

# Intrusion Detection: Techniques

Kostas Papagiannopoulos  
University of Amsterdam  
[kostaspap88@gmail.com](mailto:kostaspap88@gmail.com) // [kpcrypto.net](http://kpcrypto.net)

# Contents

Introduction

Box Plot

Multivariate Normal Distribution

Distance Based Outliers

Local Outlier Factor

Isolation Forest

Lightweight Online Detector of Anomalies

# Introduction

# Introduction

## **Definition of an anomaly:**

*"An observation that deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism"*

# Introduction

## Definition of an anomaly:

*"An observation that deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism"*

The anomaly detection literature is fragmented in various domains

- ▶ Works on **intrusion detection** tries to detect attacks, typically in a computer system or network. The goal is to produce alerts that warn a security expert.
- ▶ Works on **outlier detection** try to detect **uncommon data patterns** in a dataset. The goal is to either analyze them further due to their importance or to discard them in order to clean the dataset.
- ▶ **Anomaly detection** is a catch-all term that includes both intrusion and outlier detection

We will use the terms intrusion, outlier and anomaly interchangeably

# Introduction

Various intrusion detection algorithms have been developed depending on:

- ▶ the data distribution and prior knowledge
- ▶ the data dimensionality i.e. the number of features in a datapoint
- ▶ the number of expected anomalies
- ▶ the various anomaly types
- ▶ computational efficiency
- ▶ algorithm robustness
- ▶ failures of existing algorithms

# Introduction

Note that we can train and test the IDS using various configurations:

- ▶ **Supervised IDS.** Train the IDS for both the normal behavior and the anomaly (2 classes)
- ▶ **Threshold IDS.** Train the IDS for both the normal behavior only and use a threshold to detect anomalies (single class)

# Introduction

Note that we can train and test the IDS using various configurations:

- ▶ **Supervised IDS.** Train the IDS for both the normal behavior and the anomaly (2 classes)
- ▶ **Threshold IDS.** Train the IDS for both the normal behavior only and use a threshold to detect anomalies (single class)
- ▶ **Untainted dataset.** Train the normal behavior of the IDS using datapoints that originated from the normal behavior only
- ▶ **Tainted dataset.** Train the normal behavior of the IDS using datapoints that may include anomalies. This could happen because we are not able to label all anomalies.

# Box Plot

# Box Plot

- ▶ We can use Tukey's box plot to detect outliers
- ▶ This technique sets the threshold  $th$  on its own

**Box plot outlier detection procedure:**

1. Assume the following dataset  $\mathcal{D}$

$$\mathcal{D} = [0.08 \quad 0.07 \quad 0.10 \quad 0.05 \quad -0.03 \quad 0.04 \quad 0.45 \quad -0.02 \quad -0.11]$$

# Box Plot

- ▶ We can use Tukey's box plot to detect outliers
- ▶ This technique sets the threshold  $th$  on its own

**Box plot outlier detection procedure:**

1. Assume the following dataset  $\mathcal{D}$

$$\mathcal{D} = [0.08 \quad 0.07 \quad 0.10 \quad 0.05 \quad -0.03 \quad 0.04 \quad 0.45 \quad -0.02 \quad -0.11]$$

2. Sort the dataset  $\mathcal{D}$

$$\mathcal{D}_{sorted} = [-0.11 \quad -0.03 \quad -0.02 \quad 0.04 \quad 0.05 \quad 0.07 \quad 0.08 \quad 0.10 \quad 0.45]$$

# Box Plot

- ▶ We can use Tukey's box plot to detect outliers
- ▶ This technique sets the threshold  $th$  on its own

**Box plot outlier detection procedure:**

1. Assume the following dataset  $\mathcal{D}$

$$\mathcal{D} = [0.08 \quad 0.07 \quad 0.10 \quad 0.05 \quad -0.03 \quad 0.04 \quad 0.45 \quad -0.02 \quad -0.11]$$

2. Sort the dataset  $\mathcal{D}$

$$\mathcal{D}_{sorted} = [-0.11 \quad -0.03 \quad -0.02 \quad 0.04 \quad 0.05 \quad 0.07 \quad 0.08 \quad 0.10 \quad 0.45]$$

