computer Science & Engineering

under the Supervision of

Dr. Sakshi Babbar

By

Shaunik Seth

Roll No.: 111223

to

Jaypee University of Information and Technology

Waknaghat, Solan – 173234, Himachal Pradesh

Certificate

This is to certify that project report entitled “Outlier Detection Algorithm Suite in MATLAB”, submitted by Shaunik Seth(111223) in partial fulfillment for the award of degree of Bachelor of Technology in Computer Science And Engineering to Jaypee University of Information Technology , Waknaghat, Solan has been carried out under my supervision.

This work has not been submitted partially or fully to any other University or Institute for the award of this or any other degree or diploma.

Date:

Dr. Sakshi Babbbar

Assistant Professor(Senior Grade)

Acknowledgement

I would like to express my gratitude to all those who gave me the possibility to complete this project. I want to thank the Department of CSE & IT in JUIT for giving me the permission to commence this project in the first instance, to do the necessary research work.

I am deeply indebted to my project guide Dr. Sakshi Babbar, whose help, stimulating suggestions and encouragement helped me in all the time of research on this project. I feel motivated and encouraged every time I get her encouragement. For her coherent guidance throughout the tenure of the project, I am very thankful. I feel fortunate to be taught by her.

I am also grateful to **Mr. Amit Singh(CSE Project lab**) for his practical help and guidance.

Date: Name of the student

Table of Contents

CH No.	Topic	Page No
1.	Introduction	1
1.1	Challenges	3
1.2	Contribution	4
2.	Introduction to outliers	5
2.1	What are outliers?	5
2.2	Applications	6
2.3	Types of outliers	7
2.4	Types of outlier detection techniques	11
3.	Outlier detection algorithms	14
3.1	Univariate Outlier Detection using Box Plot	14
3.2	Multivariate Outlier Detection using Mahalanobis Distance Measure	15
3.3	Mining distance based outliers using K nearest neighbor criteria (Euclidean distance)	16
3.4	Mining distance based outliers in near linear time using randomization	16
3.5	Identifying density based local outliers- local outlier factor	17
3.6	Conditional Anomaly Detection	19
3.7	Bayesian Network based outlier detection	21
4.	Implementation	24
4.1	Codes	24
4.2	Dataset	42
4.3	Result	43
5.	Future work and Conclusion	51
6.	References	52

List of Figures

S.No	Title	Page No.
1.	A simple example of anomalies in a two-dimensional data set	5
2.	Contextual anomaly t_2 in a temperature time series. Note that the temperature at time t_1 is same as that at time t_2 but occurs in a different context and hence is not considered an anomaly	9
3.	Collective anomaly corresponding to an Atrial Premature Contraction in a human electro--cardiogram output	10
4.	A box plot for a univariate data set	14
5.	Graph of various runtime complexities	17
6.	Advantages of local density based techniques over global density based techniques	18
7.	Difference between the neighborhoods computed by Local Outlier Factor and Connectivity based Outlier Factor	19
8.	Syndromic Surveillance Application	20
9.	Bayesian Network Example	22
10.	Gaussian Distribution graph with mean μ and standard deviation σ	23
11.	Depicts the Graphical User Interface for the suite	43
12.	Result obtained by running Box Plot algorithm on Iris plant dataset in MATLAB	43
13.	Red points on the figure depict outliers in Iris Plant dataset obtained using box plot	44
14.	Result obtained by running Mahalanobis distance algorithm on Iris plant dataset in MATLAB	44
15.	Plot depicting normal points in green and outliers in red of Iris Plant dataset obtained by using Mahalanobis Distance algorithm	45
16.	Result obtained by running Euclidean distance algorithm on Iris plant dataset in MATLAB	45
17.	Plot depicting normal points in green and outliers in red of Iris Plant dataset obtained by using Euclidean Distance algorithm	46
18.	Result obtained by running Randomization & Pruning algorithm on Iris plant dataset in MATLAB	46

19.	Plot depicting normal points in green and outliers in red of Iris Plant dataset obtained by using Randomization & Pruning algorithm	47
20.	Bayesian Network for Effects of Smoking example generated in MATLAB	47
21.	Result obtained by running Bayesian network based outlier detection algorithm for discrete nodes in MATLAB	48
22.	Result obtained by running Bayesian network based outlier detection algorithm for discrete and continuous nodes in MATLAB	49
23.	Bayesian Network for Iris Plant data set example generated in MATLAB	50

Abstract

For many applications, data mining systems are required to detect anomalous (abnormal or unexpected) observations. This has so far proven to be a difficult challenge because anomalies are usually considered to be "non-normal" observations, where "normality" is typically defined by very complex concepts. Because of these and other reasons, there are no standard and principled approaches for outlier detection. Outlier detection is an important problem that has been researched within diverse research areas and application domains. Many outlier detection techniques have been specifically developed for certain application domains, while others are more generic.

Although outliers are often considered as an error or noise, they may carry important information. Detected outliers are candidates for aberrant data that may adversely lead to model misspecification and incorrect results.

The aim of this project is to implement various well known outlier detection algorithms for use by people dealing in data mining as well as for general purposes. Different approaches for outlier detection namely Statistical based approach, Distance based approach and Conditional Anomaly Detection are covered under this project. Various algorithms are implemented namely Univariate outlier detection using Boxplot, Multivariate outlier detection using Mahalanobis Distance Measure, Multivariate outlier detection using Euclidean Distance Measure, Multivariate outlier detection using Randomization and Pruning, Bayesian Network based outlier detection for discrete nodes using conditional probability tables and Bayesian Network based outlier detection for discrete and continuous nodes using parameter learning. The project also includes a graphical user interface for the suite in MATLAB. In addition to above, the recall for above stated algorithms has been calculated.

CHAPTER 1
INTRODUCTION

1. Introduction

Today, technology is changing the way people produce and handle information. From business to science and engineering, there has been a massive proliferation of huge databases storing valuable information. The opportunities of an effective use of these new data sources, Gigabytes up to Terabytes, are enormous, however, traditional data analysis techniques have not kept in pace with the type of processing needed by the new volumes of data.

The huge size and dimensionality of current large databases require new ideas to scale up current statistical and computational approaches. This is especially critical when the system needs to interact with a human expert, as it is usually the case. If a data analysis tool takes weeks or months to return a result, the interaction between the analyst and the data is severely limited. In this paper we present various algorithms for detection of anomalous records in a large database.

Outlier detection refers to the problem of finding patterns in data that do not conform to expected behaviour. These nonconforming patterns are often referred to as anomalies, outliers, discordant observations, exceptions, aberrations, surprises, peculiarities, or contaminants in different application domains. Of these, anomalies and outliers are two terms used most commonly in the context of anomaly detection; sometimes interchangeably. Outlier detection finds extensive use in a wide variety of applications such as fraud detection for credit cards, insurance, or health care, intrusion detection for cyber-security, fault detection in safety critical systems, and military surveillance for enemy activities.

The importance of anomaly detection is due to the fact that anomalies in data translate to significant, and often critical, actionable information in a wide variety of application domains. For example, an anomalous traffic pattern in a computer network could mean that a hacked computer is sending out sensitive data to an unauthorized destination. An anomalous MRI image may indicate the presence of malignant tumors. Anomalies in credit card transaction data could indicate credit card or identity theft, or anomalous readings from a space craft sensor could signify a fault in some component of the space craft .

Outlier detection has a long history in statistics, but has largely focussed on data that is univariate, and data with a known (or parametric) distribution. These two limitations have restricted the ability to apply these types of methods to large real-world databases which typically have many different fields and have no easy way of characterizing the multivariate distribution of examples. Other researchers have taken a non-parametric approach and proposed using an example's distance to its nearest neighbours as a measure of unusualness. Although distance is an effective non-parametric approach to detecting outliers, the drawback is the amount of computation

time required. Straightforward algorithms, such as hose based on nested loops, typically require O(N2) distance computations. This quadratic scaling means that it will be very difficult to mine outliers as we tackle increasingly larger data sets. This is a major problem for many real databases where there are often millions of records. Recently, researchers have presented many different algorithms for efficiently finding distance-based outliers. These approaches vary from spatial indexing trees to partitioning of the feature space with clustering algorithms. The common goal is developing algorithms that scale to large real data sets.

Anomaly detection has been an active area of computer science research for a very long time. Applications include medical informatics, computer vision, computer security, sensor networks, general-purpose data analysis and mining, and many other areas. However, in contrast to problems in supervised learning where studies of classification accuracy are the norm, little research has systematically addressed the issue of accuracy in general-purpose unsupervised anomaly detection methods. Papers have suggested many alternate problem definitions that are designed to boost the chances of finding anomalies, but there been few systematic attempts to maintain high coverage at the same time that false positives are kept to a minimum. Accuracy in unsupervised anomaly detection is important because if used as a data mining or data analysis tool, an unsupervised anomaly detection methodology will be given a "budget" of a certain number of data points that it may call anomalies.

In most realistic scenarios, a human being must investigate candidate anomalies reported by an automatic system, and usually has a limited capacity to do so. This naturally limits the number of candidate anomalies that a detection methodology may usefully produce. Given that this number is likely to be small, it is important that most of those candidate anomalies are interesting to the end user. This is especially true in applications where anomaly detection software is used to monitor incoming data in order to report anomalies in real time. When such events are detected, an alarm is sounded that requires immediate human investigation and response. Unlike the offline case where the cost and frustration associated with human involvement can usually be amortized over multiple alarms in each batch of data, each false alarm in the online case will likely result in an additional notification of a human expert, and the cost cannot be amortized.

Detecting outliers or anomalies in data has been studied in the statistics community as early as the 19th century. Over time, a variety of anomaly detection techniques have been developed in several research communities. Many of these techniques have been specifically developed for certain application domains, while others are more generic.

I begin with univariate anomaly detection and then move to multivariate where we use the distance based approach for anomaly detection with complexity $O(n^2)$, then reduce the complexity to O(nlogn) and after that head towards conditions involved in anomaly detection through the use of Bayesian Networks. It further involves two cases, first for discrete nodes using conditional probability table and second for both discrete and continuous nodes using parameter learning where continuous nodes are represented using Gaussian distribution. The algorithms that have been implemented in this project include:

1. Univariate Outlier Detection using Box Plot. [7]
2. Multivariate Outlier Detection using Mahalanobis Distance Measure. [2]
3. Mining distance based outliers using K nearest neighbor criteria (quadratic complexity). [2]
4. Mining distance based outliers in near linear time using randomization (near linear complexity). [3]
5. Bayesian Network based Outlier Detection for discrete nodes using conditional probability table. [6]
6. Bayesian Network based Outlier Detection using parameter learning. [6]

1.1 Challenges

At an abstract level, an anomaly is defined as a pattern that does not conform to expected normal behaviour. A straightforward anomaly detection approach, therefore, is to define a region representing normal behaviour and declare any observation in the data that does not belong to this normal region as an anomaly. But several factors make this apparently simple approach very challenging:

1. Defining a normal region that encompasses every possible normal behaviour is very difficult. In addition, the boundary between normal and anomalous behaviour is often not precise. Thus an anomalous observation that lies close to the boundary can be difficult to predict.

2. When anomalies are the result of malicious actions, the malicious adversaries often adapt themselves to make the outlier observations appear normal, thereby making the task of defining normal behaviour more difficult.

3. In many domains normal behaviour keeps evolving and a current notion of normal behaviour might not be sufficiently representative in the future.

4. The exact notion of an anomaly is different for different application domains. For example, in the medical domain a small deviation from normal (e.g., fluctuations in body temperature) might be an anomaly, while similar deviation in the stock market domain (e.g., fluctuations in the value of a stock) might be considered as normal. Thus applying a technique developed in one domain to another, is not straightforward.

5. Availability of labelled data for training/validation of models used by anomaly detection techniques is usually a major issue.

6. Often the data contains noise that tends to be similar to the actual anomalies and hence is difficult to distinguish and remove.[1]

Due to these challenges, the anomaly detection problem, in its most general form, is not easy to solve. In fact, most of the existing anomaly detection techniques solve a specific formulation of the problem. The formulation is induced by various factors such as the nature of the data, availability of labelled data, type of anomalies to be detected, and so on. Often, these factors are determined by the application domain in

which the anomalies need to be detected. Researchers have adopted concepts from diverse disciplines such as statistics, machine learning, data mining, information theory, spectral theory, and have applied them to specific problem formulations.

1.2 Contribution

This project is an attempt to provide a structured and broad overview of extensive research on anomaly detection techniques spanning multiple research areas and application domains.

1.Implemented outlier detection based on Statistical approach.

2.Algorithmic development of univariate outlier detection using box plot.

3.Algorithmic implementation of multivariate outlier detection using Mahalanobis Distance Measure.

4.Implemented outlier detection techniques on Distance based approach.

5.Implemented algorithm for detecting distance based outliers using K nearest neighbour approach criteria (Euclidean Distance).

6.Applied randomization and pruning technique to reduce the complexity of detecting distance based outliers to O(nlogn) from $O(n^2)$.

7.Implemented Conditional Anomaly Detection with focus on Bayesian Networks.

8.Implemented algorithm for outlier detection in Bayesian Networks of discrete nodes using conditional probability tables.

9.Implemented algorithm for outlier detection in Bayesian Networks comprising of both discrete and continuous nodes using parameter learning.

10.Calculated recall for various outlier detection techniques.

11.Developed visualization for the outliers detected through various detection algorithms.

12.Developed a graphical user interface for the suite in MATLAB.

