POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Anomaly Detection via Isolation Forest Embedding

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Author: **Manuel Salamino**

Student ID: 926959
Advisor: Prof. Giacomo Boracchi
Co-advisors: Filippo Leveni
Academic Year: 2020-21

Abstract

Data mining, nowadays, is becoming increasingly important because of the amount of data produced every day. Data Mining is the process of extract information from (large) datasets, with the aim of discover patterns and/or transform the information into a comprehensible structure for further use. Among its many applications, a particularly interesting one is Anomaly Detection: it consists on finding unusual patterns or rare observations in a set of data. Usually anomalies represent negative events, in fact anomaly detection is used in many different fields, from medicine to industry.

The problem can be solved by using different classes of techniques, we faced the problem by taking as starting point a milestone Anomaly Detection algorithm: Isolation Forest. With the introduction of a new embedding space, created starting from Isolation Forest, we have the aim of represent the data in a different way and find an algorithm with performance better than the starting point. The proposed embedding replaces the last Isolation Forest operations, with the aim of execute more complex ones that can capture new data features.

Our empirical evaluation shows that our approach, performs just as well, and sometimes better, than Isolation Forest on the same data. But the most important result is the creation of a new framework to enable other techniques to be used to improve the anomaly detection performance.

Abstract in lingua italiana

Data mining (in italiano "estrazione di dati"), oggigiorno, sta diventando sempre più importante a causa della quantità di dati prodotti giornalmente. Il processo di Data mining consiste nell'estrazione di informazioni da grandi quantità di dati. L'obiettivo è quello di trovare pattern nei dati e trasformare le informazioni contenute in modo da dargli una struttura che possa essere comprensibile anche in utilizzi successivi. Data mining ha applicazioni in molti ambiti, una molto interessante è l'anomaly detection (in italiano "identificazione delle anomalie): consiste nel, dato un insieme di dati, trovare pattern inusuali o singole osservazioni che siano fuori dall'andamento generale. Solitamente le anomalie rappresentano eventi negativi ed infatti le tecniche di anomaly detection vengono utilizzate in molti ambiti (dalla medicina all'industria).
Per risolvere il problema dell'anomaly detection possono essere utilizzati diversi tipi di tecniche, la nostra ricerca ha come punto di partenza una pietra miliare dell'anomaly detection: Isolation Forest. Con l'introduzione di un nuovo spazio di embedding, creato partendo da Isolation Forest, l'obiettivo è quello di rappresentare i dati in una maniera differente e trovare un algoritmo che abbia risultati migliori rispetto al punto di partenza. La nuova rappresentazione mediante l'embedding, sostituisce le ultime operazioni di Isolation Forest, con l'obiettivo di eseguirne di più complesse, capaci di catturare nuove caratteristiche dei dati.
Attraverso le nostre valutazioni empiriche mostriamo che il nostro approccio si comporta altrettanto bene, e a volte meglio, rispetto ad Isolation Forest sugli stessi dati. Ma il risultato più importante è la creazione di una nuova rappresentazione dei dati su cui posso applicare altre tecniche per migliorare le prestazioni di anomaly detection.

Contents

Introduction

Anomaly Detection (AD) is a Data Mining process and consists on finding unusual patterns or rare observations in a set of data. Usually anomalies represent negative events, in fact anomaly detection is used in many different fields, from medicine to industry.
We faced the problem of AD by taking as starting point a milestone algorithm: Isolation Forest[1] (iForest). It represent a breakthrough since before its arrive, all the AD techniques works by constructing a normal data profile, then test unseen data instances and identify as anomalies the instances that do not conform to the normal profile. iForest present a different approach that explicitly isolates anomalies, rather than profile normal instances, exploiting the anomalies properties for which they are *few* and *different* (are the minority of the data and are far from normal instances).

iForest is an unsupervised model-based method for anomaly detection. The model is based on trees ensemble, each tree is called Isolation Tree (iTree). When the model is built, the prediction of a test instance is made by computing its anomaly score, which is based on the average of the paths covered by the input instance in each iTree before reaching a leaf node.

Our proposed method consists of maintaining the core of iForest, thus the ensemble of trees and its training phase, and changing the prediction of the instance label. Instead of taking the average of the outputs of the iTrees, that is a simple operation, we use these outputs to build an embedding, with the aim of, computing a more complex operation, capture new features of the data and obtain better performances.
The embedding is composed by the histogram representation of the vector made by the outputs of the iTrees, and then, on this new representation are applied known classification or anomaly detection techniques.

This new embedding is the entire focus of our work: what the average of the iForest output represent there and how to exploit this new data representation to increase anomaly detection performance. The proposed solutions regards in fact two proposed techniques that are applied in the embedding space. The first one is an intuition coming from the redefinition of average path length in the embedding space, the second one is a common

anomaly detection technique applied, instead that on original data, on the embedding data.

Redefining the average path length in the embedding we note that the formulation consists in a linear combination of embedding features with predefined weights, so the intuitions is to find a way to obtain weights that can improve the performance. We use Linear Discriminant Analysis and we find that, with new dataset-specific weigths the performance increses, but not for all the dataset, and this is a surprising results because using a supervised approach to find the best weights, we expect better results in all the datasets.

The second one exploit the idea of apply a common used anomaly detection technique to the new framework: One-Class Support Vector Machine. Also in this case we reach results that are better than standard iForest in most of the dataset, even if the differences in most of the times are very small. We also find an interesting characteristic of the embedding, because this technique applied in embedding is much more stable at One-Class SVM parameter tuning than the same technique with the same parameters applied on original data.

Finally, we can summarize the main contributions of this thesis as: the creation of a framework in which we are able to represent data in a totally different way, based on the iForest output, and new algorithms to perform anomaly detection that reach results better than the starting point iForest.

Starting from our work we are sure many future developments can be done, such as the application, in the embedding space, of other anomaly detection or classification techniques, and new embedding definitions can be formulated, using representations that era different from histogram.

Thie thesis is structured as follows:

- In Chapter 1 is provided the background about Anomaly Detection: basic definitions and most common applications. Then we review the literature and we briefly describe the main techniques classified based on their approach.
- In Chapter 2 we describe Isolation Forest, the starting point of our work. It is fundamental to understand our proposed solution and our intuitions.
- In Chapter 3 we present our contributions by explaining in details how the embedding is defined, the proposed solutions and the intuitions.

- In Chapter 4 the empirical evaluation of the solutions presented before, commenting the results.
- In Chapter 5 we summarize the contributions of this work, present our conclusions and point out possible improvements.

1 | Background

In this chapter, we introduce the anomaly detection problem and the main approaches when we face with it. We also describe some state-of-the-art techniques. There is also a specific a technique that will not be mentioned now, together with the others, but it has a special focus in the next chapter since is very important to understand our solution: *Isolation Forest*.
First of all, as a necessary step, we start by defining the concept of anomaly and its characteristics. Then, an overview of the most common techniques.
The structure and the contents of this chapter are inspired from [2] and [3].

1.1. Concept of Anomaly

Before starting the description of the techniques to detect anomalies, we necessarily have to define what is an anomaly.
Anomalies can be defined as *patterns in data that do not conform to a well defined notion of normal behavior*[2]. For what concerns our work we will refer to anomalies also using other words as: *rare observations, anomalous events* or *outliers*, but all of them refers to the same concepts of *instance which differs significantly from the majority of the data*.

1.2. Type of Anomalies

Anomalies can take different forms, and given the kind of data and the context we are working on, the approach to detect them can change.
There are two different classification for anomalies: one about the number of attributes to consider and the other one about the number of instances needed for the detection.
Provide a classification criterion to the anomalies may help, for example on the choice of the algorithm to use for the detection.

The first categorization, about the number of attributes to consider, split the anomalies into two groups:

- *Univariate anomaly*: when it's possible to detect the anomalous behaviour by looking at the values of a single attribute.
- *Multivariate anomaly*: when, to detect an anomalous behaviour, is necessary to look at a combination of values from two or more attributes.

The second categorization is given by the number of instances involved in the anomalous behaviour:

- *Point anomalies*: individual instances which behave as anomalous with respect to all the other instances of the dataset. This is the simplest kind of anomaly (e.g. in credit card fraud detection the anomalies are detected by looking at the *amount spent*).
- *Contextual anomalies*: individual instances which behaves as anomalous with respect to a subset of the dataset which is in the same context. The context is defined by a subset of attributes (e.g. month and year in a time series dataset), called *contextual attributes*, while the anomalous behaviour is detected by looking at the other attributes, called *behavioural attributes*.
  A contextual anomaly is similar to the point anomaly but considering only the subset of data within the same context (same values on contextual attributes), instead of considering the entire dataset (e.g. during holidays may be normal spend a lot on food, not at other times).
- *Collective anomalies*: a group of instances which behaves as anomalous. Every single instance of the group can be classified as normal (is not a point anomaly), but if we take all the instances of the group we can note an anomalous behaviour (e.g. in an ECG a 0 can happen, but if there is a group of consecutive 0s this is an anomaly).

1.3. Anomaly Detection

Anomaly Detection is a data mining process that aims to identify data points that deviate from a dataset's normal behaviour. Anomalous data can represent catastrophic events, frauds in credit card transaction, a structural defect, medical problems or errors in a text. These are only a few examples of what an anomaly can represents. The goal of anomaly detection is to identify and prevent them.

1.3.1. The Problem Formulation

Let's look at the problem of AD in a more formal way[4]. The anomaly detection problem consists in monitoring a set of n data:

$$\mathcal{X} = \{x_i, \ldots, x_n \quad | \quad x_i \in \mathbb{R}^d\},$$

where each element x_i is realization of a random variable having pdf ϕ_0, with the aim of detect outliers, i.e. points that do not conform with ϕ_0.

$$x_i \sim \begin{cases} \phi_0 & \textit{normal data} \\ \phi_1 & \textit{anomalies} \end{cases}$$

where $\phi_0 \neq \phi_1$ and ϕ_0 and ϕ_1 are unknown. Since X has no label we are in an unsupervised setting, and X contains both normal instances and anomalies, furthermore there is no clear distinction between training and test data. Each dataset contains both normal and anomalous samples. The only mild assumption we can make is that normal data far outnumber anomalies.

1.3.2. Challenges of Anomaly Detection

The most intuitive approach to implement anomaly detection is based on defining what is normal: define a region that represent a normal behaviour, and automatically what is outside this region will be defined as anomalous. But working on anomaly detection tasks, several factors come up and make this approach very challenging:

- is very difficult to find a clear boundary that divides perfectly normal and anomalous instances;
- normal behaviour may change in time, and normal actions now, in the future, may be no longer normal;
- noise in the data may change how an observation appear (a normal may appear as an anomaly and vice versa);
- availability of labeled data. Label the observations for anomaly detection tasks is a very expensive procedure, because they require manual operations by a human expert. Moreover is not easy to collect anomalous events, indeed there is a great disproportion in terms of number of normal and anomalous labeled instances.

1.3.3. Categories of Anomaly Detection Algorithms

About the latter point of the previous list there is a further factor to consider: is prohibitively expensive to have an anomalous event for every possible anomalous behaviour. This become even more difficult if we consider the dynamic nature of the anomalous behaviours (new anomalies might arise).

Given this data label problem, we introduce three different approaches in which anomaly detection techniques can operate:

- **Supervised Anomaly Detection**: a supervised technique, in general, consists in a set of labeled instances (aka training set) used by the algorithm to learn how to recognize the correct class (in case of classification problem) of an unseen data. The anomaly detection problem is a classification problem with two classes (aka binary classification problem): *normal* and *anomaly*.

$$TR = \{(X, Y) \quad | \quad x_i \in \mathbb{R}^d, \quad y_i \in \{0, 1\}\}$$

$$TR: \text{ training set} \qquad y_i = \begin{cases} 0 & \textit{normal data} \\ 1 & \textit{anomalies} \end{cases}$$

  If we use a Supervised Technique to solve an anomaly detection problem, we have to deal with the need of labeled data, this means normal instances and anomalous instances labeled correctly by an expert. As said before this is a very hard and very expensive operation, and, due to the rarity of anomalous events, we also face the problem of class imbalance;

- **Semi-supervised Anomaly Detection**: a semi-supervised technique for anomaly detection operate with a training set made only by normal samples.

$$TR = \{(X, Y) \quad | \quad x_i \in \mathbb{R}^d, \quad y_i \in \{0\}\}$$

  The idea is to cover as better as possible the normal region in order to recognize as anomalous the observations outside this area. This is the reason why for this class of techniques the training set is made of normal samples: is impossible to do the same thing with anomalous data (a training set composed only by anomalous samples) because is impossible to cover the region of all the possible anomalous behaviours.

An advantage of this approach over the supervised one is that there is no need of labeled anomalous data;

- **Unsupervised Anomaly Detection**: an unsupervised technique operates with no labeled data, and so with no training set.

$$TR = \{X \quad | \quad x_i \in \mathbb{R}^d\}$$

This because does not need a learning phase, but approaches the problem from a different perspective, for example by exploiting the assumption for which normal instances are far more frequent than anomalies in the test data. We will see some examples in the next sections.