3. Compute the 1st quartile  $Q_1$  i.e. 25% of the data is below this datapoint.  
MATLAB provides the quantile function.

$$Q_1 = \text{quantile}(\mathcal{D}_{sorted}, 0.25) = -0.0225$$

## Box Plot

4. Compute the 2nd quartile  $Q_2$  i.e. 50% of the data is below this datapoint (a.k.a. the median)

$$Q_2 = \text{quantile}(\mathcal{D}_{sorted}, 0.5) = 0.05$$

## Box Plot

4. Compute the 2nd quartile  $Q_2$  i.e. 50% of the data is below this datapoint (a.k.a. the median)

$$Q_2 = \text{quantile}(\mathcal{D}_{\text{sorted}}, 0.5) = 0.05$$

5. Compute the 3rd quartile  $Q_3$  i.e. 75% of the data is below this datapoint

$$Q_3 = \text{quantile}(\mathcal{D}_{\text{sorted}}, 0.75) = 0.085$$

## Box Plot

4. Compute the 2nd quartile  $Q_2$  i.e. 50% of the data is below this datapoint (a.k.a. the median)

$$Q_2 = \text{quantile}(\mathcal{D}_{\text{sorted}}, 0.5) = 0.05$$

5. Compute the 3rd quartile  $Q_3$  i.e. 75% of the data is below this datapoint

$$Q_3 = \text{quantile}(\mathcal{D}_{\text{sorted}}, 0.75) = 0.085$$

6. Compute the lower and upper thresholds  $th_{lo}$  and  $th_{up}$

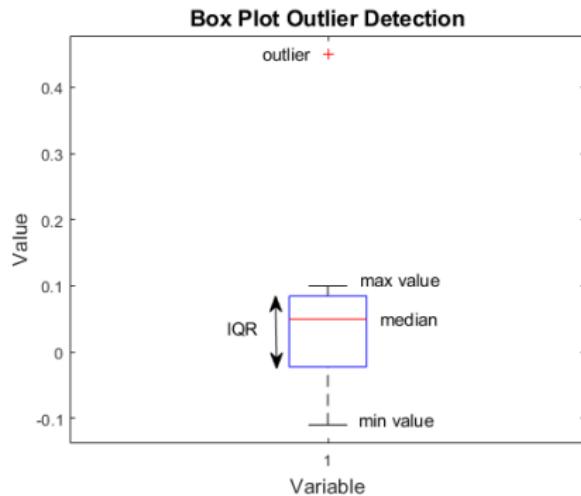
$$th_{lo} = Q_1 - 1.5 * (Q_3 - Q_1), \quad th_{up} = Q_3 + 1.5 * (Q_3 - Q_1)$$

$$th_{lo} = -0.1837, \quad th_{up} = 0.2462$$

The value  $(Q_3 - Q_1)$  is known as the interquartile range  $IQR$

# Box Plot

- ▶ Any datapoint  $o$  that is lower than  $th_{lo}$  or greater than  $th_{up}$  is considered an outlier. Thus the datapoint  $o = 0.45 > 0.2462$  is an outlier.
- ▶ Below we visualise the box plot of dataset  $\mathcal{D}$  with instruction `boxplot`



# Multivariate Normal Distribution

## Multi-dimensional IDS idea

- ▶ Most datasets have multiple dimensions
- ▶ Often anomalies can be detected using a single feature

## Multi-dimensional IDS idea

- ▶ Most datasets have multiple dimensions
- ▶ Often anomalies can be detected using a single feature
- ▶ However looking at a higher-dimensional space can reveal additional anomalies or increase our confidence about existing ones

## Multi-dimensional IDS idea

- ▶ Most datasets have multiple dimensions
- ▶ Often anomalies can be detected using a single feature
- ▶ However looking at a higher-dimensional space can reveal additional anomalies or increase our confidence about existing ones

e.g. the http KDDCUP99 dataset has 3 features: duration, src\_bytes, dst\_bytes of an http connection