CHAPTER 2
INTRODUCTION TO OUTLIERS

2.1 What are Outliers?

Outliers are patterns in data that do not conform to a well defined notion of normal behaviour. Figure 1 illustrates anomalies in a simple two-dimensional data set. The data has two normal regions, N_1 and N_2, since most observations lie in these two regions. Points that are sufficiently far away from these regions, for example, points O_1 and O_2, and points in region O_3, are anomalies[2]

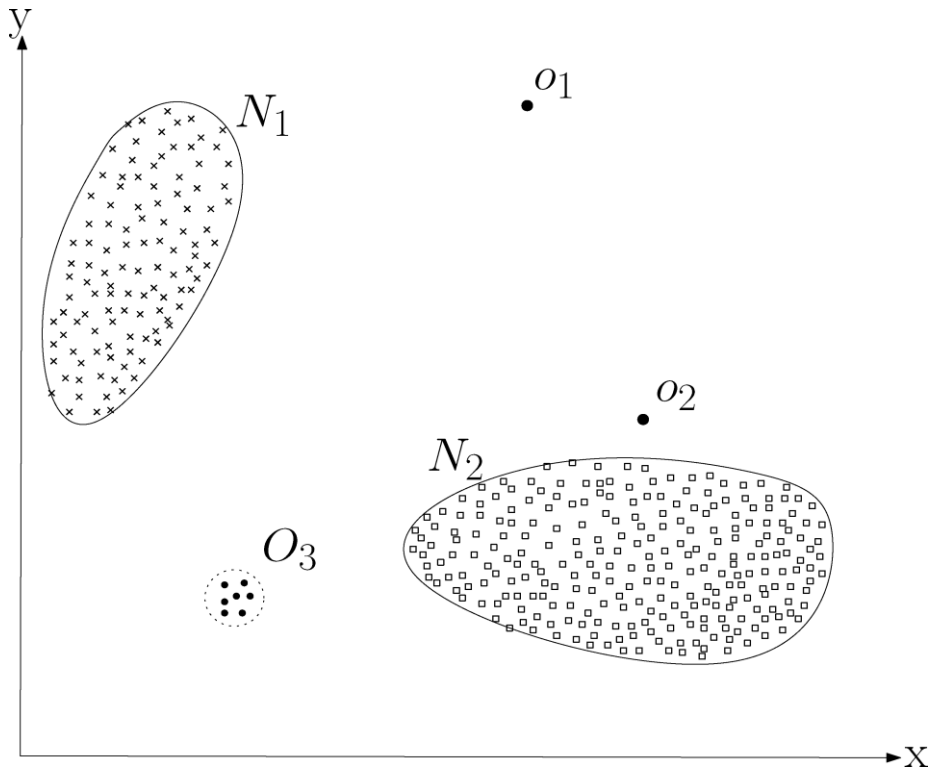


Fig. 1. A simple example of anomalies in a two-dimensional data set.[2]

Anomalies might be induced in the data for a variety of reasons, such as malicious activity, for example, credit card fraud, cyber-intrusion, terrorist activity or breakdown of a system, but all of the reasons have the common characteristic that they are interesting to the analyst. The interestingness or real life relevance of anomalies is a key feature of anomaly detection.

Anomaly detection is related to, but distinct from noise removal and noise accommodation , both of which deal with unwanted noise in the data. Noise can be defined as a phenomenon in data that is not of interest to the analyst, but acts as a hindrance to data analysis. Noise removal is driven by the need to remove the unwanted objects before any data analysis is performed. Noise accommodation refers to immunizing a statistical model estimation against anomalous observations.

Another topic related to anomaly detection is novelty detection, which aims at detecting previously unobserved (emergent, novel) patterns in the data, for example, a new topic of discussion in a news group. The distinction between novel patterns and anomalies is that the novel patterns are typically incorporated into the normal model after being detected.

2.2 Applications[1]

1. Intrusion Detection

Intrusion detection refers to detection of malicious activity (break-ins, penetrations, and other forms of computer abuse) in a computer related system. These malicious activities or intrusions are interesting from a computer security perspective. An intrusion is different from the normal behaviour of the system, and hence anomaly detection techniques are applicable in intrusion detection domain.

The key challenge for anomaly detection in this domain is the huge volume of data. The anomaly detection techniques need to be computationally efficient to handle these large sized inputs. Moreover the data typically comes in a streaming fashion, thereby requiring online analysis. Another issue that arises because of the large sized input is the false alarm rate. Since the data amounts to millions of data objects, a few percent of false alarms can make analysis overwhelming for an analyst. Labelled data corresponding to normal behaviour is usually available, while labels for intrusions are not. Thus, semi-supervised and unsupervised anomaly detection techniques are preferred in this domain.

2. Fraud Detection

Fraud detection refers to detection of criminal activities occurring in commercial organizations such as banks, credit card companies, insurance agencies, cell phone companies, stock market, and so on. The malicious users might be the actual customers of the organization or might be posing as customers (also known as *identity theft*). The fraud occurs when these users consume the resources provided by the organization in an unauthorized way. The organizations are interested in immediate detection of such frauds to prevent economic losses.

Fawcett and Provost introduced the term *activity monitoring* as a general approach to fraud detection in these domains. The typical approach of anomaly detection techniques is to maintain a usage profile for each customer and monitor the profiles to detect any deviations.

3. Medical and Public Health Anomaly Detection

Anomaly detection in the medical and public health domains typically works with patient records. The data can have anomalies due to several reasons, such as abnormal patient condition, instrumentation errors, or recording errors. Several techniques have also focussed on detecting disease outbreaks in a specific area. Thus the anomaly detection is a very critical problem in this domain and requires a high degree of accuracy.

The data typically consists of records that may have several different types of features, such as patient age, blood group, and weight. The data might also have a temporal as well as spatial aspect to it. Most of the current anomaly detection techniques in this domain aim at detecting anomalous records (point anomalies). Typically the labelled data belongs to the healthy patients, hence most of the techniques adopt a semi-supervised approach. Another form of data handled by anomaly detection techniques in this domain is time-series data, such as *Electrocardiograms* (ECG) and *Electroencephalograms* (EEG). Collective anomaly detection techniques have been applied to detect anomalies in such data.

The most challenging aspect of the anomaly detection problem in this domain is that the cost of an anomaly being undetected can be very high.

4. Industrial Damage Detection

Industrial units suffer damage due to continuous usage and normal wear and tear. Such damage needs to be detected early to prevent further escalation and losses. The data in this domain is usually referred to as sensor data because it is recorded using different sensors and collected for analysis. Anomaly detection techniques have been extensively applied in this domain to detect such damage. Industrial damage detection can be further classified into two domains, one that deals with defects in mechanical components such as motors, engines, and so on, and the other that deals with defects in physical structures.

5. Financial Applications

Based on the past records of the client ,his past bank balances, financial transactions and assets in hand, the probability of his repaying the loan can be determined and consequently interest rates that are applicable to him can be calculated. Here the outliers would indicate the potential clients who have higher probability of not repaying the loan.

2.3 Type of Outliers[1]

An important aspect of an outlier detection technique is the nature of the desired anomaly. Outliers can be classified into following three categories:

2.3.1 Point Outliers

If an individual data instance can be considered as anomalous with respect to the rest of data, then the instance is termed a point anomaly. This is the simplest type of anomaly and is the focus of majority of research on anomaly detection.

For example, in Figure 1, points O_1 and O_2, as well as points in region O_3, lie outside the boundary of the normal regions, and hence are point anomalies since they are different from normal data points.

As a real-life example, consider credit card fraud detection. Let the data set correspond to an individual's credit card transactions. For the sake of simplicity, let us assume that the data is defined using only one feature: amount spent. A transaction for which the amount spent is very high compared to the normal range of expenditure for that person will be a point anomaly.

2.3.2 Contextual Outliers

If a data instance is anomalous in a specific context, but not otherwise, then it is termed a contextual anomaly (also referred to as conditional anomaly).

The notion of a context is induced by the structure in the data set and has to be specified as a part of the problem formulation. Each data instance is defined using the following two sets of attributes:

(1) Contextual attributes. The contextual attributes are used to determine the context (or neighbourhood) for that instance. For example, in spatial data sets, the longitude and latitude of a location are the contextual attributes. In time-series data, time is a contextual attribute that determines the position of an instance on the entire sequence.
(2) Behavioural attributes. The behavioural attributes define the non-contextual characteristics of an instance. For example, in a spatial data set describing the average rainfall of the entire world, the amount of rainfall at any location is a behavioural attribute.

The anomalous behaviour is determined using the values for the behavioural attributes within a specific context. A data instance might be a contextual anomaly in a given context, but an identical data instance (in terms of behavioural attributes) could be considered normal in a different context. This property is key in identifying contextual and behavioural attributes for a contextual anomaly detection technique. Contextual anomalies have been most commonly explored in time-series data and spatial data . Figure 3 shows one such example for a temperature time-series that shows the monthly temperature of an area over the last few years. A temperature of 35◦F might be normal during the winter (at time $t1$) at that place, but the same value during the summer (at time $t2$) would be an anomaly.

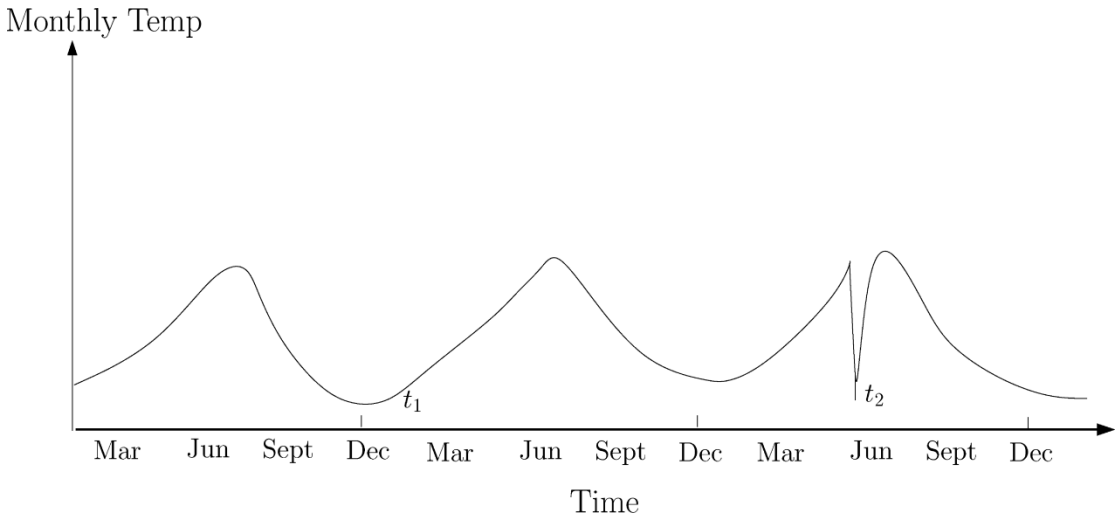


Fig. 2. Contextual anomaly *t*2 in a temperature time-series. Note that the temperature at time *t*1 is same as that at time *t*2 but occurs in a different context and hence is not considered as an anomaly.

A similar example can be found in the credit card fraud detection domain. A contextual attribute in the credit card domain can be the time of purchase. Suppose an individual usually has a weekly shopping bill of $100 except during the Christmas week, when it reaches $1000. A new purchase of $1000 in a week in July will be considered a contextual anomaly, since it does not conform to the normal behaviour of the individual in the context of time even though the same amount spent during Christmas week will be considered normal.

The choice of applying a contextual anomaly detection technique is determined by the meaningfulness of the contextual anomalies in the target application domain. Another key factor is the availability of contextual attributes. In several cases defining a context is straightforward, and hence applying a contextual anomaly detection technique makes sense. In other cases, defining a context is not easy, making it difficult to apply such techniques.

2.3.3 Collective Outliers

If a collection of related data instances is anomalous with respect to the entire data set, it is termed a collective anomaly. The individual data instances in a collective anomaly may not be anomalies by themselves, but their occurrence together as a collection is anomalous. Figure 4 is an example that shows a human electrocardiogram output [Goldberger et al. 2000]. The highlighted region denotes an anomaly because the same low value exists for an abnormally long time (corresponding to an *Atrial Premature Contraction*). Note that that low value by itself is not an anomaly.

As an another illustrative example, consider a sequence of actions occurring in a computer as shown below:

. . . http-web, buffer-overflow, http-web, http-web, smtp-mail, ftp, http-web, ssh, smtp-mail, http-web, **ssh**, **buffer-overflow**, **ftp**, http-web, ftp, smtp-mail,http-web

The highlighted sequence of events (**buffer-overflow**, **ssh**, **ftp**) correspond to a typical Web-based attack by a remote machine followed by copying of data from the host computer to a remote destination via *ftp*. It should be noted that this collection of events is an anomaly, but the individual events are not anomalies when they occur in other locations in the sequence.

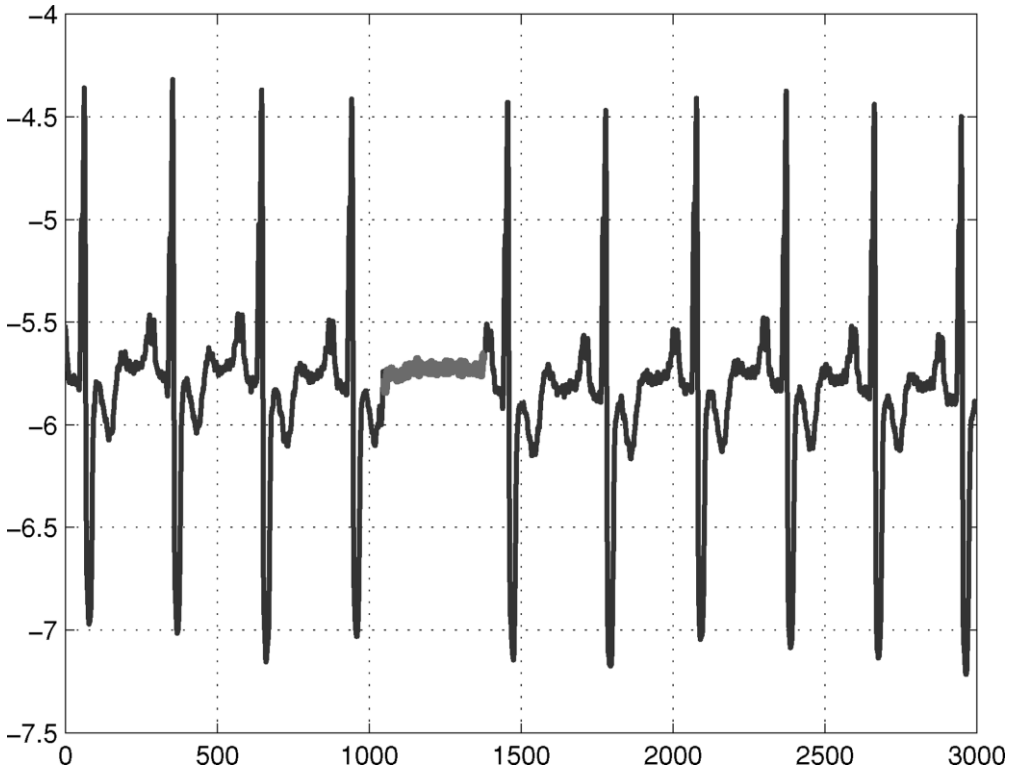


Fig. 3. Collective anomaly corresponding to an *Atrial Premature Contraction* in a human electrocardiogram output.

Collective anomalies have been explored for sequence data, graph data, and spatial data. It should be noted that while point anomalies can occur in any data set, collective anomalies can occur only in data sets in which data instances are related. In contrast, occurrence of contextual anomalies depends on the availability of context attributes in the data. A point anomaly or a collective anomaly can also be a contextual anomaly if analyzed with respect to a context. Thus a point anomaly detection problem or collective anomaly detection problem can be transformed to a contextual anomaly detection problem by incorporating the context information.