1.4. Applications of Anomaly Detection

Anomaly detection is used in different fields and for different scenarios. Here some applications:

- *Intrusion Detection.* One of the most common use of anomaly detection, the goal is to detect malicious activity in computer networks. The main challenges of this domain are the large amount of data to manage and test, and the online analysis required to detect suspicious activities in real-time. There are two types of Intrusion Detection: *Host-Based*, a malicious sequence of system calls (collective anomaly) to obtain an unauthorized behaviour, and *Network Intrusion*, remote attacks from outside the network in order to obtain an unauthorized access to the network.
- *Fraud Detection.* Detection of criminal activities by a malicious user who want to use the resources provided by a commercial organization (such as banks, insurance agencies, cell phone companies, etc) to a user, in an unauthorized way. Usually, the detection approach consists in monitoring the activity of the users and report when there are some deviations on its behaviours.
- *Medical Anomaly Detection.* Detection of anomalies on patient records in order to find disease. The data can be modeled as a record with patient data or, sometimes, as a time-series data. In the first case we will look for a point anomaly, in the second one for a collective anomaly. This domain, of course, requires a high degree of accuracy, in fact the cost of classifying an anomaly as normal can be very high.
- *Industrial Damage Detection.* Continuous usage of industrial units can lead to serious damage, and the goal is to prevent them by using sensors to monitor processes

and industrial units. There are two further classification for this domain: *Fault Detection in Mechanical Units*, to keep track of the performances of industrial components (modeled as time-series), and *Structural Defect Detection*, to find damage in industrial structures (normal data are typically static over time, so we want to perform change point detection).

- *Image Processing.* In this application the anomalous behaviour can be detected from spatial as well as temporal characteristics. In case of motion detection (videos) the interest is focused on changes in an image over time (e.g. video surveillance), while if the domain is the images we research the anomalous patterns in the appearance of a foreign object or in an instrumentation error (e.g. mammographic image analysis). The anomalies can be either point and contextual anomalies.
  The most challenging aspect of this application is the large size of the input, for example if we are dealing with videos.

- *Text Data.* Detection of novel (or anomalous) topics, or events, in a collection of documents. With the rise of social networks, many companies study the changes in trends and anomalous behavior of users for commercial purposes.
  In this applications the most challenging aspect is the difficulty of managing the differences between two documents which belongs to the same category or topic.

- *Sensor Networks.* One of the most interesting domain, since sensors in networks can collect different types of data and in an online manner.
  The most challenging aspect here are the possibility of sensors fault or, in general, the presence of noise in the collected data, and the need of lightweight anomaly detection techniques, due to the resource constrains in wireless sensors.

- *Other Domains.* Of course there also a lot of other domains, e.g. speech recognition, traffic monitoring, detecting anomalies in biological data, etc.

1.5. Anomaly Detection Techniques

The Anomaly Detection techniques can be grouped into macro-groups depending on the main idea behind the resolution technique. In the following subsections, the techniques are divided according to this criterion.

1.5.1. Classification-Based Anomaly Detection Techniques

Classification algorithms, in general, are used to learn a model able to associate to each sample a specific class, or label.

The algorithms in this category are organized in two steps: *training*, the model is learnt starting from a set of labeled observations, and *testing*, unlabeled instances are classified, hence assigned to a class, or label.
In Anomaly Detection, classification techniques label each sample as *normal* or *anomaly*, more precisely these techniques can be grouped into two further categories: *multi-class* and *one-class*.

Multi-class techniques are used when the training set TR is composed by normal samples coming from more than one class:

$$TR = \{(X, Y) \quad | \quad x_i \in \mathbb{R}^d, \quad y_i \in \{0_a, 0_b, \ldots, 1\}\},$$

where $0_a, 0_b, \ldots$ represent the normal classes and 1 the anomaous one.
Once the training stage is completed, the classification procedure will be the following: the algorithm produce a score for each class, which is the confidence for a sample to belong to that class, if this value is higher than a fixed threshold we assign the sample to that specific class, otherwise, if no classes reach the threshold, the samples is labeled as anomaly.

Instead, *one-class* techniques are used for datasets with only one normal class:

$$TR = \{(X, Y) \quad | \quad x_i \in \mathbb{R}^d, \quad y_i \in \{0, 1\}\}.$$

In this case the classification is easier, because there is only a confidence value to check, since the normal class is only one, and, again, if the confidence is not sufficient to classify the sample in that class it means that we are testing an anomalous one.
Sometimes, in one-class techniques, the test is not based on *how much a sample belong to the normal class*, but the anomaly score is computed, which represents *how much a sample is anomalous*.

Neural Networks

Neural Networks is a model inspired to biological neural networks of animal brains. It can be applied for anomaly detection, both in multi-class and one-class datasets.
The steps to follow are the ones described before: training, in which the neural network learns, by finding the optimal parameters, how to distinguish normal instances from anomalies, and testing, where an instance is provided as input to the Neural Network and the predicted label is given as output.
The Neural Network represented in Figure 1.1 is a basic version and is composed by the following layers:

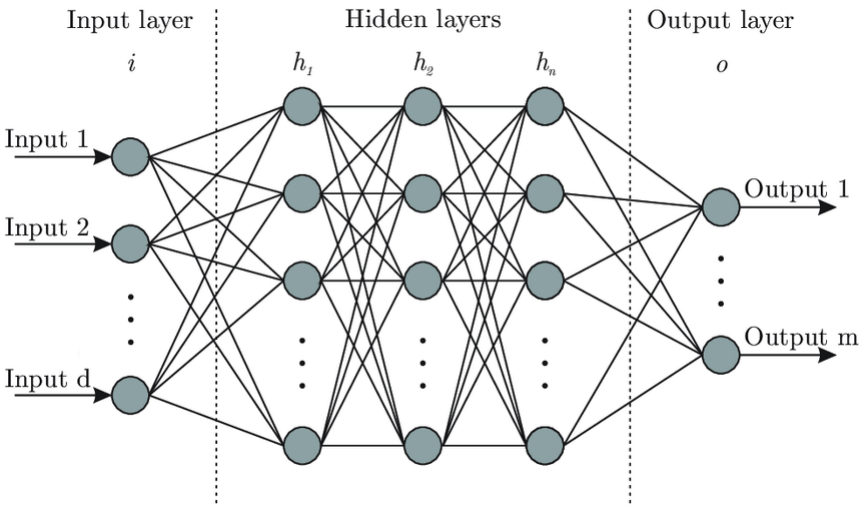


Figure 1.1: Neural Network example[5].

- *input layer*: take the input data $x \in \mathbb{R}^d$. It has d input fields, one for each data feature;
- *hidden layers*: intermediate levels which propagate the modified input by adding other operations. There are an arbitrary number of intermediate layers;
- *output layer*: the last layer of the model, it has an output for each class with the relative prediction confidence.

The edges between nodes of different levels represent weighted connections. The value of each node is multiplied by a weight when it is passed to the following connected node. So, all inputs are modified by a weight and summed. This operation is a linear combination, but then, an *activation function* controls the amplitude of the output (e.g. usually output range is [0, 1]) and add non-linearity.

Replicator Neural Networks

A Replicator Neural Network[6] (RNN) is a special Neural Network (NN) variants, in fact in this case the NN works as an *autoencoder*: the model is characterized by an input and an output layer with the same number of neurons, and three intermediate hidden layers with fewer neurons each; the input, from a high dimension representation, is turned in a lower dimension representation, and then it turns, again, to the initial dimension, with the goal of reconstruct as well as possible the input.
The only difference with autoencoders is in the activation function used in the compression layer: autoencoders use linear activation function, while RNNs use step function.
An example of RNN is shown in Figure 1.2.

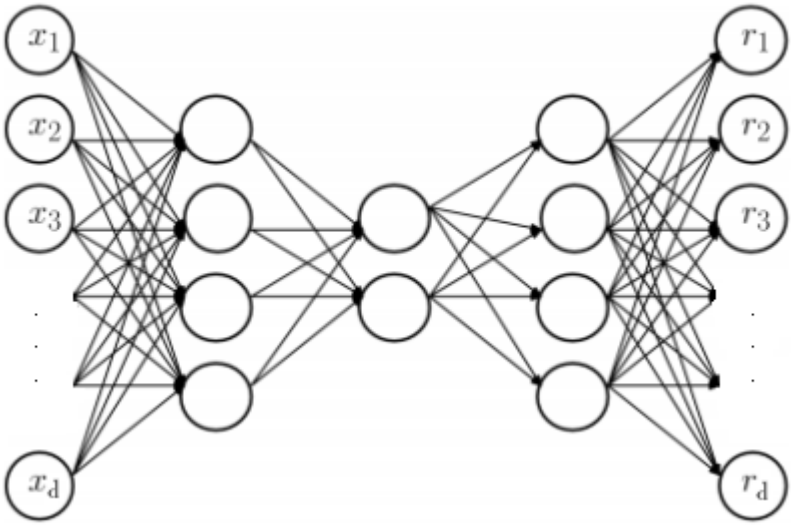


Figure 1.2: Replicator Neural Network example[7].

Of course this reconstruction may be not perfect and in fact the *reconstruction error* δ for the instance x is computed as follows:

$$\delta = \frac{1}{d}\sum_{j=1}^{d}(x_j - r_j),$$

where d is the number of features and r is the reconstruction computed by the RNN.
The goal of the training phase is to find the best weights for the NN in order to obtain the lower reconstruction error in training data.
Since it is a semi-supervised technique, the training set is composed only by normal instances. The tested instances return a reconstruction error, which is used as anomaly score, where high values indicate higher probability of being an anomaly.

One-Class SVM

One-class SVM (OC SVM) is a specific application of Support Vector Machines (SVMs) and, as its name implies, it works only with one-class training data.
It's a semi-supervised technique, in fact the training set is composed only by normal samples, and the OC SVM learns, in the feature space, the boundary (a region) that contains normal data. It is computed in the feature space because a linear separation in the feature space corresponds to a variety of non-linear boundaries in the original space, and it is defined by using a few normal data, called *support vectors*.

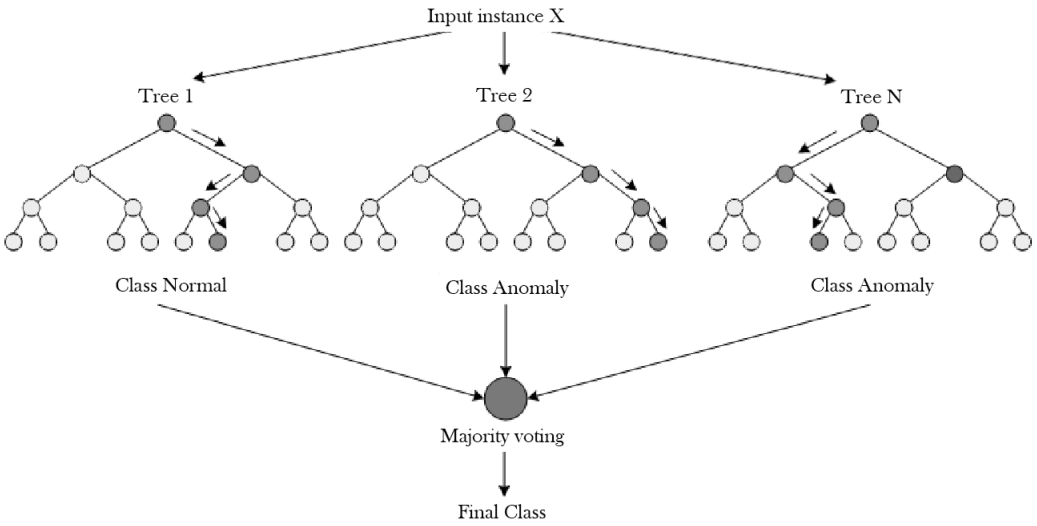


Figure 1.3: Random Forest example[10].

The classification function[8] is:

$$f(x) = sign((w \cdot \psi(x)) - p), \qquad \psi(x): \text{ feature of x}$$

$$f(x) = \begin{cases} 1 & \textit{label } x \textit{ as normal} \\ -1 & \textit{label } x \textit{ as anomaly} \end{cases}$$

w and p are the weights, and are optimized to obtain a classification function f that is positive (returns 1) on most of training samples.
When the training phase is complete, so the normal region is well defined (i.e. w and p are optimal), the labels are assigned to the test data: if they fall inside the boundary are labeled as normal, if outside as anomalies.

Random Forest

Random Forest[9] is an ensemble learning method used for classification and works constructing a forest of decision trees.

The forest is an ensemble of Decisions Trees built at training time. For anomaly detection tasks, as for classification ones, the predicted label of a specific test instance is given by majority voting the output of the trees. Figure 1.3 shows how it works.