The techniques used for detecting collective anomalies are very different than the point and contextual anomaly detection techniques, and require a separate detailed discussion.

2.4 Types of Outlier Detection Techniques

The basic techniques for outlier detection are:

2.4.1 Statistical Based Outlier Detection

The underlying principle of any statistical anomaly detection technique is: "An anomaly is an observation which is suspected of being partially or wholly irrelevant because it is not generated by the stochastic model assumed". Statistical anomaly detection techniques are based on the following key assumption:

Assumption: Normal data instances occur in high probability regions of a stochastic model, while anomalies occur in the low probability regions of the stochastic model.
Statistical techniques fit a statistical model (usually for normal behaviour) to the given data and then apply a statistical inference test to determine if an unseen instance belongs to this model or not. Instances that have a low probability of being generated from the learned model, based on the applied test statistic, are declared as anomalies. Both parametric as well as nonparametric techniques have been applied to fit a statistical model. While parametric techniques assume the knowledge of the underlying distribution and estimate the parameters from the given data, nonparametric techniques do not generally assume knowledge of the underlying distribution.

2.4.2 Distance Based Outlier Detection

The concept of nearest neighbour analysis has been used in several anomaly detection techniques. Such techniques are based on the following key assumption:

Assumption. Normal data instances occur in dense neighbourhoods, while anomalies occur far from their closest neighbours.

Nearest neighbour-based anomaly detection techniques require a distance or similarity measure defined between two data instances. Distance (or similarity) between two data instances can be computed in different ways. For continuous attributes, Euclidean distance is a popular choice, but other measures can be used. For categorical attributes, a simple matching coefficient is often used but more complex distance measures can also be used. For multivariate data instances, distance or similarity is usually computed for each attribute and then combined.

Nearest neighbour-based anomaly detection techniques can be broadly grouped into two categories:
(1) techniques that use the distance of a data instance to its *kth* nearest neighbour as the anomaly score;
(2) techniques that compute the relative density of each data instance to compute its anomaly score.

A basic nearest neighbour anomaly detection technique is based on the following definition: The anomaly score of a data instance is defined as its distance to its kth nearest neighbour in a given data set. This basic technique has been applied to detect land mines from satellite ground images and to detect shorted turns (anomalies) in the DC field windings of large synchronous turbine-generators. In the latter paper the authors use k = 1. Usually, a threshold is then be applied on the anomaly score to determine if a test instance is anomalous or not. On the other hand, select n instances with the largest anomaly scores as the anomalies.

The basic technique has been extended by researchers in three different ways. The first set of variants modifies the definition to obtain the anomaly score of a data instance. The second set of variants uses different distance/similarity measures to handle different data types. The third set of variants focuses on improving the efficiency of the basic technique (the complexity of the basic technique is O(N2), where N is the data size) in different ways.

2.4.3 Density Based Outlier Detection

Density-based anomaly detection techniques estimate the density of the neighbourhood of each data instance. An instance that lies in a neighbourhood with low density is declared to be anomalous while an instance that lies in a dense neighbourhood is declared to be normal.

Density-based techniques perform poorly if the data has regions of varying densities. To handle the issue of varying densities in the data set, a set of techniques has been proposed to compute the density of instances relative to the density of their neighbours. We may assign an anomaly score to a given data instance, known as *Local Outlier Factor* (LOF). For any given data instance, the LOF score is equal to ratio of average local density of the *k* nearest neighbours of the instance and the local density of the data instance itself.

For a given data instance, the distance to its *kth* nearest neighbour is equivalent to the radius of a hyper-sphere, centred at the given data instance, which contains *k* other instances. Thus the distance to the *kth* nearest neighbour for a given data instance can be viewed as an estimate of the inverse of the density of the instance in the data set and the basic nearest neighbour-based technique described in the previous subsection can be considered as a density-based anomaly detection technique.

2.4.4 Conditional Anomaly Detection

The anomaly detection methodology considered in this project called *conditional anomaly detection*, or *CAD*, takes into account the difference between the user specified environmental and indicator attributes during the anomaly detection process, and how this affects the idea of an “anomaly”.

This technique is used to detect contextual outliers. The context is of utmost importance in this approach. Two algorithms are famous in this approach.

1. Conditional Anomaly Detection
2. Bayesian network based outlier detection

The focus of this project is on Bayesian Network based outlier detection. Bayesian networks (BNs) are graphical models for reasoning under uncertainty, where the nodes represent variables (discrete or continuous) and arcs represent direct connections between them. These direct connections are often causal connections. In addition, BNs model the quantitative strength of the connections between variables, allowing probabilistic beliefs about them to be updated automatically as new information becomes available.

Despite the name, Bayesian networks do not necessarily imply a commitment to Bayesian statistics. Rather, they are so called because they use Bayes' rule for probabilistic inference, as we explain below. (The term "directed graphical model" is perhaps more appropriate). Bayesian nets are a useful representation for hierarchical Bayesian models, which form the foundation of applied Bayesian statistics. In such a model, the parameters are treated like any other random variable, and becomes nodes in the graph.

CHAPTER 3
OUTLIER DETECTION ALGORITHMS

3.1 Univariate Outlier Detection using Box Plot

The *box plot rule* (Figure 4) is the simplest statistical technique that has been applied to detect univariate and multivariate anomalies in medical domain data and turbine rotors data. A box plot graphically depicts the data using summary attributes such as smallest non-anomaly observation (*min*), lower quartile ($Q1$), median, upper quartile ($Q3$), and largest non-anomaly observation (*max*). The quantity $Q3 - Q1$ is called the *Inter Quartile Range* (*IQR*). The box plots also indicates the limits beyond which any observation will be treated as an anomaly. A data instance that lies more than 1.5* *IQR* lower than $Q1$ or 1.5* *IQR* higher than $Q3$ is declared as an anomaly. The region between $Q1 - 1.5IQR$ and $Q3 + 1.5IQR$ contains 99.3% of observations, and hence the choice of the $1.5IQR$ boundary makes the *box plot rule* equivalent to the 3σ technique for Gaussian data.

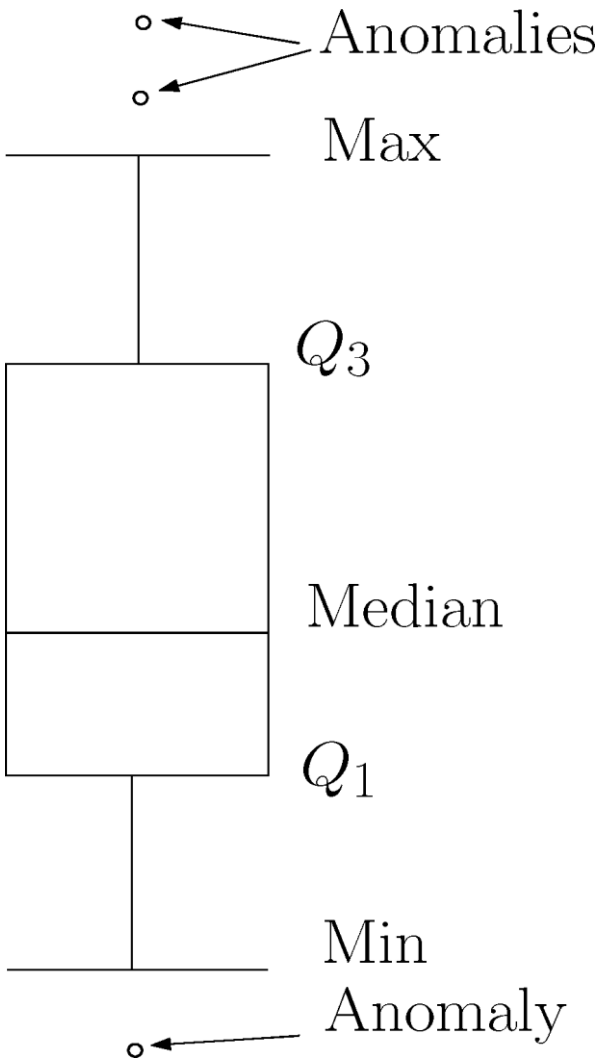


Fig. 4. A box plot for a univariate data set.

3.2 Multivariate Outlier Detection using Mahalanobis Distance Measure

A basic nearest neighbour anomaly detection technique is based on the following definition: The anomaly score of a data instance is defined as its distance to its kth nearest neighbour in a given data set. This basic technique has been applied to detect land mines from satellite ground images and to detect shorted turns (anomalies) in the DC field windings of large synchronous turbine-generators.

Nearest neighbour-based anomaly detection techniques require a distance or similarity measure defined between two data instances. Distance (or similarity) between two data instances can be computed in different ways. In this approach Mahalanobis Distance Measure is used.

Nearest neighbour-based anomaly detection techniques can be broadly grouped into two categories:
(1) techniques that use the distance of a data instance to its *kth* nearest neighbour as the anomaly score;
(2) techniques that compute the relative density of each data instance to compute its anomaly score.

For two data points p and q Mahalanobis distance is given by:

$$mahalanobis(p,q) = \sqrt{(p-q)\sum^{-1}(p-q)^T}$$

Where, ∑ is covariance matrix

$$cov(x,y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{n-1}$$

Where, x & y are variables and $\bar{x}$ & $\bar{y}$ are means of their respective variables and n is the number of data points in the sample.

3.3 Mining distance based outliers using K nearest neighbor criteria (Euclidean distance)

A basic nearest neighbour anomaly detection technique is based on the following definition: The anomaly score of a data instance is defined as its distance to its kth nearest neighbour in a given data set. This basic technique has been applied to detect land mines from satellite ground images and to detect shorted turns (anomalies) in the DC field windings of large synchronous turbine-generators. In the latter paper the authors use k = 1. Usually, a threshold is then be applied on the anomaly score to determine if a test instance is anomalous or not. On the other hand, select n instances with the largest anomaly scores as the ouliers .
The Euclidean Distance Formula is:

$$dist = \sqrt{\sum_{k=1}^{n} (p_k - q_k)^2}$$

Where n is the number of dimensions (attributes) and p_k and q_k are, respectively, the k^{th} attributes of data points p and q.

3.4 Mining distance based outliers in near linear time using randomization

Several variants of the basic technique have been proposed to improve the efficiency. Some techniques prune the search space by either ignoring instances that cannot be anomalous or by focussing on instances that are most likely to be anomalous. It can be shown that for sufficiently randomized data, a simple pruning step could result in the average complexity of the nearest neighbour search to be nearly linear. After calculating the nearest neighbours for a data instance, the algorithm sets the anomaly threshold for any data instance to the score of the weakest anomaly found so far.

Using this pruning procedure, the technique discards instances that are close, and hence not interesting. This reduces the complexity to O(nlogn) from $O(n^2)$.

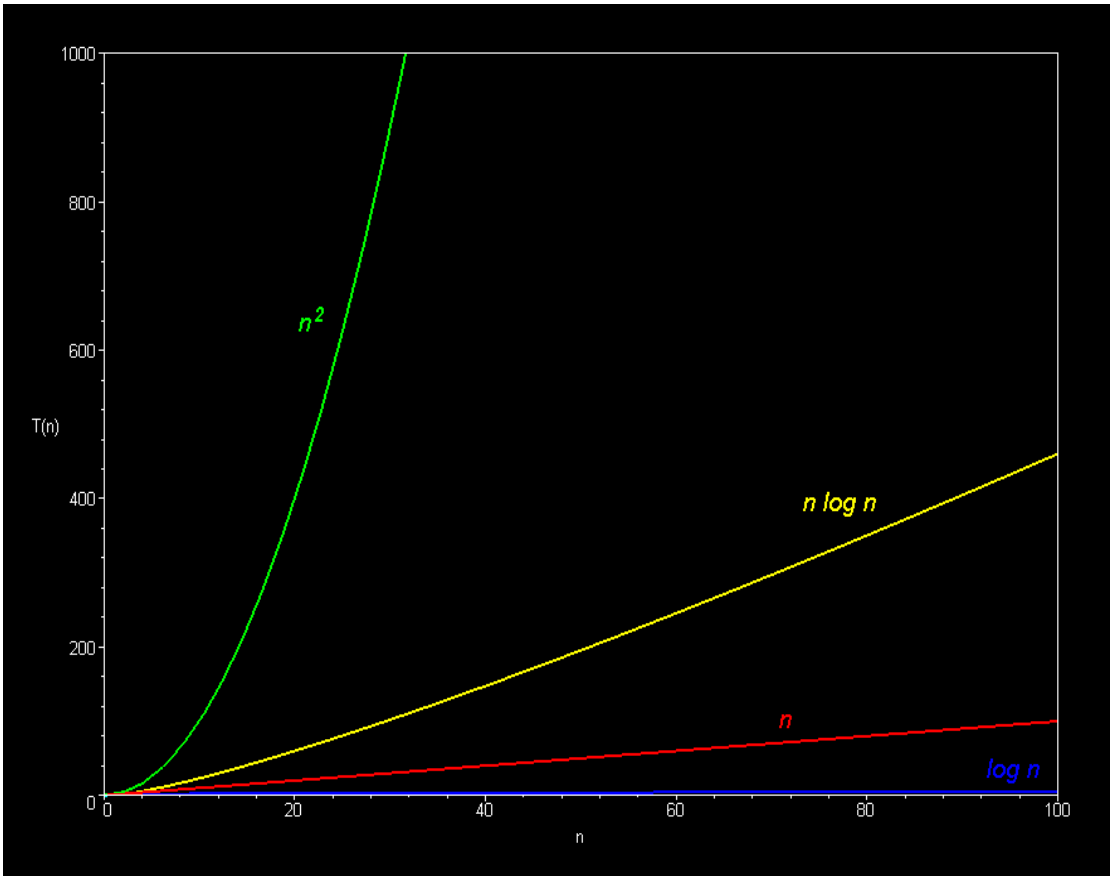


Fig 5. Graph of various runtime complexities

3.5 Identifying density based local outliers- local outlier factor

Density-based techniques perform poorly if the data has regions of varying densities. For example, consider the two-dimensional data set shown in Figure 7. Due to the low density of the cluster C_1, it is apparent that for every instance q inside the cluster C_1, the distance between the instance q and its nearest neighbour is greater than the distance between the instance p_2 and the nearest neighbour from the cluster C_2, and the instance p_2 will not be considered as anomaly. Hence, the basic technique will fail to distinguish between p_2 and instances in C_1. However, the instance p_1 may be detected.

To handle the issue of varying densities in the data set, a set of techniques has been proposed to compute the density of instances relative to the density of their neighbours. We may assign an anomaly score to a given data instance, known as *Local Outlier Factor* (LOF). For any given data instance, the LOF score is equal to ratio of average local density of the k nearest neighbours of the instance and the local density of the data instance itself. To find the local density for a data instance, the authors first find the radius of the smallest hyper-sphere centred at the data instance, that contains its k nearest neighbours. The local density is then computed by dividing k by the volume of this hyper-sphere. For a normal instance lying in a dense region, its local density will be similar to that of its neighbours, while for an anomalous

instance, its local density will be lower than that of its nearest neighbours. Hence the anomalous instance will get a higher LOF score.

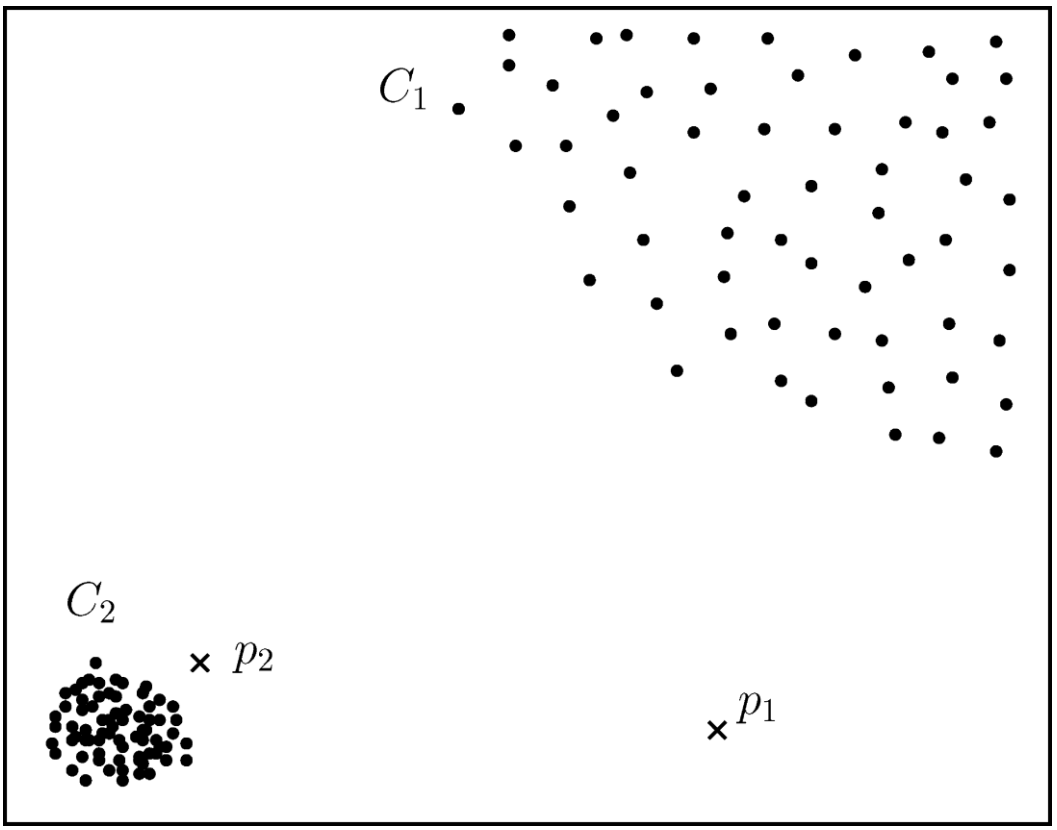


Fig.6. Advantage of local density-based techniques over global density-based techniques.

In the example shown in Figure 7, LOF will be able to capture both anomalies, p_1 and p_2, due to the fact that it considers the density around the data instances. Several researchers have proposed variants of the LOF technique. Some of these variants estimate the local density of an instance in a different way. Some variants have adapted the original technique to more complex data types. Since the original LOF technique is $O(N^2)$ (N is the data size), several techniques have been proposed that improve the efficiency of LOF.

Lets discuss a variation of the LOF, called *Connectivity-based Outlier Factor* (COF). The difference between LOF and COF is the manner in which the k neighbourhood for an instance is computed. In COF, the neighbourhood for an instance is computed in an incremental mode. To start, the closest instance to the given instance is added to the neighbourhood set. The next instance added to the neighbourhood set is such that its distance to the existing neighbourhood set is minimum among all remaining data instances. The distance between an instance and a set of instances is defined as the minimum distance between the given instance and any instance belonging to the given set. The neighbourhood is grown in this manner until it reaches size k. Once the neighbourhood is computed, the anomaly score (COF) is computed in the same manner as LOF. COF is able to capture regions such as straight lines, as shown in Figure 7.

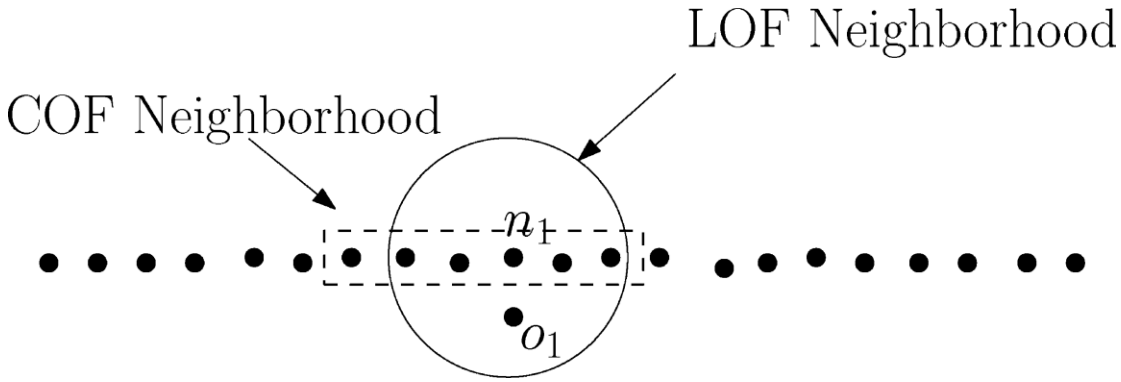


Fig. 7. Difference between the neighbourhoods computed by LOF and COF

3.6 Conditional Anomaly Detection

Rather than trying to find new and intriguing classes of anomalies, it is perhaps more important to ensure that those data points that a method does find *are in fact surprising*. To accomplish this, we ask the questions: What is the biggest source of inaccuracy for existing anomaly detection methods? Why might they return a large number of points that are not anomalies? To answer, we note that by definition, "statistical" methods for anomaly detection look for data points that refute a standard null hypothesis asserting that all data were produced by the same generative process. The null hypothesis is represented either explicitly or implicitly. However, the questionable assumption made by virtually all existing methods is that there is no *apriori* knowledge indicating that all data attributes should not be treated in the same fashion.

This is an assumption that is likely to cause problems with false positives in many problem domains. In almost every application of anomaly detection, there are likely to be several data attributes that a human being would never consider to be directly indicative of an anomaly. By allowing an anomaly detection methodology to consider such attributes equally, accuracy may suffer.

For example, consider the application of online anomaly detection to syndromic surveillance, where the goal is to detect a disease outbreak at the earliest possible instant. Imagine that we monitor two variables: *max daily temp* and *num fever*. *max daily temp* tells us the maximum outside temperature on a given day, and *num fever* tells us how many people were admitted to a hospital emergency room complaining of a high fever. Clearly, *max daily temp* should never be taken as direct evidence of an anomaly. Whether it was hot or cold on a given day should never directly indicate whether or not we think we have seen the start of an epidemic. For example, if the high in Gainesville, Florida on a given June day was only 70 degrees Fahrenheit (when the average high temperature is closer to 90 degrees), we simply have to accept that it was an abnormally cool day, but this does not indicate in any way that an outbreak has occurred.

While the temperature may not directly indicate an anomaly, it is not acceptable to simply ignore *max daily temp*, because *num fever* (which clearly is of interest in detecting an outbreak) may be directly affected by *max daily temp*, or by a hidden variable whose value can easily be deduced by the value of the *max daily temp* attribute. In this example, we know that people are generally more susceptible to illness in the winter, when the weather is cooler. We call attributes such as *max daily temp environmental* attributes. The remainder of the date attributes (which the user *would* consider to be directly indicative of anomalous data) are called *indicator* attributes.

The anomaly detection methodology considered in this project called *conditional anomaly detection*, or *CAD*, takes into account the difference between the user specified environmental and indicator attributes during the anomaly detection process, and how this affects the idea of an "anomaly". For example, consider Figure 9. In this Figure, Point A and Point B are both anomalies or outliers based on most conventional definitions. However, if we make use of the additional information that *max daily temp* is not directly indicative of an anomaly, then it is likely safe for an anomaly detection system to ignore Point A. Why? If we accept that it is a cold day, then encountering a large number of fever cases makes sense, reducing the interest of this observation. For this reason, the CAD methodology will only label Point B an anomaly.

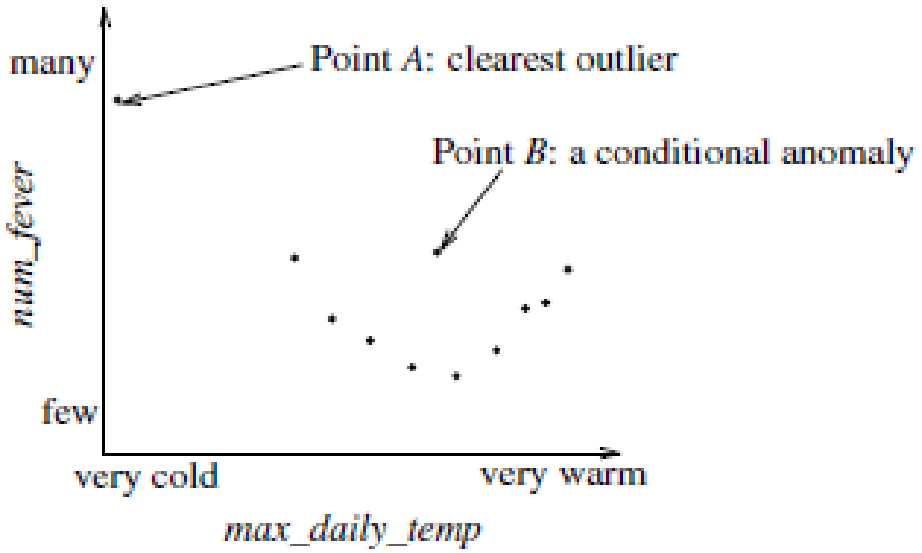


Fig.8. Syndromic Surveillance Application

3.7 Bayesian network based outlier detection

Bayesian networks (BNs) are graphical models for reasoning under uncertainty, where the nodes represent variables (discrete or continuous) and arcs represent direct connections between them. These direct connections are often causal connections. In addition, BNs model the quantitative strength of the connections between variables, allowing probabilistic beliefs about them to be updated automatically as new information becomes available.

Bayesian networks (BNs) are graphical models for reasoning under uncertainty, where the nodes represent variables (discrete or continuous) and arcs represent direct connections between them. These direct connections are often causal connections. In addition, BNs model the quantitative strength of the connections between variables, allowing probabilistic beliefs about them to be updated automatically as new information becomes available.

My attention is on the case of linear conditional Gaussian (also known as Normal) distributions and the case of linear conditional Gaussian Bayesian networks. We refer to a linear conditional Gaussian Bayesian network as an LCG Bayesian network. In such Bayesian networks discrete nodes can't have continuous parents.

Bayesian networks have been used for anomaly detection in the multi-class setting. A basic technique for a univariate categorical data set using a Na¨ıve Bayesian network estimates the posterior probability of observing a class label from a set of normal class labels and the anomaly class label, given a test data instance. The class label with largest posterior is chosen as the predicted class for the given test instance. The likelihood of observing the test instance given a class and the prior on the class probabilities, is estimated from the training data set.

The basic technique can be generalized to multivariate categorical data sets by aggregating the per-attribute posterior probabilities for each test instance and using the aggregated value to determine the outliers. The outlier in this case being the test instance with least joint probability.

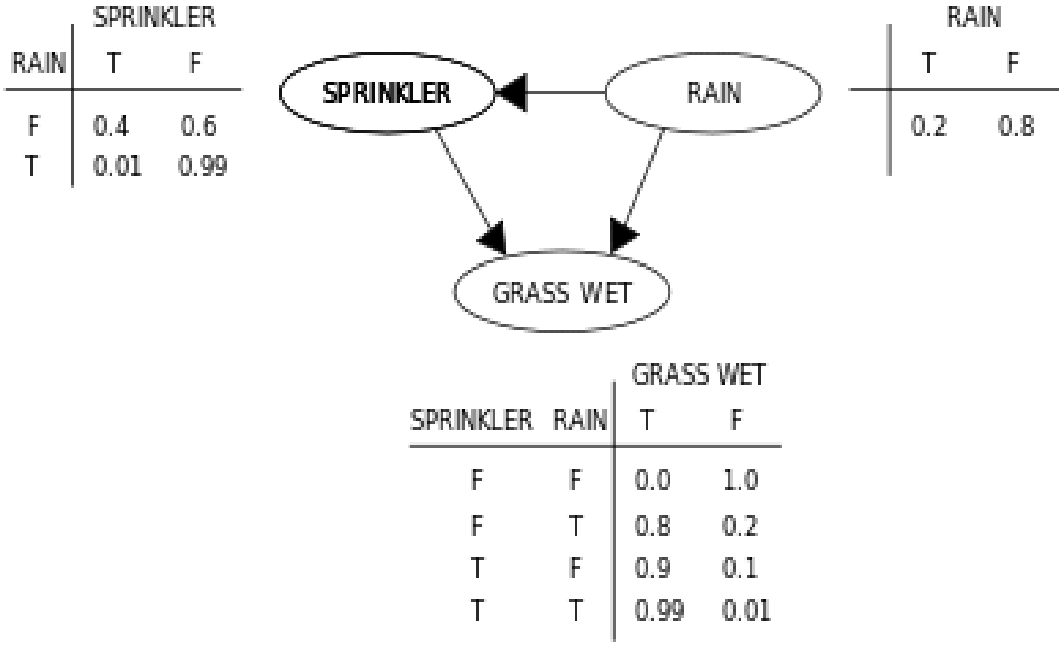
This basic technique assumes independence between the different attributes. Several variations of the basic technique have been proposed that capture the conditional dependencies between the different attributes using more complex Bayesian networks. This is used for computing the joint probability for different possibilities of occurrences. The occurrences that have low joint probability i.e. low probability of occurrence are termed as outliers. This approach in particular will be used in the project.

I have implemented two algorithms using this approach.