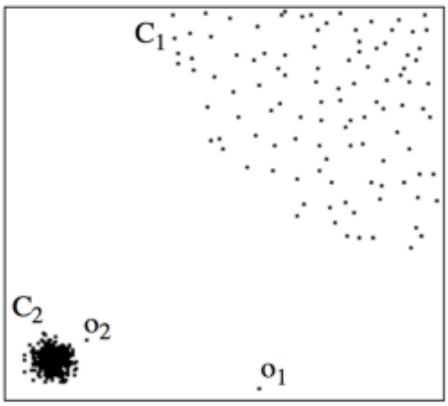


Figure 1.4: Normal regions with different densities.

1.5.2. Nearest Neighbor-Based Anomaly Detection Techniques

Nearest Neighbor-Based Techniques are based on the concept of distance (or similarity) between instances. It is therefore necessary to define it.
The distance (or similarity) can be computed in different ways and it also depends on the attributes type (e.g. Euclidean distance for continuous attributes, matching coefficient for categorical ones, and so on).
These techniques relies on the idea that normal instances are located in dense areas, while anomalies are located far from other instances.
There are two possibilities to compute the anomaly scores of a test instance, it can be based on its *distance from the k-th nearest neighbor* or on its *relative density*.

Basic Density-Based Technique

In general, the idea of the basic density-based technique is to estimate the density of the neighborhood of each data instance, and an instance that lies in a neighborhood with low density is declared to be anomalous, while an instance that lies in a dense neighborhood is declared to be normal.
The problems of the basic technique come out when there are more than one normal regions with different densities. Let's consider the example scenario in Figure 1.4[11].

There are two clusters C_1 and C_2 with different densities, if we consider the density of C_1, the distance between each instance of C_1 and its nearest neighbor is greater than the distance between the instance o_2 and the nearest neighbor from the cluster C_2. Thus the instance o_2 will not be considered as anomaly and the basic technique will fail to distinguish o_2 anomalous with respect to C_1. However, the instance o_1 may be detected.

LOF

Local Outlier Factor[11] (LOF) mitigate this problem by considering the local density of the test instance and the local density of its k-nearest neighbors.
Define *k-distance(p)* as the distance of the instance p to its *k-th* nearest neighbor, and the set of k-nearest-neighbors, indicated as *k-neighborhood(p)*. The *k-neighborhood(p)* includes all objects within that distance and can contains more than k objects, if there are more than one element at distance *k-distance(p)*.
The algorithm starts computing the reachable distance to each k-nearest-neighbor as:

$$\textit{reach-dist}_k(p, o) = max(\textit{k-distance}(o), d(p, o))$$

Then, compute the local reachability density of p:

$$lrd_k(p) = \left(\frac{\sum_{o \in \textit{k-neighborhood(p)}} \textit{reach-dist}_k(p, o)}{|\textit{k-neighborhood(p)}|} \right)^{-1}$$

Finally, the LOF is the average of the ratio between the LRD of the neighbors and the LRD of instance p:

$$LOF_k(p) = \frac{\sum_{o \in \textit{k-neighborhood(p)}} \frac{lrd_k(o)}{lrd_k(p)}}{|\textit{k-neighborhood(p)}|}$$

ODIN

A simple density-based method: the anomaly score is computed as the inverse of a quantity called ODIN.
ODIN - Outlier Detection using In-degree Number - is equal to the number of k-nearest neighbors of the test instance which have that instance as one of the k-nearest neighbors. This algorithm need big dataset since it counts the number of neighbors, and to distinguish normal and anomalies, the difference between their number of neighbors should be high.

ORCA

ORCA is a distance-based method with the boast of being able to cut down the complexity of $\mathcal{O}(n^2)$ to near linear run time.
A fundamental element is the cutoff value: when an instance's anomaly score is predicted, if it is lower than the cutoff, this instances is removed from the dataset (cannot be an

anomaly) and also all its nearest neighbors are removed (are nearest neighbor of a normal instance). With this simple pruning procedure the execution time of the algorithm receive a boost.
If a lot outliers are found, the cutoff becomes higher and the pruning is more efficient. ORCA is able to overcome the computational complexity limitation of nearest neighbor algorithms if the algorithm can find anomalies and update its cutoff value; if the dataset contains few anomalies ORCA is not able to work efficiently and it's near-quadratic.

1.5.3. Clustering-Based Anomaly Detection Techniques

In general, Clustering techniques are used to group similar data instances into bunches called *clusters*. We can use this category of techniques for anomaly detection (with some adaptions), even if they are originally developed for a different purpose.
Similar to nearest neighbors techniques are based on the concept of distance (or similarity), the main difference is that nearest neighbors techniques use these metrics directly to compute the anomaly score, while clustering techniques as first step divide the data into clusters, so this is the first purpose of the metrics, and then, using clusters division, compute the anomaly score for each instance.
We aspect normal instances grouped in cluster(s), while anomalies sparse and far from the clusters centroid. Unfortunately in some cases this not happens and, for example, anomalies form small clusters, and this make the recognition more difficult.

FindCBLOF
The proposed technique assign to each data instance an anomaly score called *CBLOF - Cluster-Based Local Outlier Factor*. CBLOF is a metric used after a clustering algorithm is applied.
First of all a clustering method is applied to data, then the clusters are distinguished in small and large (using a threshold). When a new instance is tested, it is associated to a cluster and then, its score is computed as follows:

- if it is labeled as part of a large cluster, the anomaly score is the size of the assigned cluster multiplied to the distance to the cluster centroid;
- otherwise, if it is assigned to a small cluster, the anomaly score is the size of the assigned (small) cluster multiplied to the distance to the nearest large cluster.

Therefore we can note that is fundamental the distinction between small and large clusters, in fact the knowledge of the domain is required to decide the correct value for the threshold.

1.5.4. Statistical Anomaly Detection Techniques

Statistical Techniques are based on the principle that *normal data distribution* ϕ_0 is known (or inferred).
In fact, the main idea is that normal instances lie on high probability regions of the stochastic model, while anomalies occur in low probability regions.
As said before, ϕ_0 can be given, and we use *parametric techniques*, or can be inferred from data, and we use *non-parametric techniques*.

Gaussian Model-Based
The assumption to use this technique is that the data are generated from a Gaussian distribution:

$$X \sim \mathcal{N}(\mu, \sigma^2)$$

where μ is the mean value and σ^2 is the variance. Of course this is a parametric technique. Once we know the data distribution, the anomaly score is the distance between the tested data and the mean value μ of the distribution.

There are several criteria that allow us to distinguish if the tested data is an anomaly or not and are based on the choice of the threshold to distinguish anomalies and normals. Some criteria example are:

(i) normal data lie in the $\mu \pm 3\sigma$ interval

(ii) the *box plot rule*, uses box plot attributes to define the normal region. First of all, some definitions: Q_1, the lower quartile, Q_3, the upper quartile and the *Inter Quartile Range* $IQR = Q_3 - Q_1$. The normal region is defined between $Q_1 - 1.5IQR$ and $Q_3 + 1.5IQR$.

These two criteria, respectively, contain the 99.7% and 99.3% of observations into the defined normal region.
There are also criteria based on statistical tests as Grubb's test, student's-t test or χ^2-test.

Regression Model-Based
Such technique is defined by the usual two consecutive steps: the regression model is fitted on data and the test instance is verified. Its residual (the distance between model prediction and real value) is used as anomaly score.
In fact the residual part can be seen as the part of the test instance which is not explained

by the regression model, and, higher is this part, higher is the probability that the instance is anomalous.
Regression techniques are widely used for time-series data, in fact there are regression techniques that can be applied to specific time-series model, as ARMA or ARIMA models.

Histogram-Based
This is the simplest non-parametric statistical technique and consists in profile normal data using histogram(s).
For univariate data is composed by two steps: create an histogram based on the only attribute of training data, then prediction of the test instances is computed according to the height of the bin in which they fall, since it represents the frequency.
For multivariate data, during the training phase attribute-wise histograms are built, while for testing, each attribute falls in a bin and the partial anomaly score is computed according to the bin height, then the total anomaly score is computed by aggregating the partial ones.

Kernel Function-Based
The kernel-based techniques works similar to the parametric ones, the only difference is the need of a density estimation phase.
Parzen windows estimation is a non-parametric technique for density estimation. Once we have the density, we check the test instance and if lies in a low probability area is declared as anomalous.

1.5.5. Information Theoretic Anomaly Detection Techniques

Information Theoretical algorithms find outliers relying on the idea that anomalies induce irregularities in the information content of the dataset. In fact the base concept is that an anomaly point introduce more complexity than a normal one in the description of the data.
Several measures are available, as Kolmogorov Complexity and entropy. The choice of the measure is however critical: it must be sensitive enough to detect the change in complexity induced by very few anomalies in very large datasets.
These techniques are a good alternative to statistics, because they measure the fit of an instance to the rest of the dataset without having to assume a distribution.

Basic Technique

Assume $\mathcal{C}(X)$ as the complexity of dataset X, the basic technique find the minimal subset of instances I, such that $\mathcal{C}(X) - \mathcal{C}(X - I)$ is maximum. The instances of the solution subset I are denoted as the anomalies of the dataset.
The complexity function $\mathcal{C}(\cdot)$ can be one of the available measures (e.g. entropy).
Since this technique involves two optimization problems simultaneously (minimize the subset while maximizing the complexity reduction), an approach in which every possible subset of the dataset is tested would be too much expensive, hence some approximation techniques are used, e.g. Local Search Algorithm is a widely used heuristic for this kind of problems and allows a transition from exponential to linear time.

1.5.6. Spectral Anomaly Detection Techniques

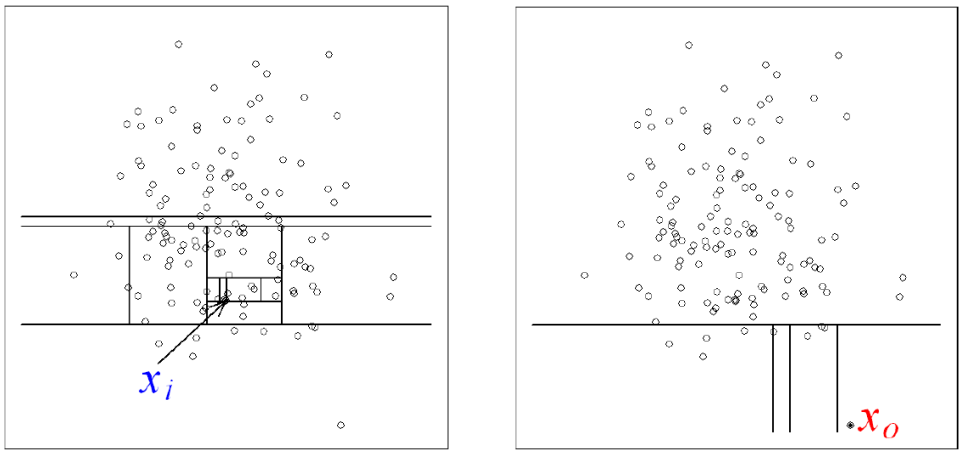
The idea of this class of algorithms is to embed data in a lower dimensional subspace in which normal and anomalies appear significantly different.
These techniques can works both in semi-supervised and unsupervised settings.

Principal Component Analysis
Approximate data in a lower dimensional subspace by maintaining only the directions in which data expresses higher variance.
In fact PCA perform a change of basis by creating a new orthonormal basis in which each component is the best fit along one direction, of course maintaining the orthogonal restriction, and only the first n components are mantained.
An example of technique to perform anomaly detection starting from PCA is measure the data variation along the components with low variance, because are the most sensitive to variations caused by outliers.

2 | Isolation Forest

We have seen an overview of the main applications and methods for Anomaly Detection. Now we will focus our attention on a specific algorithm: *Isolation Forest*[1].
The reason of this focus is that our work is based on this algorithm and we want to improve it by maintaining its core and changing only the last operations, in order to achieve better results in the same conditions.

Isolation Forest (*iForest*) is an unsupervised model-based method for anomaly detection. The first thing we can say about this algorithm, which differentiates it from all the previous ones, is that it relies on the idea of directly isolate anomalies, instead of profiling the normal behaviour and label as anomalies the instances far from this profile.
The idea of this approach comes from the assumption for which the anomalies are *few and different*, and therefore they are more susceptible to isolation.
The fact that an anomaly is more susceptible to isolation than a normal instance can be visualized in the example in Figure 2.1[1], in which we can note that an normal instance requires much more partitions to be isolated.



(a) Isolating normal. (b) Isolating anomaly.

Figure 2.1: (a) Normal point x_i requires twelve partitions to be isolated; (b) Anomaly x_0 requires only four partitions to be isolated.

The model is based on a trees ensemble, in which each tree is called *Isolation Tree*.
The nature of the ensemble, together with the idea of an anomaly-oriented approach, allows the use of sub-sampling (more details in Section 2.3). A trees ensemble fits perfectly the idea of sub-sampling because we have the opportunity to train each tree using a different sub-sampling of the dataset and the isolation of anomalies can be performed using a small amount of data (unlike normal data profilation which requires a large amount of data).
These two factor combined create an algorithm which has a linear time complexity with a low constant and a low memory requirement, in addition to excellent performance.