1. Bayesian Network based Outlier Detection for discrete nodes using conditional probability table
2. Bayesian Network based Outlier Detection using parameter learning

The algorithms have been implemented in MATLAB using an additional tool called Bayesian Network Toolbox (BNT). BNT is an open-source collection of MATLAB functions for inference and learning of (directed) graphical models. It was written by Kevin Murphy in 1997 and has been updated many times. It has 43,000 lines of code of which 8,000 are comments.

First algorithm is for discrete nodes and makes use of conditional probability tables which are used to compute marginal probability of each node and hence joint probability for the network.



RAIN	SPRINKLER T	SPRINKLER F
F	0.4	0.6
T	0.01	0.99

RAIN T	RAIN F
0.2	0.8

SPRINKLER	RAIN	GRASS WET T	GRASS WET F
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

Fig.9. Bayesian Network discrete node Example

Bayesian Joint Probability:

$$\mathbf{P}(X_1, X_2, .., X_n) = \prod_{i=1,..n} \mathbf{P}(X_i \mid pa(X_i))$$

where pa stands for parents of X_i.

Second algorithm is for continuous and discrete nodes where continuous nodes follow Gaussian distribution and detects outliers using parameter learning of BNT.

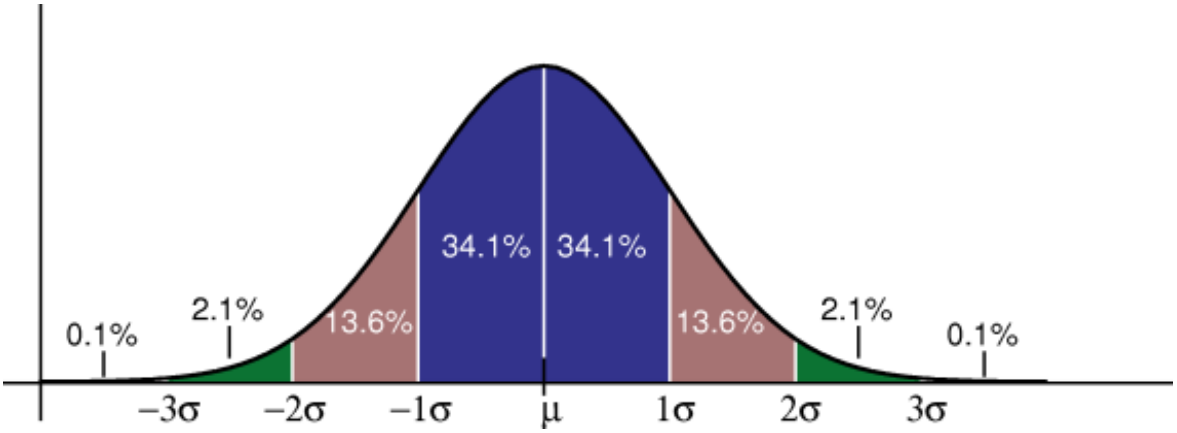


Fig.10. Gaussian Distribution graph with mean μ and standard deviation σ

CHAPTER 4
Implementation

Working Assumption:

1.The basic approach followed for outlier detection is unsupervised.
2.The attributes in data sets that are discrete or non-numeric in character are transformed by giving them numerals for each class value.
3.There are considerably more normal observations than abnormal observations.

4.1 Codes

This includes:
1. Code for Graphical User Interface developed in MATLAB.
2. Code for various Outlier Detection Algorithms.

4.1.1 Code for Graphical User Interface:

```
function varargout = suite(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @suite_OpeningFcn, ...
                   'gui_OutputFcn',  @suite_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
% --- Executes just before suite is made visible.
function suite_OpeningFcn(hObject, eventdata, handles, varargin)
% Choose default command line output for suite
handles.output = hObject;
% Update handles structure
guidata(hObject, handles);
```

```
% --- Outputs from this function are returned to the command line.
function varargout = suite_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
function input_Callback(hObject, eventdata, handles)
function input_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
function recall_input_Callback(hObject, eventdata, handles)
function recall_input_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
% --- Executes on button press in discrete_nodes.
function discrete_nodes_Callback(hObject, eventdata, handles)
cla;
legend(handles.axes1,'hide');
[handles.inp,handles.recall]=bayesian();
set(handles.input,'String',handles.inp);
set(handles.recall_input,'String',handles.recall);
% --- Executes on button press in continuous_nodes.
function continuous_nodes_Callback(hObject, eventdata, handles)
cla;
legend(handles.axes1,'hide');
[handles.inp,handles.recall]=bayescontinuous();
set(handles.input,'String',handles.inp);
set(handles.recall_input,'String',handles.recall);
guidata(hObject, handles);
% --- Executes on button press in euclidean_distance.
function euclidean_distance_Callback(hObject, eventdata, handles)
cla;
[handles.inp,handles.recall]=euclidean();
set(handles.input,'String',handles.inp);
set(handles.recall_input,'String',handles.recall);
guidata(hObject, handles);
% --- Executes on button press in mahalanobis_distance.
function mahalanobis_distance_Callback(hObject, eventdata, handles)
cla;
[handles.inp,handles.recall]=mahalanobis();
set(handles.input,'String',handles.inp);
set(handles.recall_input,'String',handles.recall);
guidata(hObject, handles);
% --- Executes on button press in randomization_pruning.
```

```
function randomization_pruning_Callback(hObject, eventdata, handles)
cla;
[handles.inp,handles.recall]=pruning();
set(handles.input,'String',handles.inp);
set(handles.recall_input,'String',handles.recall);
guidata(hObject, handles);
% --- Executes on button press in boxplot.
function boxplot_Callback(hObject, eventdata, handles)
cla;
[handles.inp,handles.recall]=box();
set(handles.input,'String',handles.inp);
set(handles.recall_input,'String',handles.recall);
guidata(hObject, handles);
```

4.1.2 Code for Univariate Outlier Detection using Box Plot:

```
%Box Plot
function [inp,recall]=box()
clear all;
clc;
%Reading Data from csv file and storing in variables
file=uigetfile('*.csv','Select a file');
a=CSVREAD(file);
slength=a(:,1);
slength=sort(slength);
[m,n]=size(slength);
inp=sprintf('n/a');
%Computing Median and Separating halves
if(mod(m,2)==0)
   t=m/2;
   q=(slength(t)+slength(t+1))/2;
   for i=1:t
      slength1(1,i)=slength(i,1);
   end
   for i=(t+1):m
      slength2(1,(i-t))=slength(i,1);
   end
else
   t=(m+1)/2;
   q=slength(t);
   for i=1:(t-1)
      slength1(1,i)=slength(i,1);
   end
   for i=(t+1):m
      slength2(1,(i-t))=slength(i,1);
```

```
    end
end
%Computing Q1,Q3 & other factors
[m,n]=size(slength1);
if(mod(n,2)==0)
    t=n/2;
    q1=(slength1(t)+slength1(t+1))/2;
else
    t=(n+1)/2;
    q1=slength1(t);
end
[m,n]=size(slength2);
if(mod(n,2)==0)
    t=n/2;
    q3=(slength2(t)+slength2(t+1))/2;
else
    t=(n+1)/2;
    q3=slength2(t);
end
iqr=q3-q1;
lif=q1-(1.5*iqr);
lof=q1-(3*iqr);
uif=q3+(1.5*iqr);
uof=q3+(3*iqr);
%Computing Outliers
[m,n]=size(slength);
counte=0;
countm=0;
mo=[];
eo=[];
for i=1:m
    if (slength(i,1)<lof) | (slength(i,1)>uof)
        counte=counte+1;
        eo(1,counte)=slength(i,1);
    elseif (slength(i,1)<lif) | (slength(i,1)>uif)
        countm=countm+1;
        mo(1,countm)=slength(i,1);
    end
end
%Displaying
disp('The 1st parameter values are as follows:'),disp(slength);
X=sprintf('Q=%f',q);
disp(X);
X=sprintf('Q1=%f',q1);
disp(X);
```

```
X=sprintf('Q3=%f',q3);
disp(X);
X=sprintf('IQR=%f',iqr);
disp(X);
X=sprintf('Lower Inner Fence=%f',lif);
disp(X);
X=sprintf('Lower Outer Fence=%f',lof);
disp(X);
X=sprintf('Upper Inner Fence=%f',uif);
disp(X);
X=sprintf('Upper Outer Fence=%f',uof);
disp(X);
disp('Mild Outliers:');
disp(mo);
disp('Extreme Outliers:');
disp(eo);
boxplot(slength);
xlabel('Box Plot');
ylabel('1st Dimension');
%Code for printing outliers
h=flipud(findobj(gcf,'tag','Outliers')); % flip order of handles
for jj=1:length(h)
   x=get(h(jj),'XData');
   y=get(h(jj),'YData');
   for ii=1:length(x)
      if not(isnan(x(ii)))
         text(x(ii),y(ii),sprintf('\\leftarrowOutlier'))
      end
   end
end
[c,d]=size(mo);
[e,f]=size(eo);
recall=sprintf('No. of Mild Oultiers=%d, No. of Extreme Outliers=%d',d,f);
```

4.1.3 Code for Multivariate Outlier Detection using Mahalanobis Distance measure:

```
%Mahalanobis Distance
function [inp,Result]=mahalanobis()
clear all;
clc;
%Reading Data from csv file and storing in variables
file=uigetfile('*.csv','Select a file');
a=CSVREAD(file);
```

```
C=cov(a);
C=inv(C);
[M,N]=size(a);
mah=zeros(M);
n=8;
answer=inputdlg('Enter the value of k:','Enter the number');
k=str2num(answer{1});
inp=sprintf('k=%d',k);
%Introducing outliers
for i=1:M
   cat(i)='n';
end
cat(8)='o';
cat(24)='o';
cat(30)='o';
cat(65)='o';
cat(83)='o';
cat(97)='o';
cat(120)='o';
cat(150)='o';
%Calculating Mahalanobis Distance
for i=1:M
   for j=1:M
      for c=1:N
         pq(1,c)=a(i,c)-a(j,c);
      end
      product=C*pq';
      product=pq*product;
      product=sqrt(product);
      mah(i,j)=product;
   end
end
neighbour=mah;   %Mahalanobis Distance Matrix
neighbour=sort(neighbour,2,'ascend');   %Calculating Nearest Neighbours
points=neighbour(:,k);   %Selecting the k^th neighbours
parranged=sort(points,'descend');   %Arrange in Descending order
outliers=parranged(1:n,1);   %Selecting the outliers
for i=1:length(points)
   check(1,i)=0;
end
i=1;
while i~=n+1
   for j=1:length(points)
      if points(j)==outliers(i) && check(1,j)==0
         p(1,i)=j;   %Storing the respective data entries
```

```
            i=i+1;
            break;
        end
    end
end
%Calculating recall for the method
count=0;
for i=1:n
    if (cat(p(1,i))=='o')
        count=count+1;
    end
end
recall=(count/n)*100;
%Storing values for displaying on graph
for i=1:n
    for j=1:N
        values(i,j)=a(p(1,i),j);    %storing outlier values to plot
    end
end
%Displaying
disp('Following are the outliers distance:'),disp(outliers);
disp('Following are the outlier points:'),disp(p);
disp('Corresponding values are as follows:'),disp(values);
X=sprintf('For the Method, Recall= %f',recall);
disp(X);
%Plots
plot(a(:,1),a(:,2),'g*',values(:,1),values(:,2),'r*')
xlabel('1st Dimension');
ylabel('2nd Dimension');
legend('Non-Outliers','Outliers');
Result=sprintf('Recall= %f %',recall);
```

4.1.4 Code for Multivariate Outlier Detection using Euclidean Distance measure:

```
%Euclidean Distance
function [inp,Result]=euclidean()
clear all;
clc;
%Reading Data from csv file and storing in variables
file=uigetfile('*.csv','Select a file');
a=CSVREAD(file);
[M,N]=size(a);
euc=zeros(M);
```

```
answer=inputdlg('Enter the value of k:','Enter the number');
k=str2num(answer{1});
n=8;
inp=sprintf('k=%d',k);
%Introducing outliers
for i=1:M
   cat(i)='n';
end
cat(8)='o';
cat(24)='o';
cat(30)='o';
cat(65)='o';
cat(83)='o';
cat(97)='o';
cat(120)='o';
cat(150)='o';
%Calculating Euclidean and Hamming Distance Matrix
for i=1:M
   for j=1:M
      esum=0;
      for c=1:N
         esum=esum+((a(i,c)-a(j,c))*(a(i,c)-a(j,c)));
      end
      euc(i,j)=sqrt(esum);
   end
end
neighbour=euc;   %Total Distance Matrix
neighbour=sort(neighbour,2,'ascend');   %Calculating Nearest Neighbours
points=neighbour(:,k);   %Selecting the k^th neighbours
parranged=sort(points,'descend');   %Arrange in Descending order
outliers=parranged(1:n,1);   %Selecting the outliers
for i=1:length(points)
   check(1,i)=0;
end
i=1;
while i~=n+1
   for j=1:length(points)
      if points(j)==outliers(i) && check(1,j)==0
         p(1,i)=j;   %Storing the respective data entries
         i=i+1;
         break;
      end
   end
end
%Calculating recall for the method
```

```
count=0;
for i=1:n
   if (cat(p(1,i))=='o')
      count=count+1;
   end
end
recall=(count/n)*100;
%Storing values for displaying on graph
for i=1:n
   for j=1:N
      values(i,j)=a(p(1,i),j);    %storing outlier values to plot
   end
end
%Displaying
disp('Following are the outliers distance:'),disp(outliers);
disp('Following are the outlier points:'),disp(p);
disp('Corresponding values are as follows:'),disp(values);
X=sprintf('For the Method, Recall= %f',recall);
disp(X);
%Plots
plot(a(:,1),a(:,2),'g*',values(:,1),values(:,2),'r*')
xlabel('1st Dimension');
ylabel('2nd Dimension');
legend('Non-Outliers','Outliers');
Result=sprintf('Recall= %f %',recall);
```

4.1.5 Code for Multivariate Outlier Detection using Randomization and Pruning:

```
%Pruning Algorithm
function [inp,Result]=pruning()
clear all;
clc;
%Reading Data from csv file and storing in variables
file=uigetfile('*.csv','Select a file');
a=CSVREAD(file);
[M,N]=size(a);
arr=randperm(M); %generating random numbers for blocks
answer=inputdlg('Enter the value of k:','Enter the number');
k=str2num(answer{1});
n=8;
c=0;    %set the cut off for pruning to zero
outlier(1:M)=0;
inp=sprintf('k=%d',k);
```

```
%Introducing outliers
for i=1:M
   cat(i)='n';
end
cat(8)='o';
cat(24)='o';
cat(30)='o';
cat(65)='o';
cat(83)='o';
cat(97)='o';
cat(120)='o';
cat(150)='o';
%initialize score as high value
%Separating into blocks
M1=M/5;   %1st block and also size of each block
M1=floor(M1);
M2=2*M1;
M3=3*M1;
M4=4*M1;
for i=1:M
   if i<=M1
      B(1,i)=arr(i);
   elseif i<=M2
      B(2,(i-M1))=arr(i);
   elseif i<=M3
      B(3,(i-M2))=arr(i);
   elseif i<=M4
      B(4,(i-M3))=arr(i);
   else
      B(5,(i-M4))=arr(i);
   end
   score(i)=100;
end
%General Algorithm
block=0;
while block~=5
   block=block+1;
   neighbour(1:M1,1:M)=100;    %initialize neighbours as high value
   for d=1:M
      for b=1:M1
         j=B(block,b);
         dist=0;
         if j~=-1
            esum=0;
            for f=1:N
```

```
            esum=esum+((a(d,f)-a(j,f))*(a(d,f)-a(j,f)));
        end
        dist=sqrt(esum);
    end
    if dist~=0
        z=1;
        while neighbour(b,z)~=100
            z=z+1;
        end
        neighbour(b,z)=dist;
        neighbour=sort(neighbour,2,'ascend');   %Calculating Nearest Neighbours
        score(j)=neighbour(b,k);    %Calculating score
        if score(j)~=100
            if score(j)<c
                B(block,b)=-1;  %Remove b from B
            end
        end
    end
  end
end
%Taking top n outliers
t=1;
while outlier(t)~=0
  t=t+1;
end
t=t-1;
for i=1:M1
  j=B(block,i);
  if j~=-1
    t=t+1;
    outlier(t)=j;
    for i=2:t
      temp=outlier(i);
      j=i-1;
      while (j>=1 & score(temp)>score(outlier(j)))
        outlier(j+1)=outlier(j);
        j=j-1;
      end
      outlier(j+1)=temp;
    end
  end
end
for i=1:length(outlier)
  if i>n
    outlier(i)=0;
```

```
        end
    end
    c=min(score);   %redefinig cut off
end
%Calculating recall for the method
count=0;
for i=1:n
    if (cat(outlier(i))=='o')
        count=count+1;
    end
end
recall=(count/n)*100;
%Storing values for displaying on graph
for i=1:n
    for j=1:N
        values(i,j)=a(outlier(i),j);    %storing outlier values to plot
    end
end
%Displaying
disp('Following are the outlier points:'),disp(outlier(1:n));
disp('Corresponding values are as follows:'),disp(values);
X=sprintf('For the Method, Recall= %f',recall);
disp(X);
%Plots
plot(a(:,1),a(:,2),'g*',values(:,1),values(:,2),'r*')
xlabel('1st Dimension');
ylabel('2nd Dimension');
legend('Non-Outliers','Outliers');
Result=sprintf('Recall= %f %',recall);
```

4.1.6 Code for Bayesian Network based Outlier Detection for discrete nodes using conditional probability tables:

```
%Bayesian Network
function [inp,Result]=bayesian()
clear all;
clc;
recall=sprintf('n/a');
answer=inputdlg('Enter 1 for the built in network and 2 to generate your own
network','Enter the option');
k=str2num(answer{1});
answer=inputdlg('Enter the no. of outliers to be detected:','Enter the number');
o=str2num(answer{1});
inp=sprintf('Option=%d, No. of Outliers=%d',k,o);
if k==1
```

```
    n=4;
    dag=zeros(n,n); %Directed Acylic Graph
    for i=1:n
       a(1,i)=i;
    end
    dag(a(1,1),a(1,2))=1;
    dag(a(1,1),a(1,3))=1;
    dag(a(1,3),a(1,4))=1;
    dag(a(1,2),a(1,4))=1;
    node_sizes=2*ones(1,n);

bnet=mk_bnet(dag,node_sizes,'names',{'cloudy','sprinkler','rain','wet'},'discrete',1:n);
    G=bnet.dag;
    draw_graph(G);
    C=bnet.names('cloudy');
    R=bnet.names('rain');
    S=bnet.names('sprinkler');
    W=bnet.names('wet');
    bnet.CPD{C}=tabular_CPD(bnet,C,[0.5 0.5]);
    bnet.CPD{R}=tabular_CPD(bnet,R,[0.8 0.2 0.2 0.8]);
    bnet.CPD{S}=tabular_CPD(bnet,S,[0.5 0.9 0.5 0.1]);
    bnet.CPD{W}=tabular_CPD(bnet,W,[1 0.1 0.1 0.01 0 0.9 0.9 0.99]);
elseif k==2
       answer=inputdlg('Enter the number of nodes:','Enter the number');
       n=str2num(answer{1});
       dag=zeros(n,n);
       for i=1:n
          a(1,i)=i;
       end
       for i=1:n
          for j=1:n
             X=sprintf('Connection from %d to %d:',i,j);
             answer=inputdlg(X,'Enter 1 for a connection and 0 otherwise');
             x=str2num(answer{1});
             dag(i,j)=x;
          end
       end
       node_sizes=2*ones(1,n);
       bnet=mk_bnet(dag,node_sizes,'discrete',1:n);
       for i=1:n
          [c,d]=size(parents(dag,i));
          d=d+1;
          m=1;
          for l=1:d
             m=2*m;
```

```
            end
            X=sprintf('%d probability values have to be entered',m);
            for j=1:m
               answer=inputdlg('Enter the value of p:',X);
               p=str2num(answer{1});
               value(i,j)=p;
            end
            bnet.CPD{i}=tabular_CPD(bnet,i,value(i,1:m));
        end
end
%Generating random conditions
condition=1+rand(20,n);
condition=round(condition);
%Calculating marginal probability of each node
[e,f]=size(condition);
for k=1:e
   product=1;
   for i=1:n
      engine=jtree_inf_engine(bnet);
      evidence=cell(1,n);
      ps=parents(dag,i);
      [c,d]=size(ps);
      for j=1:d
         evidence{a(1,ps(1,j))}=condition(k,ps(1,j));
      end
      [engine,ll]=enter_evidence(engine, evidence);
      m=marginal_nodes(engine,a(1,i));
      product=product*m.T(condition(k,a(1,i)));
   end
   probability(k,1)=product;
end
%Detecting top o outliers
parranged=sort(probability,'ascend');   %Arrange in Descending order
outliers=parranged(1:o,1);   %Selecting the outliers
for i=1:length(probability)
   check(1,i)=0;
end
i=1;
while i~=o+1
   for j=1:length(probability)
      if probability(j)==outliers(i) && check(1,j)==0
         p(1,i)=j;   %Storing the respective data entries
         i=i+1;
         break;
      end
```

```
    end
end
%Storing values of conditions
for i=1:o
    for j=1:n
        values(i,j)=condition(p(1,i),j);    %storing outlier values to plot
    end
end
disp('Joint Probability for different conditions are as follows:');
probability
disp('Logic: Lower probability means that the condition has lower chance of
occurrence and hence an outlier');
disp('Top outliers have probability');
outliers
disp('Index: 1 for false and 2 for true')
disp('Corresponding conditions are as follows:');
values
G=bnet.dag;
draw_graph(G);
xlabel('Bayesian Network');
ylabel('');
Result=sprintf('The top outlier has probability %f\nThe conditions are as
follows:',outliers(1));
r=sprintf('%d ',values(1,:));
Result=strcat(Result,r);
```

4.1.7 Code for Bayesian Network based Outlier Detection for discrete and continuous nodes where continuous nodes follow Gaussian distribution using parameter learning:

```
%Bayesian Network with continuous variables
function [inp,Result]=bayescontinuous()
clear all;
clc;
answer=inputdlg('Enter 1 for the built in network and 2 to generate your own
network','Enter the option');
k=str2num(answer{1});
o=8;   %No. of outliers present in the test dataset
correct=1;
inp=sprintf('Option=%d',k);
if k==1
    n=5;
    data=load('E:\Project Work\iris_bayes.csv');
    ncases=size(data,1);
    cases=cell(n,ncases);
```

```
    cases(:,:)=num2cell(data');
    dag=zeros(n,n); %Directed Acylic Graph
    for i=1:n
       a(1,i)=i;
    end
    dag(a(1,5),a(1,1))=1;
    dag(a(1,5),a(1,2))=1;
    dag(a(1,5),a(1,3))=1;
    dag(a(1,5),a(1,4))=1;
    node_sizes=[1 1 1 1 3];
    bnet=mk_bnet(dag,node_sizes,'discrete',5);
    bnet.CPD{a(1,1)}=gaussian_CPD(bnet,a(1,1));
    bnet.CPD{a(1,2)}=gaussian_CPD(bnet,a(1,2));
    bnet.CPD{a(1,3)}=gaussian_CPD(bnet,a(1,3));
    bnet.CPD{a(1,4)}=gaussian_CPD(bnet,a(1,4));
    bnet.CPD{a(1,5)}=tabular_CPD(bnet,a(1,5));
    engine=jtree_inf_engine(bnet);
    bnet2=learn_params(bnet,cases);
    engine2=jtree_inf_engine(bnet2);
else if k==2
        answer=inputdlg('Enter the number of nodes:','Enter the number');
        n=str2num(answer{1});
        dag=zeros(n,n);
        for i=1:n
           a(1,i)=i;
        end
        for i=1:n
           for j=1:n
              X=sprintf('Connection from %d to %d:',i,j);
              answer=inputdlg(X,'Enter 1 for a connection and 0 otherwise');
              x=str2num(answer{1});
              dag(i,j)=x;
           end
        end
        answer=inputdlg('Enter the number of discrete nodes:','Enter the number');
        y=str2num(answer{1});
        for i=1:y
           answer=inputdlg('Enter the discrete node:','Enter the number');
           dnodes(1,i)=str2num(answer{1});
        end
        for i=1:n
           X=sprintf('Enter the size of node %d',i);
           answer=inputdlg(X,'Enter the size of each node');
           node_sizes(1,i)=str2num(answer{1});
        end
```

```
        %Checking for correct bayesian network
        for i=1:n
           if node_sizes(1,i)~=1
              ps=parents(dag,i);
              [c,d]=size(ps);
              for j=1:d
                 if node_sizes(1,ps(1,j))==1
                    correct=0;
                 end
              end
           end
        end
        if correct==0
           disp('Bayesian Network entered is not correct as it does not follow LCG
conditions');
           return;
        end
        file=uigetfile('*.csv','Select a file for training dataset');
        data=load(file);
        ncases=size(data,1);
        cases=cell(n,ncases);
        cases(:,:)=num2cell(data');
        bnet=mk_bnet(dag,node_sizes,'discrete',dnodes);
        for i=1:n
           if node_sizes(1,i)==1
              bnet.CPD{a(1,i)}=gaussian_CPD(bnet,a(1,i));
           else
              bnet.CPD{a(1,i)}=tabular_CPD(bnet,a(1,i));
           end
        end
        engine=jtree_inf_engine(bnet);
        bnet2=learn_params(bnet,cases);
        engine2=jtree_inf_engine(bnet2);
     end
end
G=bnet.dag;
draw_graph(G);
xlabel('Bayesian Network');
ylabel('');
%Getting test dataset
file=uigetfile('*.csv','Select a file for test dataset');
condition=csvread(file);
[M,N]=size(a);
%Introducing outliers
for i=1:M
```

```
    cat(i)='n';
end
cat(2)='o';
cat(6)='o';
cat(8)='o';
cat(11)='o';
cat(13)='o';
cat(16)='o';
cat(24)='o';
cat(30)='o';
%Calculating marginal probability of each node
[e,f]=size(condition);
for k=1:e
    product=1;
    for i=1:n
        engine2=jtree_inf_engine(bnet2);
        evidence=cell(1,n);
        ps=parents(dag,i);
        [c,d]=size(ps);
        for j=1:d
            evidence{a(1,ps(1,j))}=condition(k,ps(1,j));
        end
        [engine3,ll]=enter_evidence(engine2, evidence);
        m=marginal_nodes(engine3,a(1,i));
        if node_sizes(1,i)==1
            prob=normpdf(condition(k,a(1,i)),m.mu,m.Sigma);
            product=product*prob;
        else
            product=product*m.T(condition(k,a(1,i)));
        end
    end
    probability(k,1)=product;
end
%Detecting top o outliers
parranged=sort(probability,'ascend');   %Arrange in Ascending order
outliers=parranged(1:o,1);   %Selecting the outliers
for i=1:length(probability)
    check(1,i)=0;
end
i=1;
while i~=o+1
    for j=1:length(probability)
        if probability(j)==outliers(i) && check(1,j)==0
            p(1,i)=j;   %Storing the respective data entries
            i=i+1;
```

```
            check(1,j)=1;
            break;
        end
    end
end
%Calculating recall for the method
count=0;
for i=1:o
    if (cat(p(1,i))=='o')
        count=count+1;
    end
end
recall=(count/o)*100;
%Storing values of conditions
for i=1:o
    for j=1:n
        values(i,j)=condition(p(1,i),j);    %storing outlier values to plot
    end
end
disp('Probability for different conditions are as follows:');
probability
disp('Logic: Lower probability means that the condition has lower chance of
occurrence and hence an outlier');
disp('Top outliers have probability');
outliers
disp('Corresponding conditions are as follows:');
values
X=sprintf('For the Method, Recall= %f',recall);
disp(X);
Result=sprintf('Recall= %f %',recall);
```

4.2 Example Datasets

Datasets	Dimensions
Iris Plant	150X4
Wine Quality	178X14
Forest Fires	517X11
Breast Cancer	699X10

4.3 Results

4.3.1 Graphical User Interface for the Suite

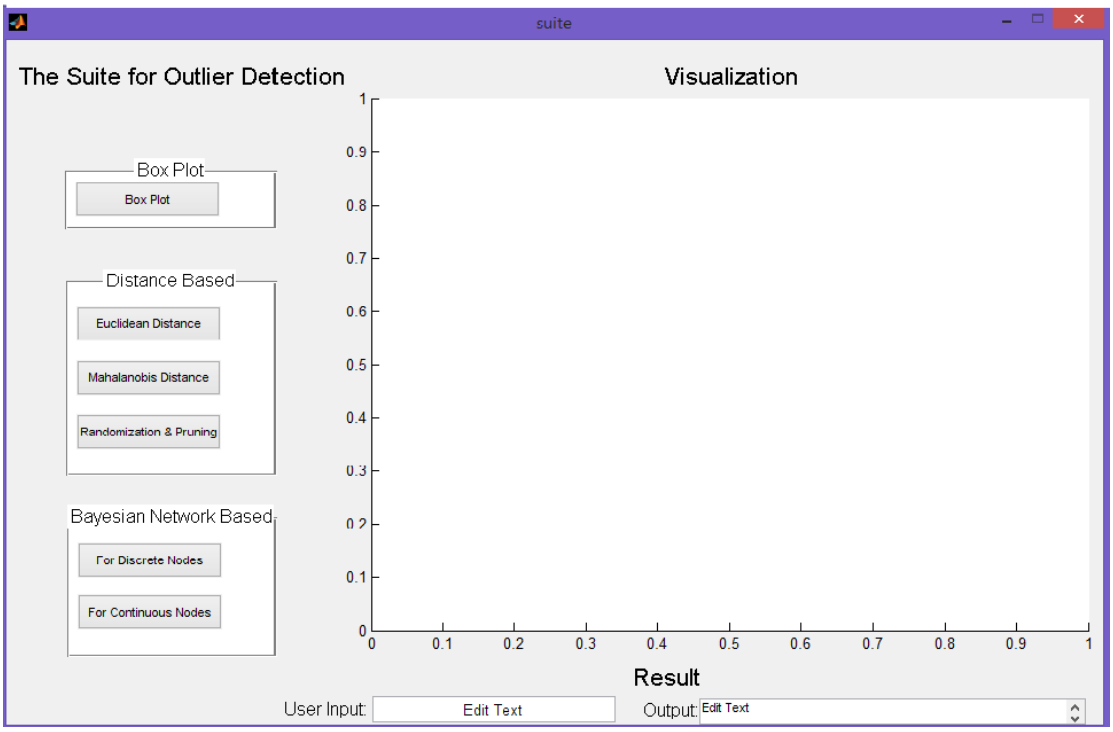


Fig 11. Depicts the Graphical User Interface for the suite

4.3.2 Univariate Outlier Detection using Box Plot

```
Q=5.800000
Q1=5.100000
Q3=6.400000
IQR=1.300000
Lower Inner Fence=3.150000
Lower Outer Fence=1.200000
Upper Inner Fence=8.350000
Upper Outer Fence=10.300000
Mild Outliers:
     3     3
Extreme Outliers:
     1
```

Fig 12. Result obtained by running Box Plot algorithm on Iris plant dataset in MATLAB

Fig 13 shows corresponding visualization.