Now let's see, in the next sections, how the trees are built and how they cooperate into the ensemble in order to distinguish anomalous and normal instances.

2.1. Isolation Tree

An Isolation Tree (iTree) has the same structure of a Proper Binary Tree, a tree in which each node belongs exactly to one of the following categories:

- *external node*, a leaf node, no child;
- *internal node*, a node with exactly two children nodes (right and left).

We build an iTree starting from a subset X of the dataset, because of sub-sampling.

$$|X| = \psi \qquad\qquad x_i \in \mathbb{R}^d$$

with ψ sub-sample size and d number of features.

To build an iTree, as well as for a Binary Tree, we recursively divide X using as *splitting criterion* to divide data into right and left child. The *splitting criterion* consists in randomly selecting an attribute q (among the d possible) and a split value v in order to test if $q < v$. All the instances for which the test returns True are put in the left child, whereas all the others in the right one. The *split value* of p is randomly selected in the range of values of v in X.
The splits are repeated until one of the following conditions is reached:

(i) the tree reaches a height limit l;

(ii) $|X| = 1$;

(iii) all data in X have the same value.

By following these rules and the termination criterion we build an iTree.

Memory requirement. In the worst case all instances are distinct, hence the number of external nodes is n and the number of internal nodes is $n-1$. The total number of nodes is bounded at $2n-1$ for the worst case and this means that the memory requirement grows linearly with n.

2.2. Anomaly Score

The *anomaly score* is a number in $(0, 1]$ that measures how much a test instance is an anomaly. In order to label an instance as anomalous or normal we have to set a threshold, usually is set to 0.5, and if the anomaly score is higher than that the instance is labeled as anomalous, otherwise as normal.
Before defining the anomaly score, we have to define the **Path Length** $p(x)$ of given instance x as the number of edges traversed by x from the root node to the external node reached following the iTree rules defined in the training phase.
It is the fundamental value for the anomaly score computation, since the main idea behind iForest is the one explained before: an anomaly is easier to isolate than a normal instance. In terms of iTrees, the expected behaviour of normal instances is to exit the iTrees from the deepest leaves, while anomalies are expected to exit from the shallowest ones. In terms of path length this results in: longer path length for normal instances and shorter path length for anomalies. This happens because, as shown before, the anomalies are more susceptible to isolation.

For this reason the anomaly score s of an instance x is defined as:

$$s(x, n) = 2^{-\frac{E(p(x))}{c(n)}}, \tag{2.1}$$

where $E(p(x))$ is the average $p(x)$ of the iTrees of the Forest and $c(n)$ is a normalization factor. The value of $c(n)$ estimate the average path length for external node, its value is taken from the Binary Search Tree unsuccessful search, since it represent the same operation, and is computed as follows:

$$c(n) = 2H(n-1) - \frac{2(n-1)}{n}, \tag{2.2}$$

where $E(p(x))$ is the average $p(x)$ of the iTrees of the Forest and $c(n)$ is a normalization factor.

The value of $c(n)$ estimate the height of a tree of n elements and we use it as normalization factor for $E(p(x))$. Its value is taken from the Binary Search Tree unsuccessful search, since it represent the same operation, and is computed as follows:

- when $E(p(x)) \to 0$, $s \to 1$;
- when $E(p(x)) \to n-1$, $s \to 0$;
- when $E(p(x)) \to c(n)$, $s \to 0.5$.

and it reflects what we said before about the relation between $E(p(x))$ and the predicted label:

(a) shorter path length, s close to 1, the test instance is an anomaly;

(b) longer path length, s close to 0, is normal;

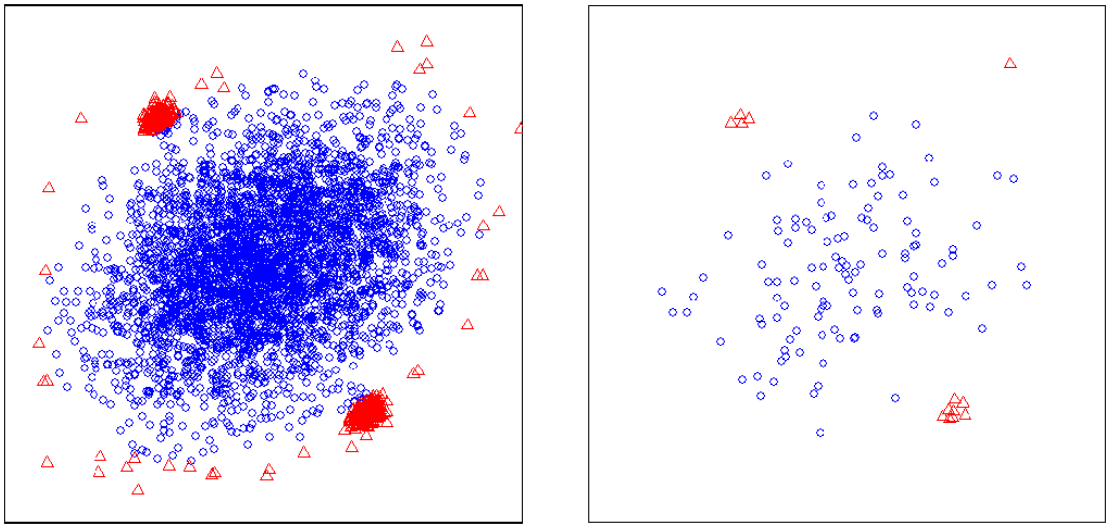
(c) if $s \approx 0.5$, uncertain decision.

2.3. Sub-Sampling

Since iForest is based on a iTree ensemble, every iTree doesn't need to have excellent performance. The forest can be seen as a collection of different *experts* in which each one capture different kind of anomalies. Then, all the trees together create a complex model able to capture a huge amount of different anomalies.

Isolation methods work better when the sampling size is kept small, unlike existing method where large sampling size is more desirable, in fact sub-sampling is used to satisfy this condition and reach excellent results.
In iForest, sub-samples are produced by randomly selecting without replacement instances from the original dataset, and each sub-sample is used to train an iTree.

Another import aspect is that sub-sampling alleviates the effects of two common problems in anomaly detection: swamping and masking. *Swamping* happens when normal instances are labeled as anomalies, while *masking* is the opposite, anomalous instance not recognized as anomalous and labeled as normal.
Masking is a common problem when anomalies form a cluster, because is harder to isolate them and they tend to be recognized as normal. Swamping occurs when normal instances are far from normal clusters and can be mispredicted, since they are in a lower density area. In Figure 2.2[1] a clear example of how sub-sample works and why alleviates the masking and the swamping problems.



(a) Original sample (4096 instances) (b) Sub-sample (128 instances)

Figure 2.2: Data coming from Mulcross dataset. Original data (a) and a sub-sample (b). Blue circles denote normal instances and red triangles denote anomalies.

2.4. Implementation

2.4.1. Training Phase

Isolation Forest is composed by the usual two steps, we now describe the first one: *training*.

As introduced before, iForest consists in a collection of iTrees built using a sub-sample of the training set and each iTree use a different random sub-sample (Algorithm 1).

Algorithm 1 : *iForest(X,t,ψ)*

Input: X - input data, t - number of trees, ψ - sub-sampling size
Output: a set of t iTrees

1: **Initialize** *Forest*
2: set height limit $l = ceiling(log_2\psi)$
3: **for** $i = 1$ to t **do**
4: $X' \leftarrow sample(X, \psi)$
5: $Forest \leftarrow Forest \cup iTree(X', 0, l)$
6: **end for**
7: **return** *Forest*

Each iTree is constructed by recursively partitioning the given sub-sample until the instances are isolated or a specific tree height l is reached (usually is based on the sub-sampling size ψ and set to $l = ceiling(log_2\psi)$).

The reason why make sense to stop the iTree growth is that the algorithm focuses on data points that can be isolated in a few splits, hence an anomalous data point can be detected using only the first levels of the iTree.

In the training phase there are only two input parameters to select: **sub-sampling size** ψ and the **number of trees** t.
The *sub-sampling size* ψ establish how many data instance are included in each sub-sample of the original dataset: in [1] is suggest to set $\psi = 256$. They note, empirically, that this is the best value, and if the value is increased the memory and time requirement increases but the performance doesn't get any improvement.
The *number of trees* t controls the ensemble size. The suggested value is $t = 100$, in fact the path length converges well before $t = 100$. This is empirically demonstrated, again, in [1].
At the end of the training process the returned model is the ensemble of iTrees ready to be used for the evaluation stage.

The *complexity* of iForest training is $\mathcal{O}(t\psi \log \psi)$.
In algorithm 1 and 2 we can see more details about the training stage.

Algorithm 2 : *iTree(X,e,l)*

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

1: **if** $e \geq l$ or $|X| \leq 1$ **then**
2: **return** $exNode\{Size \leftarrow |X|\}$
3: **else**
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: $X_l \leftarrow filter(X, q < p)$
8: $X_r \leftarrow filter(X, q \geq p)$
9: **return** $inNode\{Left \leftarrow iTree(X_l, e+1, l),$
$Right \leftarrow iTree(X_r, e+1, l),$
$SplitAtt \leftarrow q,$
$SplitValue \leftarrow p\}$
10: **end if**

2.4.2. Evaluating Phase

The second step is the *evaluation*, to predict if an input instance is an anomaly or not. In order to do that we have to compute the anomaly score as defined in (2.1).
In the anomaly score computation the most important variable is $E(p(x))$ - the average

path length - defined as follows:

$$E(p(x)) = \frac{1}{t}\sum_{i=1}^{t} p_i(x) \tag{2.3}$$

where $p_i(x)$ is the path length of the instance x in the *i-th* iTree.
The *path length* $p(x)$ is defined as the number of edges traversed by the instance x from the root node to a terminating/external node. When x reach the external node, this node can be external for two reasons: a) no further split available or b) the tree height limit is reached. If the reason is b) we want take into account that the path length should be longer.
To do that we add to the path length an adjustment $c(Size)$, which estimates the truncated path length by computing the average length of a tree with $Size$ elements, which represent the truncated subtree.
More details of the *PathLength* computation in Algorithm 3.

Algorithm 3 : *PathLength(x,T,e)*

Input: x - data instance, T - an iTree, e - current path length (initialized to zero at the first call)
Output: path length of x

```
1: if T is an external node then
2:     return e + c(T.size)          // c(·) normalization factor defined in (2.2)
3: end if
4: a ← T.splitAtt
5: if x_a < T.splitValue then
6:     return PathLength(x, T.left, e + 1)
7: else
8:     return PathLength(x, T.right, e + 1)
9: end if
```

Finally, once the $p(x)$ is obtained from all the iTrees, we compute the anomaly score $s(x,\psi)$ as defined in (2.1).

The *complexity* of the evaluation process for a test dataset of n instances is $\mathcal{O}(nt\log\psi)$.

2.5. Empirical Evaluation

In this section we report the empirical evaluation explained in the original paper[1], in which they compare the iForest performance with other anomaly detection common methods, such as ORCA, LOF and Random Forest (RF).

The comparison is based on *ROC AUC* and *processing time* for the above mentioned methods on twelve datasets.

2.5.1. ROC AUC

If processing time is a self-explain metric for evaluation, ROC AUC (*Receiver Operating Characteristic - Area Under the Curve*) needs a more detailed explanation.

First of all let's introduce the concept of *confusion matrix*: a table that shows the performance of a classification algorithm. An example is shown in Table 2.1, where each row represents the actual number of instances in a class, while each column represents the number of instances predicted in a class.

In case of anomaly detection we categorize data into two classes: *anomaly* and *normal*, even if we are facing with a multi-class anomaly detection problem, because our algorithm must be evaluated for anomaly detection performance and not for multi-class classification performance.
In order to better understand the next concepts in Table 2.1 the two classes of the confusion matrix are represented as *Positive* and *Negative*. Positive corresponds to the Anomaly class, while Negative to the Normal one.

		Predicted Class		
		Positive (PP)	Negative (PN)	Total
Actual Class	Positive (P)	TP	FN	$TP + FN$
	Negative (N)	FP	TN	$FP + TN$
	Total	$TP + FP$	$FN + TN$	N

Table 2.1: Confusion Matrix.

Given a total population of N test samples, P of them are positive and N negative ($N = P + N$).
The algorithm we want to evaluate has predicted PP (Predicted Positive) of them as positive and PN (Predicted Negative) as negative.
The confusion matrix is used to evaluate its performance because we can see how many of them have been labeled both correctly and incorrectly: TP - True Positive - represent

the instance belonging to the positive class and predicted as positive, same thing for TN - True Negative - which represent the number of negative instances correctly labeled.
The labeling errors are represented by FN and FP: False Negative, positive instances labeled as negative, and viceversa for False Positive. A FN represent a miss on detection of an anomaly, while FP represent a false alarm. Depending on the domain we are working on, an error can have different costs. For example in the field of health care a miss (FN) has higher cost than a false alarm (FP).