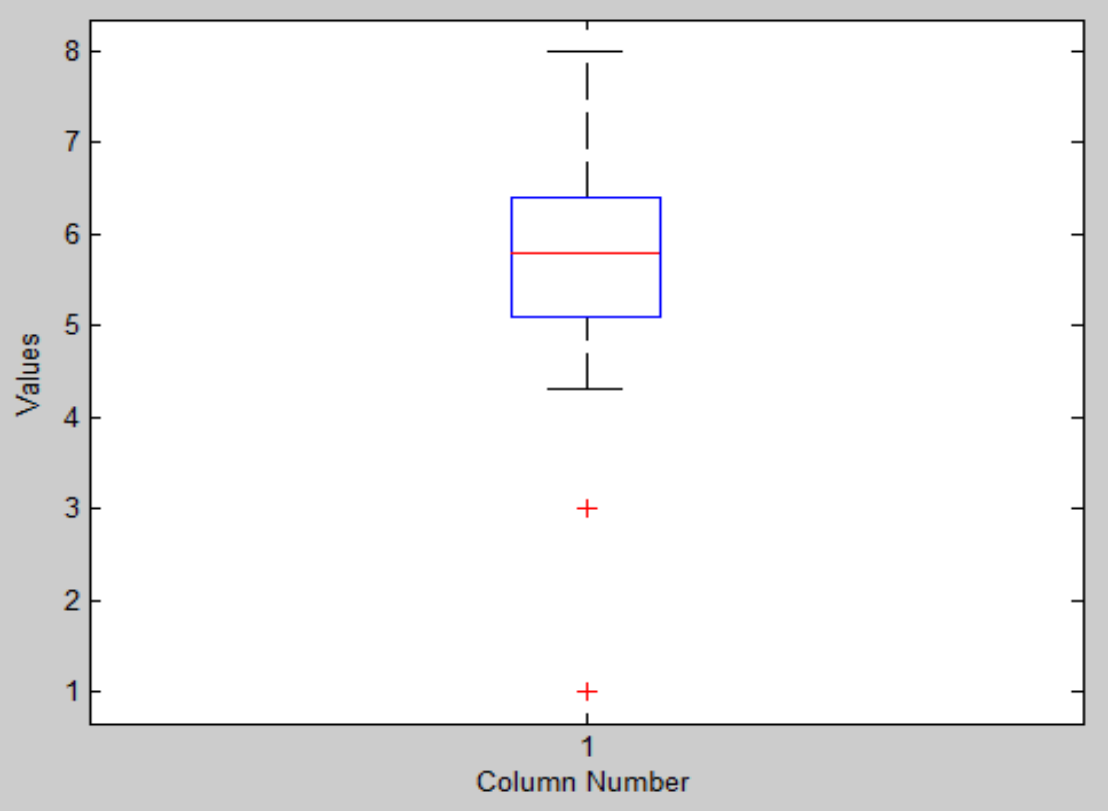


Fig 13. Red points on the figure depict outliers in Iris Plant dataset obtained using box plot

4.3.3 Multivariate Outlier Detection using Mahalanobis Distance Measure

```
Enter the value of k:6
Following are the outliers distance:
    5.8226
    5.7818
    5.1993
    5.1246
    4.2205
    4.0703
    3.1382
    1.7348
Following are the outliers points:
    24    30   150    65     8    97   120   132
Sepal Length: 1.000000, Sepal Width: 2.000000, Petal Length: 2.000000, Petal Width: 0.500000
Sepal Length: 5.000000, Sepal Width: 4.000000, Petal Length: 5.000000, Petal Width: 0.300000
Sepal Length: 3.000000, Sepal Width: 1.000000, Petal Length: 2.000000, Petal Width: 2.000000
Sepal Length: 8.000000, Sepal Width: 6.000000, Petal Length: 2.000000, Petal Width: 1.500000
Sepal Length: 7.000000, Sepal Width: 6.000000, Petal Length: 3.000000, Petal Width: 0.400000
Sepal Length: 3.000000, Sepal Width: 1.000000, Petal Length: 5.000000, Petal Width: 1.500000
Sepal Length: 5.000000, Sepal Width: 2.000000, Petal Length: 3.000000, Petal Width: 2.000000
Sepal Length: 7.900000, Sepal Width: 3.800000, Petal Length: 6.400000, Petal Width: 2.000000
For the Method, Recall= 87.500000
```

Fig 14. Result obtained by running Mahalanobis distance algorithm on Iris plant dataset in MATLAB

Fig 15 shows corresponding visualization.

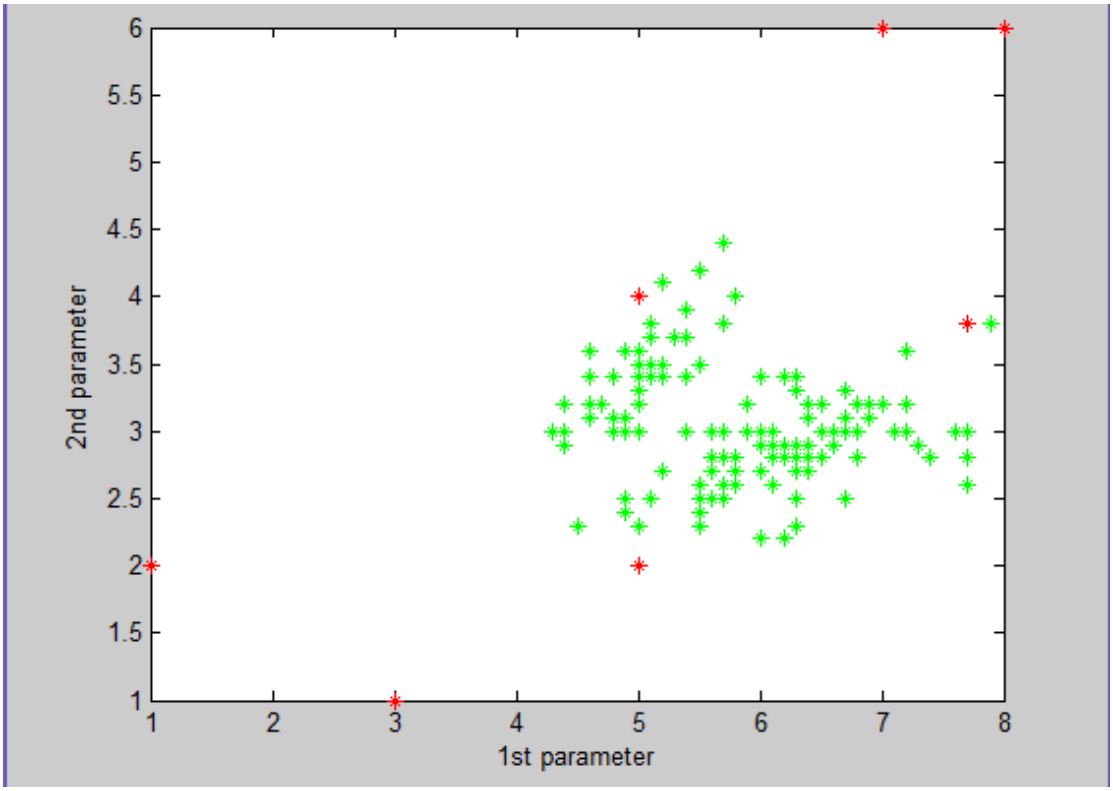


Fig 15. Plot depicting normal points in green and outliers in red of Iris Plant dataset obtained by using Mahalanobis Distance algorithm

4.3.4 Mining distance based outliers using K nearest neighbor criteria (Euclidean distance)

```
Enter the value of k:4
Following are the outliers distance:
    4.0175
    3.6926
    3.5875
    2.9715
    2.8443
    2.7464
    2.0863
    1.1180
Following are the outliers points:
    65   150    24    97     8    30   120   107
Sepal Length: 8.000000, Sepal Width: 6.000000, Petal Length: 2.000000, Petal Width: 1.500000
Sepal Length: 3.000000, Sepal Width: 1.000000, Petal Length: 2.000000, Petal Width: 2.000000
Sepal Length: 1.000000, Sepal Width: 2.000000, Petal Length: 2.000000, Petal Width: 0.500000
Sepal Length: 3.000000, Sepal Width: 1.000000, Petal Length: 5.000000, Petal Width: 1.500000
Sepal Length: 7.000000, Sepal Width: 6.000000, Petal Length: 3.000000, Petal Width: 0.400000
Sepal Length: 5.000000, Sepal Width: 4.000000, Petal Length: 5.000000, Petal Width: 0.300000
Sepal Length: 5.000000, Sepal Width: 2.000000, Petal Length: 3.000000, Petal Width: 2.000000
Sepal Length: 4.900000, Sepal Width: 2.500000, Petal Length: 4.500000, Petal Width: 1.700000
For the Method, Recall= 87.500000
```

Fig 16. Result obtained by running Euclidean distance algorithm on Iris plant dataset in MATLAB

Fig 17 shows corresponding visualization.

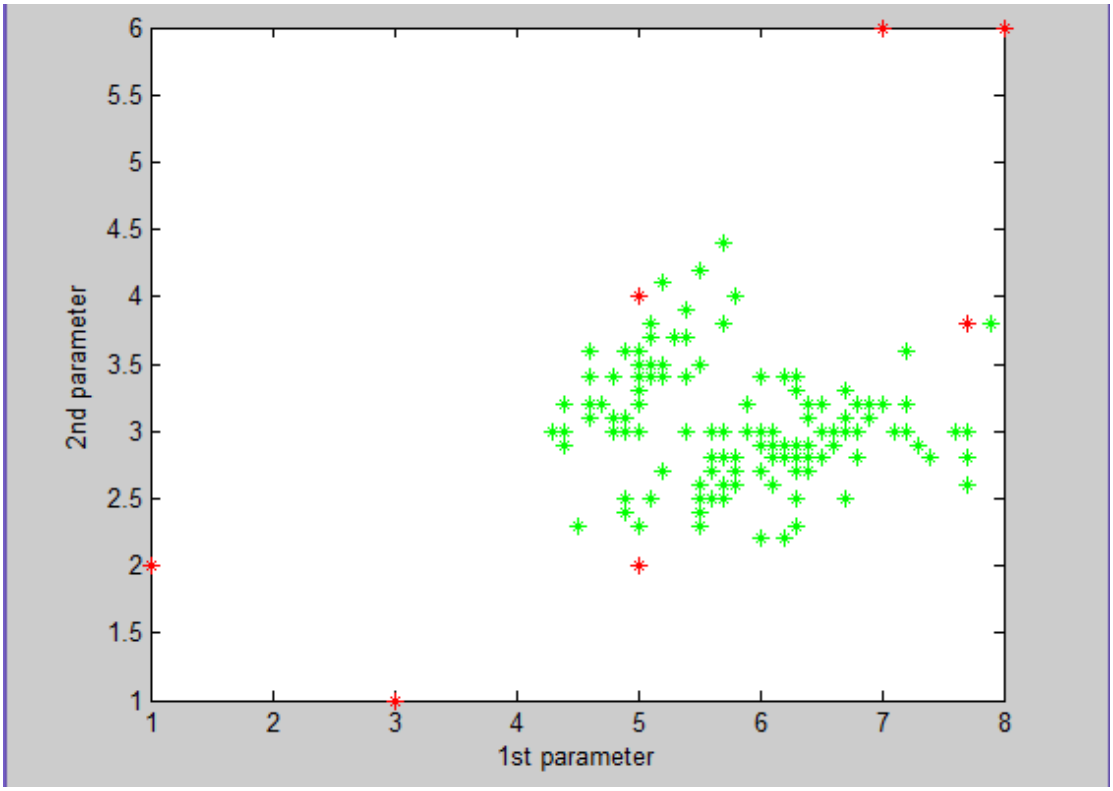


Fig 17. Plot depicting normal points in green and outliers in red of Iris Plant dataset obtained by using Euclidean Distance algorithm

4.3.5 Mining distance based outliers in near linear time using randomization

```
Enter a file name:iris
Enter the value of k:4
Following are the outlier points:
    24    65    97   150     8    30   120   118
Corresponding values are as follows:
    1.0000    2.0000    2.0000    0.5000
    8.0000    6.0000    2.0000    1.5000
    3.0000    1.0000    5.0000    1.5000
    3.0000    1.0000    2.0000    2.0000
    7.0000    6.0000    3.0000    0.4000
    5.0000    4.0000    5.0000    0.3000
    5.0000    2.0000    3.0000    2.0000
    7.7000    3.8000    6.7000    2.2000
For the Method, Recall= 87.500000
```

Fig 18. Result obtained by running Randomization & Pruning algorithm on Iris plant dataset in MATLAB

Fig 19 shows corresponding visualization.

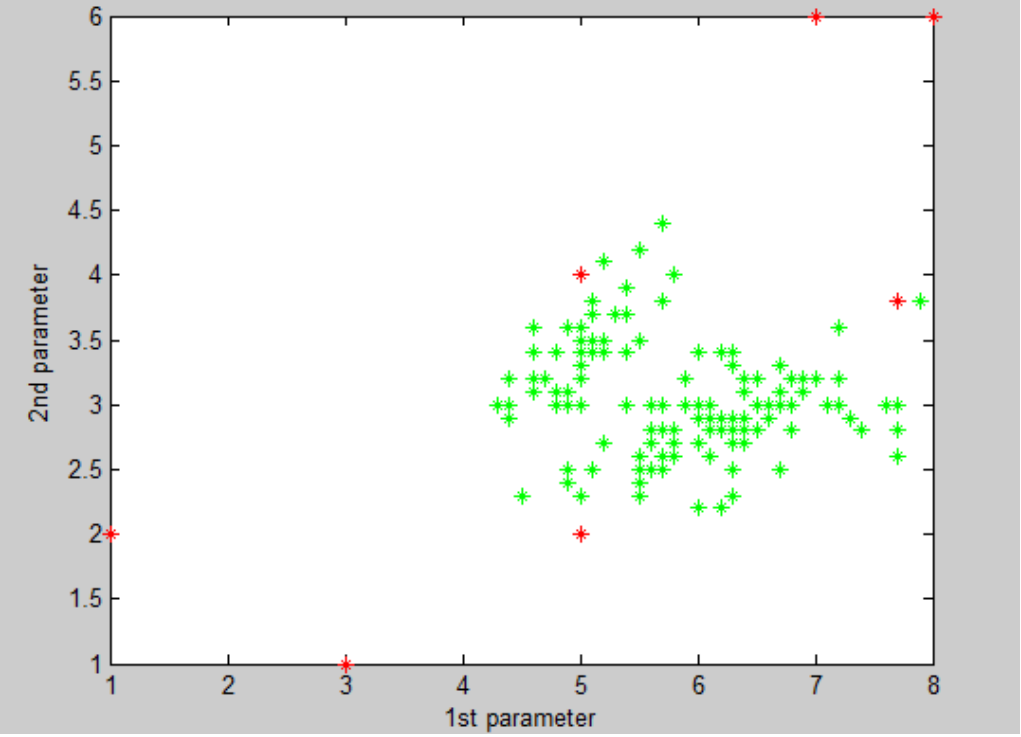


Fig 19. Plot depicting normal points in green and outliers in red of Iris Plant dataset obtained by using Randomization & Pruning algorithm

4.3.6 Bayesian Network based Outlier Detection for discrete nodes using conditional probability tables:

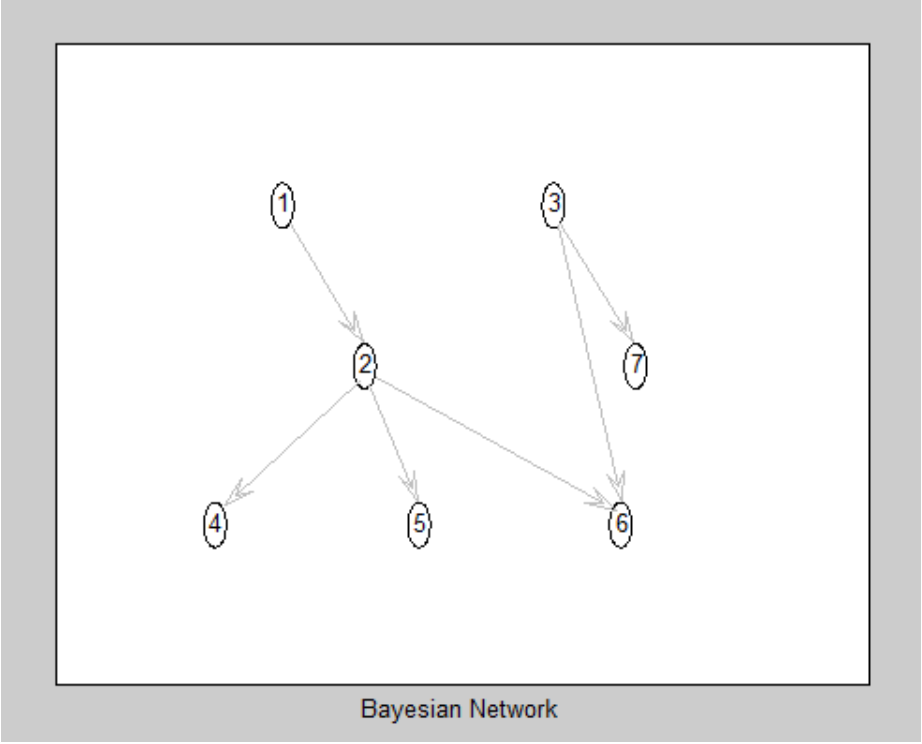


Fig 20.Bayesian Network for Effects of Smoking example generated in MATLAB

```
Joint Probability for different conditions are as follows:

probability =

    0.0016
    0.0000
    0.0000
    0.0000
    0.0000
    0.0017
    0.0000
    0.0000
    0.0000
    0.0004
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000

Logic: Lower probability means that the condition has lower chance of occurrence and hence an outlier
Top outliers have probability

outliers =

  1.0e-006 *

    0.0078
    0.0078
    0.1745

Index: 1 for false and 2 for true
Corresponding conditions are as follows:

values =

     2     2     2     1     1     1     1
     2     2     2     1     1     1     1
     1     2     2     2     1     1     1
```

Fig 21. Result obtained by running Bayesian network based outlier detection algorithm for discrete nodes in MATLAB

4.3.7 Bayesian Network based Outlier Detection for discrete and continuous nodes where continuous nodes follow Gaussian distribution using parameter learning

```
Probability for different conditions are as follows:

probability =

    9.2098
    0.0000
    0.1718
    0.0234
    6.9402
    0.0000
    0.0000
         0
    0.0300
    0.0645
    0.0000
    0.0000
    0.0000
    0.0000
    8.6762
    0.0000
    0.0000
    0.0015
    0.0002
    0.0000
    0.0194
    0.0000
    0.0000
    0.0000
    0.0898
    0.0033
    1.7015
    0.0000
    0.0002
         0

Logic: Lower probability means that the condition has lower chance of occurrence and hence an outlier
Top outliers have probability

outliers =

  1.0e-033 *

         0
         0
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.1631
```

```
Corresponding conditions are as follows:

values =

    7.0000    6.0000    3.0000    0.4000    1.0000
    5.0000    4.0000    5.0000    0.3000    1.0000
    1.0000    2.0000    2.0000    0.5000    1.0000
    8.0000    6.0000    2.0000    1.5000    2.0000
    5.0000    3.3000    1.4000    0.2000    2.0000
    3.0000    1.0000    2.0000    2.0000    3.0000
    3.0000    1.0000    5.0000    1.5000    2.0000
    5.0000    2.0000    3.0000    2.0000    3.0000

For the Method, Recall= 87.500000
```

Fig 22. Result obtained by running Bayesian network based outlier detection algorithm for discrete nodes in MATLAB

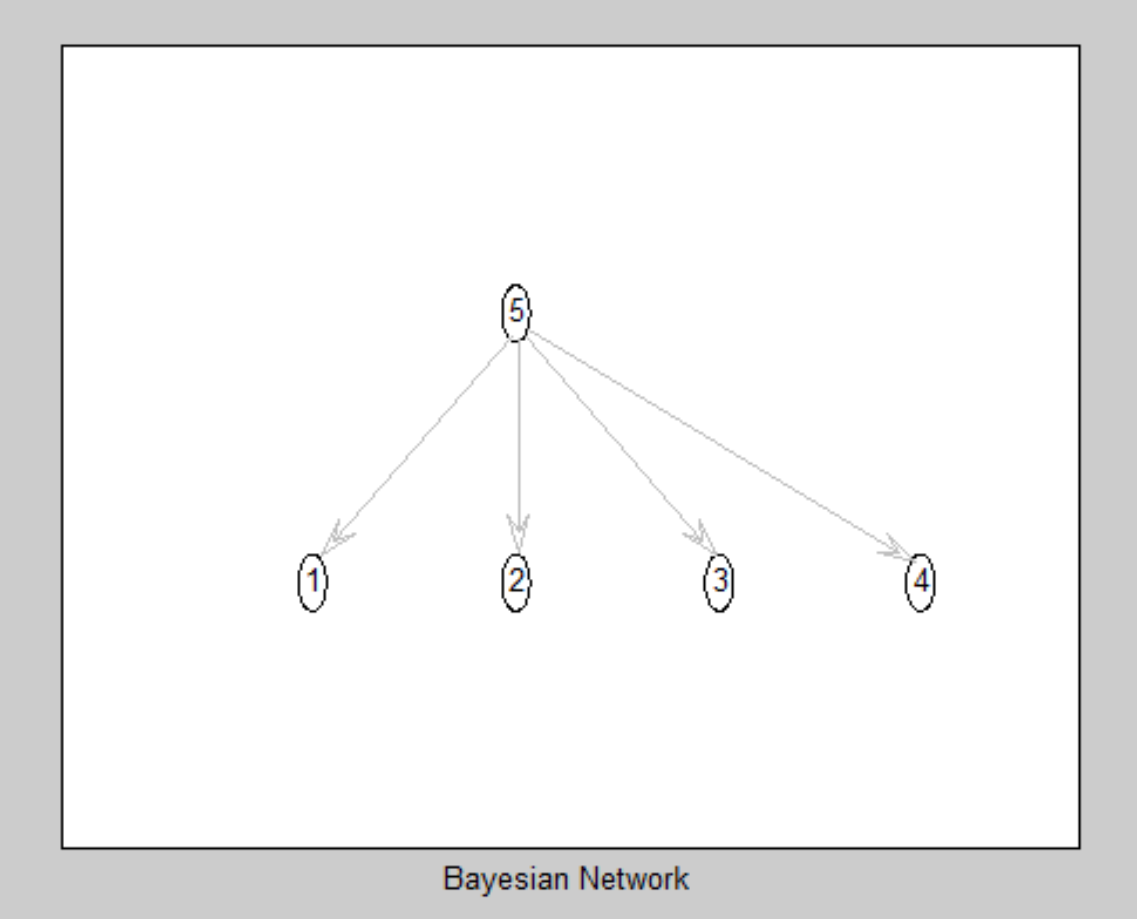


Fig 23. Bayesian Network for Iris Plant data set example generated in MATLAB

CHAPTER 5
Future Work and Conclusion

This project has discussed different ways in which the problem of anomaly detection has been formulated in the literature, and has attempted to provide an overview of the huge literature on various techniques.

This project has implementations of various well known outlier detection algorithms for use by people dealing in data mining. Different approaches for outlier detection namely Statistical based approach, Distance based approach and Conditional Anomaly Detection have been covered under this project. Various algorithms have been implemented namely Univariate outlier detection using Boxplot, Multivariate outlier detection using Mahalanobis Distance Measure, Multivariate outlier detection using Euclidean Distance Measure, Multivariate outlier detection using Randomization and Pruning, Bayesian Network based outlier detection for discrete nodes using conditional probability tables and Bayesian Network based outlier detection for discrete and continuous nodes using parameter learning. The project also includes a graphical user interface for the suite in MATLAB. In addition to above, the recall for above stated algorithms has been calculated and depicted. The suite also has an implementation for visualization of outliers detected using various techniques.

The algorithms implemented in this project have a lot of future scope and can be applied to various data sets to detect the outliers. A lot of work is being done in this field and will continue to grow as the time progresses. The Outlier Detection Algorithm Suite that has been developed can be used by people dealing in data mining and also for general purposes to detect outliers in data sets.

References

1. Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM Comput. Surv.* 41, 3, Article 15 (July 2009), 58 pages. DOI=10.1145/1541880.1541882 http://doi.acm.org/10.1145/1541880.1541882

2. Edwin M. Knorr, Raymond T. Ng, and VladimirTucakov.2000. Distance-based outliers: algorithms and applications. *The VLDB Journal* 8, 3-4 (February2000),237253. DOI=10.1007/s007780050006 http://dx.doi.org/10.1007/s007780050006

3. Stephen D. Bay and Mark Schwabacher. 2003. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining* (KDD '03). ACM, New York, NY, USA, 29-38. DOI=10.1145/956750.956758 http://doi.acm.org/10.1145/956750.956758

4. Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. 2000. LOF: identifying density-based local outliers. *SIGMOD Rec.* 29, 2 (May 2000), 93-104. DOI=10.1145/335191.335388 http://doi.acm.org/10.1145/335191.335388

5. Xiuyao Song, Mingxi Wu, Christopher Jermaine, and Sanjay Ranka. 2007. Conditional Anomaly Detection. *IEEE Trans. on Knowl. and Data Eng.* 19, 5 (May 2007), 631-645. DOI=10.1109/TKDE.2007.1009 http://dx.doi.org/10.1109/TKDE.2007.1009

6. Antonio Cansado and Alvaro Soto. 2008. UNSUPERVISED ANOMALY DETECTION IN LARGE DATABASES USING BAYESIAN NETWORKS. *Appl. Artif. Intell.* 22, 4 (April 2008), 309-330. DOI=10.1080/08839510801972801 http://dx.doi.org/10.1080/08839510801972801

7. http://www.physics.csbsju.edu/stats/box2.html

8. Edwin M. Knorr and Raymond T. Ng. 1999. Finding Intensional Knowledge of Distance-Based Outliers. In *Proceedings of the 25th International Conference on Very Large Data Bases* (VLDB '99), Malcolm P. Atkinson, Maria E. Orlowska, Patrick Valduriez, Stanley B. Zdonik, and Michael L. Brodie (Eds.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 211-222.

9. Victoria Hodge and Jim Austin. 2004. A Survey of Outlier Detection Methodologies. *Artif. Intell. Rev.* 22, 2 (October 2004),85-126. OI=10.1023/B:AIRE.0000045502.10941.a9 http://dx.doi.org/10.1023/B:AIRE.0000045502.10941.a9

10. Stephen J. Chapman. 2001. *MATLAB Programming for Engineers (2nd Edition)* (2nd ed.). Brooks/Cole Publishing Co., Pacific Grove, CA, USA.

11. Aggarwal, C.C.A. (2011). Introduction to Outlier Analysis. IBM T. J. Watson Research Center, Yorktown Heights, NY, USA: Kluwer Academic Publishers.