Once the confusion matrix is well defined there are a lot of metrics that can be derived from it, let's see some of them:

- **Precision**: the fraction of Predicted Positive correctly labeled

$$Precision = \frac{TP}{TP + FP}$$

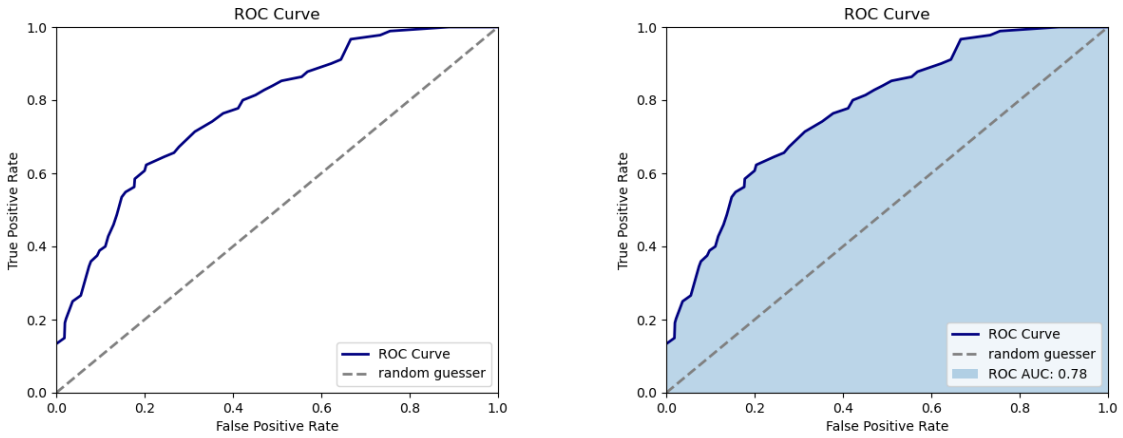
- **Recall, or True Positive Rate (TPR)**: the fraction of Positive correctly retrieved, also defined as the *probability of detection*

$$Recall = \frac{TP}{TP + FN}$$

- **Fall-out, or False Positive Rate (FPR)**: the fraction of Negative instances labeled as Positive over the total number of Negative, also defined as the *probability of false alarm*

$$Fall - out = \frac{FP}{FP + TN}$$

Before the definition of ROC AUC, we have to show what the *ROC Curve* is: a graph that shows the performance of a binary classifier as the discrimination threshold varies.
The ROC Curve plots the TPR against the FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives.
Figure 2.3(a) shows a typical ROC Curve.
The idea is to have a curve as close as possible to the top-left corner, which means a TPR close to 1 even for low value of FPR. The dashed gray line in the middle (Figure 2.3(a)) represent the random guesser performance, in fact the minimum goal for a classifier is to have performance better than the random guesser, which means a curve that is above the dashed line and closer than that to the top-left corner.



(a) ROC Curve

(b) ROC AUC

Figure 2.3: The two graphs represent the same ROC Curve. (a) shows only the curve, (b) highlights the Area Under the ROC Curve

We can now define the *ROC AUC*, which is the Area Under the ROC Curve, represented in Figure 2.3(b) as the light blue area under the blue line of the ROC Curve. Of course, the goal is to have the ROC AUC as close as possible to 1, which means a curve close to the top-left corner.

As it is correctly reported in [12], AUC is desirable for the following two reasons:

- is **scale-invariant**, it measures how well predictions are ranked, rather than their absolute values;
- is **classification-threshold-invariant**, it measures the quality of the model's predictions irrespective of what classification threshold is chosen.

This metric is very useful to compare two ROC Curves that looks different and for which is difficult to find the best one only looking at the curves.

We have defined the ROC AUC and, as said before, this is the metric on which the iForest performances are based.

2.5.2. Datasets

The datasets used for testing are twelve: eleven of them are natural datasets, plus a synthetic one. They are selected because are used in the literature to evaluate anomaly detectors in similar settings and their anomalies belongs to known anomaly classes.
They include: *Http* and *Smtp* (two big datasets used for network intrusion detection),

Dataset	n	d	anomaly class
Http (KDDCUP99)	567497	3	attack(0.4%)
ForestCover	286048	10	attack(0.9%) vs class 2
Mulcross	262144	4	2 clusters(10%)
Smtp (KDDCUP99)	95156	3	attack(0.03%)
Shuttle	49097	9	classes 2,3,5,6,7 (7%)
Mammography	11183	6	class 1 (2%)
Annthyroid	6832	6	classes 1, 2(7%)
Satellite	6435	36	3 smallest classes (32%)
Pima	768	8	pos(35%)
Breastw	683	9	malignant(35%)
Arrhythmia	452	274	classes 3,4,5,7 8,9,14,15 (15%)
Ionosphere	351	32	bad(36%)

Table 2.2: Datasets properties. n: number of instances, d: number of dimensions and in brackets the contamination[1].

Annthyroid, *Arrhythmia*, *Breastw* (Wisconsin Breast Cancer), *ForestCover*, *Ionosphere*, *Pima*, *Satellite*, *Shuttle*, *Mammography* and *Mulcross* (the synthetic one).
The synthetic dataset, *Mulcross*, is generated by setting the *contamination ratio=10%* (number of anomalies over the total number of points), *distance factor=2* (distance between the center of normal cluster and anomaly clusters), and *number of anomaly clusters=2*. The sub-sampling example seen before, in Figure 2.2, uses *Mulcross* data.

Figure 2.2 shows, using a table, the main characteristics of the datasets and information on anomaly classes.

2.5.3. Results

In Figure 2.4 are reported all the results: ROC AUC and time processing both for iForest and for ORCA, LOF and RF.
We observe that iForest outperforms all the three methods, both in terms of ROC AUC and time processing.

	AUC				Time (seconds)					
	iForest	ORCA	LOF	RF	iForest			ORCA	LOF	RF
					Train	Eval.	Total			
Http (KDDCUP99)	**1.00**	0.36	NA	NA	0.25	15.33	**15.58**	9487.47	NA	NA
ForestCover	**0.88**	0.83	NA	NA	0.76	15.57	**16.33**	6995.17	NA	NA
Mulcross	**0.97**	0.33	NA	NA	0.26	12.26	**12.52**	2512.20	NA	NA
Smtp (KDDCUP99)	**0.88**	0.80	NA	NA	0.14	2.58	**2.72**	267.45	NA	NA
Shuttle	**1.00**	0.60	0.55	NA	0.30	2.83	**3.13**	156.66	7489.74	NA
Mammography	**0.86**	0.77	0.67	NA	0.16	0.50	**0.66**	4.49	14647.00	NA
Annthyroid	**0.82**	0.68	0.72	NA	0.15	0.36	**0.51**	2.32	72.02	NA
Satellite	**0.71**	0.65	0.52	NA	0.46	1.17	**1.63**	8.51	217.39	NA
Pima	0.67	**0.71**	0.49	0.65	0.17	0.11	0.28	**0.06**	1.14	4.98
Breastw	**0.99**	0.98	0.37	0.97	0.17	0.11	0.28	**0.04**	1.77	3.10
Arrhythmia	**0.80**	0.78	0.73	0.60	2.12	0.86	2.98	**0.49**	6.35	2.32
Ionosphere	0.85	**0.92**	0.89	0.85	0.33	0.15	0.48	**0.04**	0.64	0.83

Figure 2.4: Performance comparison in terms of ROC AUC and processing time. In bold the best performance for each dataset.

More precisely, the comparison with ORCA shows the differences, in terms of execution time, between a distance-based method (ORCA), which requires pairwise distance computation, and a model-based method (iForest): for smaller dataset (<1000 instances) ORCA is faster, but for larger dataset there is a massive difference in favour of iForest.
The comparison with ORCA also shows that iForest works very well in large dataset in terms of ROC AUC, achieving excellent perfomances.
iForest compares favourable to LOF, too, in seven out of eight data sets, and is better than RF in all the four data sets tested, both in terms of ROC AUC and processing time.

3 | Proposed Solution

3.1. The Problem Formulation

Let's look at the problem of AD in a more formal way[4]. The anomaly detection problem consists in monitoring a set of n data:

$$\mathcal{X} = \{x_i, \ldots, x_n \quad | \quad x_i \in \mathbb{R}^d\},$$

where each element x_i is realization of a random variable having pdf ϕ_0, with the aim of detect outliers, i.e. points that do not conform with ϕ_0.

$$x_i \sim \begin{cases} \phi_0 & \textit{normal data} \\ \phi_1 & \textit{anomalies} \end{cases}$$

where $\phi_0 \neq \phi_1$ and ϕ_0 and ϕ_1 are unknown. Since X has no label we are in an unsupervised setting, and X contains both normal instances and anomalies, furthermore there is no clear distinction between training and test data. Each dataset contains both normal and anomalous samples. The only mild assumption we can make is that normal data far outnumber anomalies.

3.2. Embedding Definitions

We introduce a new embedding that gives to input data a new representation, but first of all introduce some definitions:

- **depths vector** y: the input instance x is processed by the iTrees and the output will be a t-dimensional vector $y \in \mathbb{R}^t$, where *t: number of trees of the forest.* Intuitively, we might think at the elements of y (the resulting depths of iTrees) as integers, but we have to consider the correction factor $c(\cdot)$, which is a real number, hence the *depths vector* is composed by real numbers, $y \in \mathbb{R}^t$;

- **histogram** h: given the *depths vector* y, we compute the histogram and we obtain $h \in \mathbb{Q}^n$, a n-dimensional vector of rational element, more precisely each element lies in the interval $[0, 1]$ and it's a rational number. n is defined as n=*ceiling(max(y))*, thus the first integer bigger than the max of y.
  Each element of the histogram vector h represents the fraction of the t elements that are into a specific bin. This because, each element of h is computed as the number of elements of y that are in a specific interval, then h is normalized in order to have *norm-1* equal to 1.
  Follows the mathematical formulation:

$$h = \textit{histogram(y)}, \qquad \begin{aligned} &\|h\|_1 = 1 \\ &h_i = \frac{1}{t} \begin{cases} \#([i, i+1)) & \text{if i} = 1, \ldots, \text{n-1} \\ \#([i, i+1]) & \text{if i} = \text{n} \end{cases} \end{aligned} \tag{3.1}$$

  where $\frac{1}{t}$ yield $\|h\|_1 = 1$ and $\#(\cdot)$ operator counts the number of elements that are into the given bin. All the bins are half-open, except for the last one which is closed.

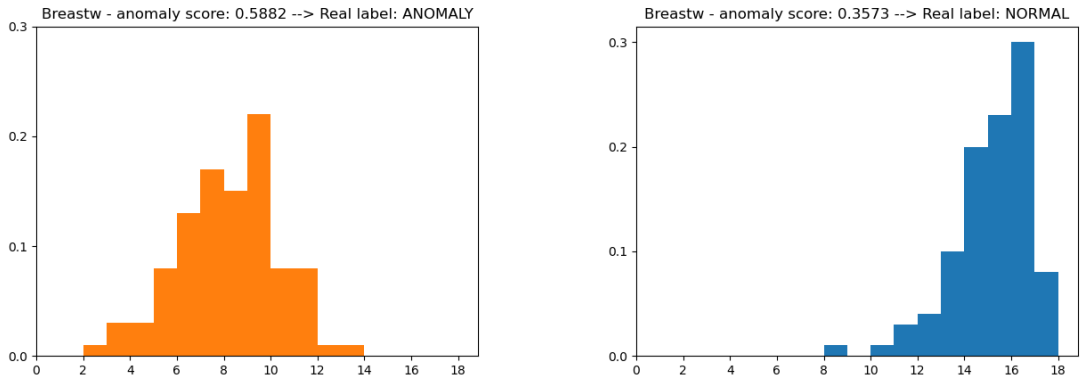
3.3. Embedding

Now we have all the definitions we need to describe the embedding. First of all, let's summarize, using the following formulation, the steps we have to do to obtain the histogram h starting from an input instance x:

$$x \in \mathbb{R}^d \xrightarrow{iForest} y \in \mathbb{R}^t \xrightarrow{histogram} h \in \mathbb{Q}^n$$

Our *embedding* is a new dimensional space in $\mathbb{Q}^n$, where input instances x, transformed in the corresponding histogram h, lie in a n-dimensional *simplex* (a triangle in n dimensions), because the constraint $\|h\|_1 = 1$ positions all of them in an hyperplane, while $h_i \in [0, 1]$, for $i, \ldots, n$, constraints them in a simplex on that hyperplane.

Using the embedding, and so using this new representation of input data, we expect different histograms for normal and anomalous instances.
More precisely, a correct labeled anomalous instance have high frequencies for bins representing low depths, while a correct labeled normal instance have high frequencies for bins representing high depths. Figure 3.1 shows an example of both normal and anomaly histogram.



(a) Anomaly instance

(b) Normal instance

Figure 3.1: Data coming from Breastw dataset. (a) Anomaly shows that most iTrees return a depth between 5 and 12. (b) Normal shows highest frequencies between 13 and 18.

3.4. Remove the Correction Factor

Describing the evaluation phase of iForest in Section 2.4.2 comes out that, when an input instance x reaches a terminal node, to $p(x)$ is added the correction factor $c(Size)$, which estimates the height of the truncated subtree. We want to remove it, the reasons are described in the following section.

3.4.1. Why remove the correction factor

The correction factor leads to two problems in the embedding:

(i) *increases the embedding dimension* (increases the n of $\mathbb{Q}^n$), and

(ii) *the depth become a real number*, instead of an integer.

We want to start the improvements by redefining the anomaly score (2.1) in the embedding, to do that is necessary to have $y \in \mathbb{N}^t$, so from this point on, the embedding is defined using depths vector $y \in \mathbb{N}^t$ computed using iForest *without correction*. Later, a formulation of $E(p(x))$ in the embedding if there is no correction factor.

3.4.2. Estimation of Correction Factor

The value of the correction factor $c(n)$ depends on the number n of elements of the subtree for which we have to estimate the height.

Now, knowing the range of values of n during the evaluation phase, we will compute exactly the maximum value of $c(n)$.

In order to discover the range of values of n, let's take a step back (to Algorithm 2 and 3): during the training phase the iTree growth can be stopped for three reasons, the one we are interested in is when the maximum iTree height l is reached. When the growth is stopped for this reason, in the terminal node the remaining number of elements (not split) is also stored.
This value, called *Size* in Algorithm 2, is the value assigned to n during the evaluation to compute the correction factor $c(n)$.

During the training, each node has a set of data X and the splitting criterion divides X into two subsets: X_l and X_r. Each resulting subset is composed by at least one element. In the root node, $|X| = \psi$ and in the worst case the two splits have cardinality $|X_l| = 1$ and $|X_r| = |X| - 1$. If this bad split happens at each level, after s splits the resulting subset will have cardinality $|X| = \psi - s$, while the other children of the splits are terminal nodes since they have only one element.

Each iTree is built using a sub-sample size of $\psi = 256$, so the maximum iTree height is $l = ceiling(log_2\psi) = 8$, where $ceiling(\cdot)$ take the first integer greater than the parameter. Thus, from the root we have $s = 8$ splits to reach the terminal node, and the $Size$ attribute of the *external Node* in worst case will be $Size = 256 - 8 = 248$.
We found that n, during evaluation stage, has values in $[1, 248]$ and so $c(n)$ has maximum value $c(n_{max}) = c(248) = 10.18$.

This is the impact of the correction factor on the output depth of an iTree, this increase of depth reflects on the embedding size, as introduced above while listing the main problems of use the correction factor in embedding. For this specific sub-sampling size $\psi = 256$, the maximum depth, in the worst case, is more than doubled, going from 8 to $8 + 10.18 = 18.18$. Same enlargement also occurs to the embedding size.

3.5. Average Path Length in Embedding

As a starting point, in order to try to improve the performance of iForest using the embedding, we redefine the average path length computation in this new space. It's necessary, for the reasons described before (in Section 3.4.1), to consider the iForest version *without correction factor*.

So the average path length $E(p(x))$ of iForest, in the embedding, is computed with the

following expression:

$$E(p(x)) = \sum_{i=1}^{n} w_i \cdot h_i, \tag{3.2}$$

where $w_i = i$, h_i is the *i-th* element of h and $h = histogram(y)$, as defined in (3.1). Since we are considering iForest *without correction*, the depths vector y is composed only by integer values ($y \in \mathbb{N}^t$), thus h_i is the fraction of elements of the depths vector y that are equal to i:

$$h_i = \frac{1}{t} \sum_{j=1}^{t} \alpha(y_j == i), \qquad \text{where} \quad \alpha(test) = \begin{cases} 1 & \text{test is True} \\ 0 & \text{test is False} \end{cases}$$

h_i is computed by looking at all the elements of y, counting how many of them are equal to i and then, dividing this value by t, we obtain the fraction.

Now we retrieve the formula of the average path length $E(p(x))$ used for iForest and defined in (2.3), and together with one defined above, in (3.2), we show the correspondence between this new formulation and the average path length:

$$\begin{aligned} \text{(no embedding)} \qquad & E(p(x)) = \frac{1}{t} \sum_{i=1}^{t} p_i = \frac{1}{t} \sum_{i=1}^{t} y_i \\ \text{(embedding)} \qquad & E(p(x)) = \sum_{i=1}^{n} w_i \cdot h_i = \\ & = \sum_{i=1}^{n} w_i \cdot \frac{1}{t} \sum_{j=1}^{t} \alpha(y_j == i) \\ & = \frac{1}{t} \sum_{i=1}^{n} w_i \cdot \sum_{j=1}^{t} \alpha(y_j == i) \end{aligned}$$

In both the two formulations appear the normalization factor $\frac{1}{t}$, now we show the equivalence of the two formulations by removing this common factor $\frac{1}{t}$:

$$\sum_{i=1}^{t} y_i = \sum_{i=1}^{n} w_i \cdot \sum_{j=1}^{t} \alpha(y_j == i)$$

because both of them compute the sum of all the elements of the *depths vector y*.

The *no embedding* formula simply computes the sum of the elements from y, while the *embedding* formula computes the sum by exploiting the histogram formulation: for every

possible element of the *depths vector* y (an integer number in range $[1, n]$), it count how many times it is present, and multiply this count by its value.

Example. number 2 (w_i) appear 3 ($\sum_{j=1}^{t} \alpha(y_j == i)$) times in y,
applying the formula $w_i \cdot \sum_{j=1}^{t} \alpha(y_j == i) = 2 \cdot 3 = 6$
we obtain the partial sum of the elements equal to 2.

Repeat the partial sum computation for all the possible
value of w_i: $w_i = 1, \ldots, n$
and we obtain all the partial sums.

The external summation put together all the partial sums
and compute the final sum which is equivalent the one computed
by the *no embedding* version.

Thus, showing the correspondence of the two formulations, now we know how to represent in the embedding space the average path length.

3.6. Linear Discriminant Analysis

Looking at (3.2), we note that the $E(p(x))$ in the embedding is simply a linear combination of the embedding feature $h_1, \ldots, h_n$ with predefined weights. The intuition is about finding a new weight vector $\hat{w}$, which is dataset-specific and can increase the AD performance.

For this purpose we use Linear Discriminant Analysis (LDA) to find the new weights. LDA is a supervised method used in statistics and pattern recognition to find a linear combination of features that separates two or more classes. Hence we use LDA to find a linear combination of $h_1, \ldots, h_n$ which better separates the two classes.
The new formulation is:

$$L(y) = \sum_{i=1}^{n} \hat{w}_i \cdot h_i, \tag{3.3}$$

$\hat{w}_1, \ldots, \hat{w}_n$ weights computed using LDA.
Using LDA we are using a supervised technique, so we are computing something like an upper bound of the performance of iForest, since the linear combination is computed using optimal weights $\hat{w}$.

3.7. One-Class SVM

Another idea comes from the fact that the embedding is a totally new representation of the data. We apply in this new framework a well known AD technique: One-Class Support Vector Machine (OC SVM). It is able to identify the best boundary, in feature space, which separates normal instances from anomalies. We want to know if in this new data representation makes easier perform anomaly detection.

3.8. Visual Representation of the Embedding

We want to show graphically how the input data are represented in the embedding space and what the average path length computation, used to compute the anomaly score in iForest, represent in the embedding.

As defined in Section 3.2, the embedding representation of input data is located in a $\mathbb{Q}^n$ space. Of course the visualization can be done only if the space is at most 3D, so if we want a graphical representation of the embedding, n must be at most $n = 3$.

For the following representation we consider iForest *without correction*, since we want to represent the average path length in the embedding, and, as explained in the previous sections, to do that is necessary to use this iForest version.

3.8.1. 2D representation

Let's start our visualizations by taking $n = 2$. The steps to obtain the histogram h from input data are the following:

$$x \in \mathbb{R}^d \xrightarrow{iForest} y \in \mathbb{N}^t \xrightarrow{histogram} h \in \mathbb{Q}^2$$

We define:

- the **histogram h**, which is the embedding version of the input data x, is represented as:
$$h = [h_1 h_2]$$
- all the possible combinations of h_1 and h_2 obtained from input data, as a **segment**:
$$h_1 + h_2 = 1, \qquad 0 \leq h_i \leq 1, \qquad i = 1, 2$$

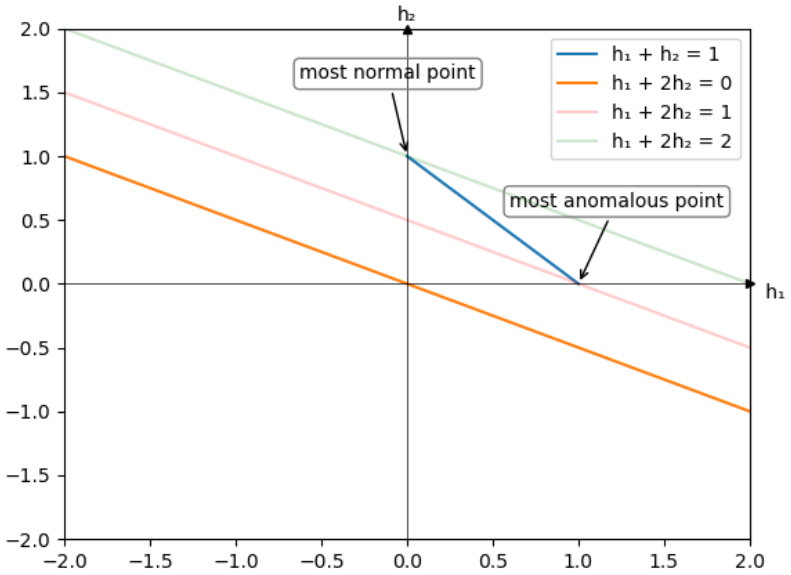


Figure 3.2: 2D embedding visualization.

because in Formula (3.1) we define $\|h\|_1 = 1$, thus $\sum_{i=1}^{n} h_i = 1$;

- the zero average path length $\mathbf{E}(\mathbf{p}(\mathbf{x})) = \mathbf{0}$ in the embedding, as a **line**:

$$1 \cdot h_1 + 2 \cdot h_2 = 0,$$

which represents the *most anomalous instances*, since *E(p(x))=0* $\Rightarrow$ *s(x)=1*, and the anomaly score *s(x)=1* is the highest possible value for anomaly score. To define the average path length in the embedding we use the Formula (3.2).

These definitions are represented in Figure 3.2. As described also in the legend of the figure, the *blue segment* represents all the possible combinations of h_1 and h_2 obtained from the depths vector y, and the *orange line* represents all the possible combinations of h_1 and h_2 such that $E(p(x)) = 0$.

If we consider the $h_1 + 2 \cdot h_2 = 0$ (orange) line and the $h_1 + h_2 = 1$ (with $0 \leq h_i \leq 1$, $i = 1, 2$) (blue) segment, there is no intersection. In fact is impossible to obtain an average path length $E(p(x)) = 0$, since each depth $p_i(x) \geq 1$.
But, since that line represents the *most anomalous instances*, even if that combinations cannot be obtained, the closer an instance is to that line, higher is the anomaly score associated to that instance.
In fact the point indicated with *most anomalous point* is the closer (on the blue segment) to that line, while the furthest is labeled as the *most normal* one.

The other two lines: $E(p(x)) = 1$ and $E(p(x)) = 2$, represent all the possible combinations of h_1 and h_2 to obtain, respectively, $E(p(x)) = 1$ and $E(p(x)) = 2$.
In this case each one of the two lines has an intersection with the (blue) segment. It means that is possible to have an input instance from which results $E(p(x)) = 1$ or $E(p(x)) = 2$, and respectively they are $[1, 0]$ and $[0, 1]$.
All the other values in the blue segment have a different $E(p(x))$, these two represent the extreme ones.

The most important thing is to note that $h_1 + 2 \cdot h_2 = i$ are the lines the represent the average path length as i varies ($E(p(x)) = i$).

3.8.2. 3D representation

We do the same thing in 3D. The steps are always the same, the only difference is the dimensions of the embedding space:

$$x \in \mathbb{R}^d \xrightarrow{iForest} y \in \mathbb{N}^t \xrightarrow{histogram} h \in \mathbb{Q}^3$$

The components are the same as before, but there is one more dimension to consider:

- the **histogram h** is represented as:

$$h = [h_1 h_2 h_3]$$

- all the possible combinations of h_1, h_2 and h_3 obtained from input data, are no more represented by a segment, but by a section of a plane, more precisely the section is a **triangle** (Figure 3.3). And the equation is:

$$h_1 + h_2 + h_3 = 1, \qquad 0 \leq h_i \leq 1, \qquad i = 1, 2, 3$$

  again because in Formula (3.1) we define $\|h\|_1 = 1$, thus $\sum_{i=1}^{n} h_i = 1$;

- the zero average path length $\mathbf{E(p(x)) = 0}$ in the embedding, is a **plane**:

$$1 \cdot h_1 + 2 \cdot h_2 + 3 \cdot h_3 = 0,$$

  and represents, again, the *most anomalous instances.*

Figure 3.3 and Figure 3.4 shows the same 3D representation from two differents perspec-

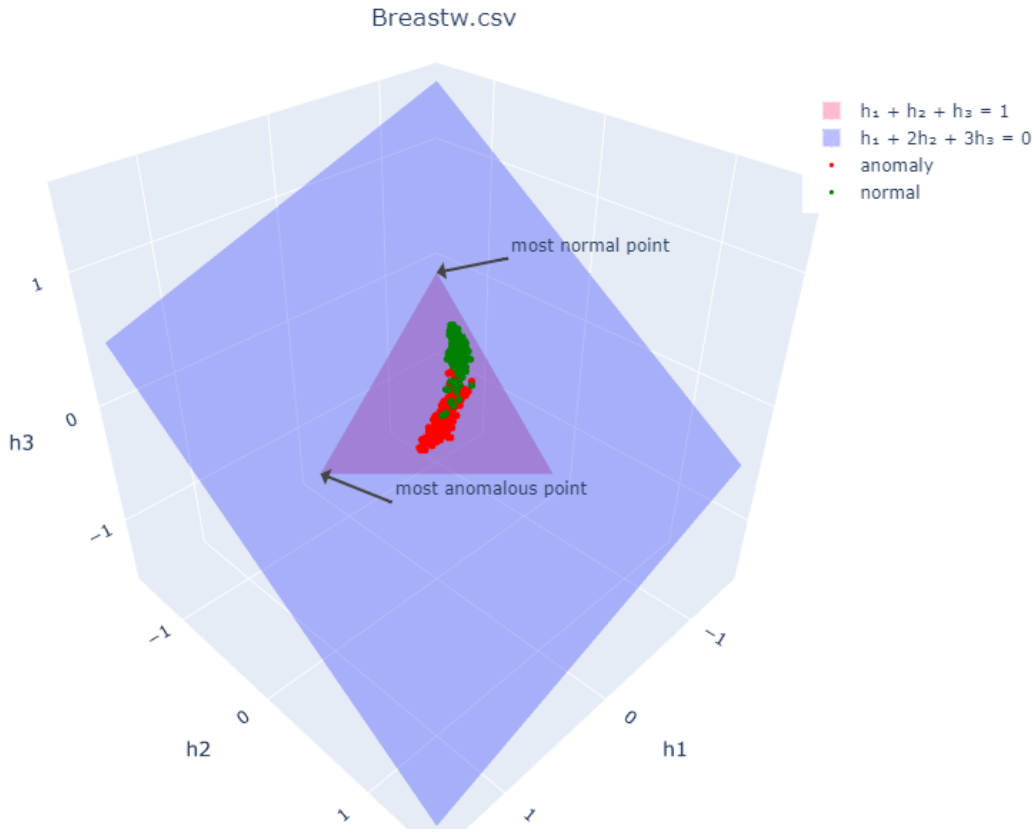


Figure 3.3: 3D embedding representation.

tives.

The *triangle* in the section of plane $h_1+h_2+h_3 = 1$, for which $0 \leq h_i \leq 1$, is the equivalent of the 2D blue segment, i.e. all the possible combinations of h_1, h_2 and h_3 obtained from the depths vector y.
In this 3D representation also input data are drawn (red for anomalies and green for normals).

The *purple plane* represents the space of the points for which $h_1 + 2 \cdot h_2 + 3 \cdot h_3 = 0$, the equivalent of the 2D orange line. It represents the plane of the *most anomalous instances*. Again, the $h_1 + 2 \cdot h_2 + 3 \cdot h_3 = 0$ cannot contains input data since is impossible to obtain an average path length $E(p(x)) = 0$, because each depth $p_i(x) \geq 1$.
In fact, looking at Figure 3.4, we note that there is no intersection between the plane and the triangle. Again, the *most anomalous point* is the closer one to the plane $h_1 + 2 \cdot h_2 + 3 \cdot h_3 = 0$, while the furthest one is the *most normal*.

The three vertices of the triangle corresponds to the three points $[1, 0, 0]$, $[0, 1, 0]$ and $[0, 0, 1]$, with the first one and the third one that are, respectively, the most normal one and the most anomalous one, while the middle one does not have particular characteristics.

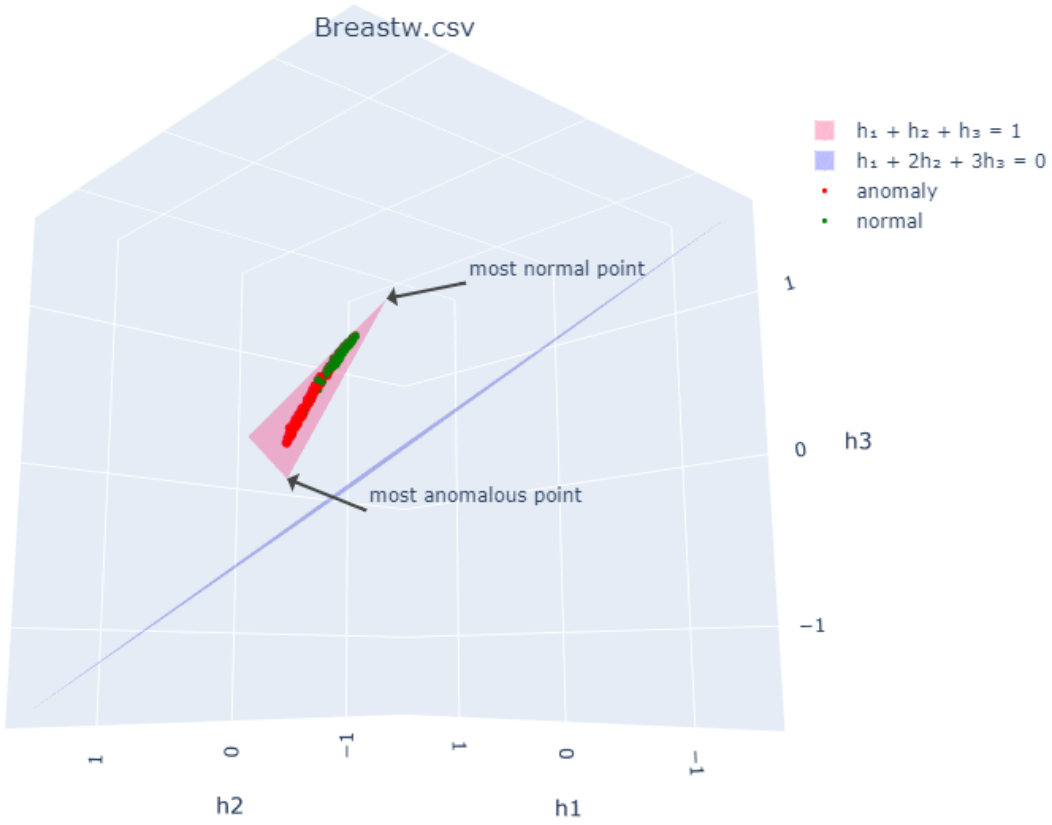


Figure 3.4: 3D representation, from another perspective.

As before, we have to highlight that the plane $h_1 + 2 \cdot h_2 + 3 \cdot h_3 = 0$ and its parallel ones, $h_1 + 2 \cdot h_2 + 3 \cdot h_3 = i$, are the planes that represent the average path length as i varies $(E(p(x)) = i)$.

3.8.3. nD representation

In n-dimensions of course a visualization is not possible but we can provide a formal definition. The steps are always the same, but the embedding space is in n-dimensional:

$$x \in \mathbb{R}^d \xrightarrow{iForest} y \in \mathbb{N}^t \xrightarrow{histogram} h \in \mathbb{Q}^n$$

The components are:

- the **histogram h** is represented as:

$$h = [h_1 \dots h_n]$$

- all the possible combinations of $h_1, \ldots, h_n$ form a **simplex**, which is a n-dimensional triangle, its equation is:

$$h_1 + \cdots + h_n = 1, \qquad 0 \leq h_i \leq 1, \qquad i = 1, \ldots, n$$

because in Formula (3.1) we define $\|h\|_1 = 1$, thus $\sum_{i=1}^{n} h_i = 1$;

- the zero average path length $\mathbf{E}(\mathbf{p}(\mathbf{x})) = \mathbf{0}$ in the embedding, is an **hyperplane**:

$$\sum_{i=1}^{n} i \cdot h_i = 1,$$

and represents, again, the *most anomalous instances*.

4 | Experiments

4.1. Datasets

The datasets used for testing our experiments are the same used to test iForest in the original paper[1] and described in Section 2.5.2.
The reason of this choice is that the benchmark for our performance is iForest, so we want to evaluate our iForest modification proposals using the same datasets on which iForest was originally tested.
In Figure 4.1 a short recap of the main characteristics of the datasets.

Dataset	n	d	anomaly class
Http (KDDCUP99)	567497	3	attack(0.4%)
ForestCover	286048	10	attack(0.9%) vs class 2
Mulcross	262144	4	2 clusters(10%)
Smtp (KDDCUP99)	95156	3	attack(0.03%)
Shuttle	49097	9	classes 2,3,5,6,7 (7%)
Mammography	11183	6	class 1 (2%)
Annthyroid	6832	6	classes 1, 2(7%)
Satellite	6435	36	3 smallest classes (32%)
Pima	768	8	pos(35%)
Breastw	683	9	malignant(35%)
Arrhythmia	452	274	classes 3,4,5,7 8,9,14,15 (15%)
Ionosphere	351	32	bad(36%)

Table 4.1: Datasets properties. n: number of instances, d: number of dimensions and in brackets the contamination[1].

4.2. Evaluation Metric

The metric for the evaluation of our experiments is ROC AUC, defined in Section 2.5.1. What said before about the datasets choice applies also here: iForest is evaluated using ROC AUC metric, so we have to evaluate our experiments with ROC AUC, too.

There is also one important thing to say: since the ROC AUC is *scale-invariant* and *classification-threshold-invariant* (Section 2.5.1 for more details), for the next experiments is not given a specific rule to compute an anomaly score with the characteristic described in Section 2.2 (values in $(0, 1]$ and a classification-threshold), but the only thing we take into account is to find scores for which is possible to compute the ROC AUC, and so the evaluation is based on their ranking.

4.3. Remove the Correction Factor

Goal. Check whether iForest *with correction factor* and iForest *without correction factor* have the same performance.

Now we want to check if there are significant changes in terms of performance if we remove the correction factor contribution during evaluation.

We execute the iForest both with and without correction factor. Then, a *paired-T-test* is performed in order to check if the performances of the two variants of the algorithm are identical or not and if there is a statistical relevance on this equality.

The *paired-T-test* is a statistical technique used to compare the mean difference between two sets of observations. Is used to check if the mean difference differs significantly across samples. If the test returns a large p-value, for example greater than 0.05 or 0.1 then we cannot reject the null hypothesis of identical averages. If the p-value is smaller than the threshold (0.05 or 0.1, as before) then we reject the null hypothesis of equal averages.

In Table 4.2 are reported only some results: one run per dataset, in order to have an idea of the differences of the two variants.

The T-test is performed on the ROC AUCs obtained by executing 10 times for every dataset the iForest algorithm. The algorithm is executed in the original setting (*with correction*) and then ROC AUC is computed for that configuration. To obtain the performance index also for the other variant, we use the obtained iTrees outcome and we limit them to $l = ceiling(log_2 \psi)$. We obtain, in this way, the result of the *"without correction"* version, using exactly the same forest.

Dataset	ROC AUC with correction	ROC AUC without correction
Http (KDDCUP99)	1.00	1.00
ForestCover	0.85	0.92
Mulcross	0.96	0.88
Smtp (KDDCUP99)	0.90	0.89
Shuttle	1.00	0.99
Mammography	0.86	0.85
Annthyroid	0.81	0.79
Satellite	0.71	0.70
Pima	0.68	0.65
Breastw	0.99	0.96
Arrhythmia	0.81	0.79
Ionosphere	0.84	0.86

Table 4.2: The three columns contain respectively: the datasets, the performance WITH correction and the performance WITHOUT correction.

First of all, we perform a *two-sided paired-T-test* to check if the mean difference of the two variants is equal across different runs. It shows a *p-value*=0.00313, and this means that the **null hypothesis of identical averages is rejected** and the variants of iForest have different perfomances.

Then, we have also executed *one-sided* tests in order to verify which variant has best perfomances:

- **with correction > without correction**: this test returns a positive *t-statistic* and a *p-value*=0.998435, which needs to be halved because we are performing a one-sided test, thus *p-value*=0.499217, but it clearly shows that the null hypothesis cannot be rejected;

- **with correction < without correction**: this test returns, again, a positive *t-statistic*, but a *p-value*=0.001565. When it is halved we have *p-value*=0.000783 and it clearly shows that the null hypothesis is rejected;

We can state that the traditional iForest, *with correction*, performs better than the one *without correction*. We have to take this into account, but for the reasons listed before we have to populate the embedding with iForest outputs generated using the *without correction* version.

4.4. Embedding + LDA

Goal. Use Linear Discriminant Analysis (LDA) to find the best linear combination of embedding features that separates the two classes.

Let's have a brief recap of the steps:

$$x \in \mathbb{R}^d \xrightarrow{iForest} y \in \mathbb{N}^t \xrightarrow{histogram} h \in \mathbb{Q}^n \xrightarrow{LDA} \hat{w} \in \mathbb{R}^n$$

The new formulation to compute the anomaly score using LDA is defined in (3.3).
Once the optimal vector weights $\hat{w}$ is computed, the training phase of LDA is finished.
We proceed with the evaluation by computing the anomaly scores and compute the ROC AUC.

4.4.1. Python implementation and Results

These experiments are executed by using the LDA python implementation available in the scikit-learn package `sklearn.discriminant_analysis.LinearDiscriminantAnalysis`[13].

Parameter Tuning. We identified *Least Squares* as the best *solver* among the available ones. All the other parameters are the same as the default settings.

The following results are made by executing 10 runs of iForest on each dataset, and after each run, are computed:

(i) the anomaly scores using average path length (standard iForest, no embedding);

(ii) the anomaly scores computed as linear combination using LDA after the embedding transformation of input data. LDA performances are measured by doing 5-fold cross validation[14].

Now let's have a look at the Table 4.3 to analyse at the obtained results.

Dataset	iForest	iForest embedding + LDA
Http (KDDCUP99)	**1.000**	0.999
ForestCover	0.938	**0.969**
Mulcross	0.899	**0.957**
Smtp (KDDCUP99)	0.850	**0.853**
Shuttle	0.995	**0.997**
Mammography	0.764	**0.823**
Annthyroid	0.816	**0.818**
Satellite	0.707	**0.726**
Pima	0.631	**0.638**
Breastw	0.957	**0.972**
Arrhythmia	**0.781**	0.776
Ionosphere	**0.863**	0.856

Table 4.3: In bold the higher ROC AUC. The embedding + LDA variant shows better results in 9 out of 12 datasets.

Among the 12 tested datasets, in 9 of them the version Embedding+LDA has performance better than the standard iForest. The other good thing is that in the three datasets in which standard iForest performs better, the difference is very small, while the proposed version, in some datasets (i.e. Mulcross, Mammography), has better results with a significant gap.

An important fact to remark is that the optimal weights $\hat{w}$ are *dataset-specific*: we couldn't find a vector $\hat{w}$ that reaches very good performance in all the dataset. So, for each dataset is necessary a LDA training phase to find the best weights for that domain. The fact that some datasets show results that are better in traditional iForest than in the embedding+LDA, is surprising because we expect that use a supervised techniques, trained on each single dataset, returns better performances.

4.5. Embedding + One-Class SVM

Goal. Obtain improvements applying One-Class SVM in the embedding.

With this experiment we want to test the embedding space with another anomaly detection algorithm. We use OC SVM and we want to analyse the characteristics of this

results.

4.5.1. Python implementation and Results

The OC SVM python implementation used for the tests is taken from the scikit-learn package `sklearn.svm.OneClassSVM`[15].

Parameter tuning. Using the scikit-learn documentation[15] we have found the following parameter to tune: *kernel*, *gamma* and ν.

- The *kernel* is a function that allows the computation of the similarity of two instances in feature space, different kernels compute different features, and so the resulting similarity will be different.
  As introduced in Section 1.5.1 when talking about OC SVM, represent input data in feature space is a convenient approach because a linear separation in the feature space corresponds to a variety of non-linear boundaries in the original space.
  We test the *kernels* proposed by scikit-learn: *linear*, *poly*, *rbf* and *sigmoid*.
- *Gamma* is the Kernel coefficient and defines how far the influence of a single training example reaches. It is used only used only for *poly*, *rbf* and *sigmoid* kernels, it is a float number, but the two suggested values are $scale{=}1/(n\_features \cdot X.variance)$, with X input data, or $auto{=}1/n\_features$.
- ν set an upper bound to the fraction of anomalies accepted in the training set, so the fraction of training data that can lie outside the boundary of normal region.
  If we consider $\nu = 0.1$, at most 10% of the training samples are allowed to be wrongly classified or can be considered as outliers by the decision boundary.

Scikit-learn documentation[15] describe OC SVM as an *unsupervised outlier detector*, in fact even if the training data are used to define the normal region (so the first intuition suggest that is a semi-supervised technique), using the ν parameter, OC SVM works well even with some anomalies in training data, because is able to handle them.

Now we will show, first of all, the results of *embedding+OC SVM* against standard *iForest*, then we will have an interesting insight on parameter tuning and the performances obtained giving them different values.

The following results (Table 4.4) are made by executing 10 runs of iForest on each dataset, and after each run, are computed: a) the standard iForest performance and b) the embedding+OC SVM performance. In b), once from the input data is computed the embedding representation, OC SVM performances are measured using 5-fold Cross Validation.

Dataset	iForest	Embedding + OC SVM
Http (KDDCUP99)	**1.000**	0.999
ForestCover	0.943	**0.912**
Mulcross	0.896	**0.918**
Smtp (KDDCUP99)	**0.878**	0.872
Shuttle	0.995	**0.997**
Mammography	0.752	**0.754**
Annthyroid	0.823	**0.828**
Satellite	0.703	**0.711**
Pima	0.631	**0.641**
Breastw	0.957	**0.966**
Arrhythmia	0.780	**0.787**
Ionosphere	**0.868**	0.865

Table 4.4: In bold the higher ROC AUC. The embedding+OC SVM variant shows better results in 8 out of 12 datasets.

The results shows that in 8 out of 12 datasets the version Embedding+OC SVM has performance better than the standard iForest, even if these differences are small (<0.01), except for *Mulcross* for which the difference, in favour of Embedding+OC SVM, is 0.022, and *ForestCover* for which the performance of standard iForest are better for 0.031.

The interesting thing about *parameter tuning* is when, tuning the OC SVM we observe that embedding+OC SVM has much more stability with different parameter values than the OC SVM on original data.
We tune the OC SVM by changing the values of three parameters described before and we obtain this behaviour in all the datasets.

In Table 4.5 is reported a proof of that. For simplicity the example comes only from one dataset and take only a few parameters combinations (in particular only two values of nu), but the trend is repeated in all the dataset and for more parameters combinations.

The proposed ROC AUC are computed without cross validation, because cross validation

dataset	kernel	gamma	ν	Embedding+OC SMV ROC AUC	OC SVM ROC AUC
Pima	poly	auto	0.9	0.618	0.243
Pima	poly	auto	0.1	0.618	0.307
Pima	poly	scale	0.9	0.618	0.243
Pima	poly	scale	0.1	0.618	0.307
Pima	linear	auto	0.9	0.618	0.237
Pima	linear	auto	0.1	0.618	0.292
Pima	linear	scale	0.9	0.618	0.237
Pima	linear	scale	0.1	0.618	0.292
Pima	rbf	auto	0.9	0.449	0.490
Pima	rbf	auto	0.1	0.547	0.530
Pima	rbf	scale	0.9	0.453	0.636
Pima	rbf	scale	0.1	0.552	0.669
Pima	sigmoid	auto	0.9	0.618	0.5
Pima	sigmoid	auto	0.1	0.618	0.5
Pima	sigmoid	scale	0.9	0.618	0.233
Pima	sigmoid	scale	0.1	0.618	0.274

Table 4.5: Performance of *Embedding+OC SVM* vs *OC SVM* on original data during parameter tuning. ROC AUCs rounded to the third decimal place.

compute the mean of the performances generated by more than one model, and so the stability is easier to obtain (because it is a mean value). We simply split the datain train and test, train the OC SVM model with a parameter combination and then evaluate the performance. The split is done by using a seed in order to obtain every time the same data split and train the model on the same training data but with different parameters.

For the embedding+OC SVM the values change only for *rbf* as *kernel*, all the other are similar. In the table the values are rounded to the third decimal place, of course in the other decimal places there are some changes, but we are talking about differences of 10^{-4} or 10^{-5} from one parameters combination and another. While looking at the OC SVM in original data these differences are much bigger and the ROC AUCs change a lot for

different *kernels*.

Finally, we can infer that the embedding project input data in such a way that OC SVM perform in a similar way even if it is executed with different parameters, and this behaviour highlight a higher stability.

5 | Conclusions

In this thesis we faced the problem of anomaly detection by using a state-of-the-art algorithm as starting point. In particular the main goal was to change its last operation with the intention of replace it with a more complex one, able to capture new features in the data.
The main contributions are:

- the creation of a new data representation. In this new embedding (Section 3.3) we are able to represent input data in a totally different way, based on the Isolation Forest output;
- an analysis of this new embedding space and the explanation (Section 3.8) of what the average path length, the last operation of Isolation Forest, represent there;
- new algorithms (Sections 4.4 and 4.5) to perform anomaly detection that reach results better than the starting point Isolation Forest.

The new algorithms above mentioned are not really *new*, but are simply Isolation Forest modifications, the core mechanism is based on that.
The two algorithms are both tested in the same datasets on which Isolation Forest was tested when it was presented from the inventors, and in both the presented algorithms the performances are better than Isolation Forest on most datasets. But the best thing left by our work is this new embedding space where a lot of new experiments can be done and other techniques can be applied.

The future developments that can be done starting from our work are the following:

- create the embedding *with correction factor*. In Section 3.4 we described what is it and why we want to remove it, but a future development can be an in-depth analysis of how to maintain the correction factor when the output of Isolation Forest is obtained and the embedding space is populated by input transformed data. For example the computation of the maximum value of the correction factor can be

useful to determine the upper bound of the embedding dimension and work with histograms with an higher number of bins and probably with more information (even if the increasing dimensionality is a drawback);

- change the embedding space formulation. The idea of starting from the Isolation Forest output to create a new representation is maintained, but the histogram can be computed by using different bins, or an operation different from the histogram can be used, giving more importance to the output of specific iTrees, instead of using an aggregative measure as the histograms.;
- the embedding is a new data representation, and this gives the possibility of apply there all the other techniques that can be applied also in original space. So, new experiments with other anomaly detection or classification techniques can be done.

Bibliography

[1] Zhi-Hua Zhou, Fei Tony Liu, and Kai Ming Ting. *Isolation Forest.* 2008.

[2] Varun Chandola, Arindam Banerjee, and Vipin Kumar. *Anomaly detection: A survey. ACM Comput. Surv.*, 41:15:1–15:58, 2009.

[3] Gianluca Goffredi. *Multivariate Spatio-Temporal Anomaly Detection in a Mobile Network.* Master's thesis, Politecnico di Milano, 2015. pages 7-21.

[4] Giacomo Boracchi and Diego Carrera. *Anomaly Detection in Images.* pages 17–22. ICIAP, September 2019.

[5] Facundo Bre, Juan Gimenez, and Víctor Fachinotti. *Prediction of wind pressure coefficients on building surfaces using Artificial Neural Networks. Energy and Buildings*, 158, 11 2017. doi: 10.1016/j.enbuild.2017.11.045. Figure 1.

[6] Simon Hawkins, Hongxing He, Graham Williams, and Rohan Baxter. *Outlier Detection Using Replicator Neural Networks.* pages 113–123, 2002.

[7] Wari. *Predictive Maintenance Using Replicator Neural Network and Edge AI.* 5 2020. Figure 1, URL `https://medium.com/predictive-maintenance/predictive-maintenance-using-replicator-neural-network-and-edge-ai-6d1da43adfe4`.

[8] Giacomo Boracchi and Diego Carrera. *Anomaly Detection in Images.* pages 50–51. ICIAP, September 2019.

[9] Leo Breiman. *Random Forest. Machine Learning*, 45:5–32, 2001.

[10] Rifkie Primartha and Bayu Adhi Tama. *Anomaly detection using random forest: A performance revisited. International Conference on Data and Software Engineering (ICoDSE)*, 2017. Figure 2.

[11] Markus Breunig, Hans-Peter Kriegel, Raymond Ng, and Joerg Sander. *LOF: Identifying Density-Based Local Outliers. ACM Sigmod Record*, 29:93–104, 2000.

[12] Google Developers. *Machine Learning Crush Course - Classification: ROC*

Curve and AUC. URL https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc.

[13] Scikit-Learn. *LinearDiscriminantAnalysis Documentation*, . URL https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html.

[14] Scikit-Learn. *cross_val_score Documentation*, . URL https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html.

[15] Scikit-Learn. *OneClassSVM Documentation*, . URL https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html.

List of Figures

List of Tables