Analyzing them in a joint manner can improve the accuracy of the IDS

# MVN

- ▶ We will use the **multivariate normal distribution** to model the normal system behavior
- ▶ The multivariate normal distribution is a generalization of the univariate normal distribution with  $d > 1$  dimensions

# MVN

- ▶ We will use the **multivariate normal distribution** to model the normal system behavior
- ▶ The multivariate normal distribution is a generalization of the univariate normal distribution with  $d > 1$  dimensions
- ▶ Assume dataset  $\mathcal{D} = \{\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{n-1}\}$  that contains  $n$  datapoints
- ▶ Every datapoint contains  $d$  features i.e.  $\mathbf{x}_i = [x^0 \ x^1 \ \dots \ x^{d-1}]^\top$

- ▶ We will use the **multivariate normal distribution** to model the normal system behavior
- ▶ The multivariate normal distribution is a generalization of the univariate normal distribution with  $d > 1$  dimensions
- ▶ Assume dataset  $\mathcal{D} = \{\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{n-1}\}$  that contains  $n$  datapoints
- ▶ Every datapoint contains  $d$  features i.e.  $\mathbf{x}_i = [x^0 \ x^1 \ \dots \ x^{d-1}]^\top$
- ▶ We used the univariate normal probability density function (pdf)

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2} \left(\frac{x - \mu}{\sigma}\right)^2\right)$$

- ▶ Now we will use the multivariate normal probability density function (pdf)

$$f(\mathbf{x}) = f([x^0 \ x^1 \ \dots \ x^{d-1}]^\top) = \frac{1}{\sqrt{(2\pi)^d \det(\Sigma)}} \exp\left(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right)$$

# MVN

$$f(\mathbf{x}) = f([x^0 \ x^1 \ \dots \ x^{d-1}]^\top) = \frac{1}{\sqrt{(2\pi)^d \det(\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right)$$

- ▶ We denote the  $d$ -dimensional datapoint using a random vector  $\mathbf{X}$
- ▶ The expression  $\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$  states that the datapoint follows a multivariate normal distribution

$$f(\mathbf{x}) = f([x^0 \ x^1 \ \dots \ x^{d-1}]^\top) = \frac{1}{\sqrt{(2\pi)^d \det(\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right)$$

- ▶ We denote the  $d$ -dimensional datapoint using a random vector  $\mathbf{X}$
- ▶ The expression  $\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$  states that the datapoint follows a multivariate normal distribution
- ▶ The distribution has 2 parameters that we need to specify
  - ▶ mean vector  $\boldsymbol{\mu}$
  - ▶ covariance matrix  $\boldsymbol{\Sigma}$

## Training the multivariate normal distribution

- To train the IDS we must estimate the mean vector  $\mu$  and the covariance matrix  $\Sigma$  from our trainset  $\mathcal{D}$

$$\mathcal{D} = \begin{bmatrix} \mathbf{x}_0^T \\ \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_{n-1}^T \end{bmatrix} = \begin{bmatrix} x_0^0 & x_0^1 & \dots & x_0^{d-1} \\ x_1^0 & x_1^1 & \dots & x_1^{d-1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n-1}^0 & x_{n-1}^1 & \dots & x_{n-1}^{d-1} \end{bmatrix}$$

## Training the multivariate normal distribution

- To train the IDS we must estimate the mean vector  $\mu$  and the covariance matrix  $\Sigma$  from our trainset  $\mathcal{D}$

$$\mathcal{D} = \begin{bmatrix} \mathbf{x}_0^T \\ \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_{n-1}^T \end{bmatrix} = \begin{bmatrix} x_0^0 & x_0^1 & \dots & x_0^{d-1} \\ x_1^0 & x_1^1 & \dots & x_1^{d-1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n-1}^0 & x_{n-1}^1 & \dots & x_{n-1}^{d-1} \end{bmatrix}$$

- The mean vector is estimated using the mean of every column of  $\mathcal{D}$

$$\mu = \left[ \frac{1}{n} \sum_{i=0}^{n-1} x_i^0 \quad \frac{1}{n} \sum_{i=0}^{n-1} x_i^1 \quad \dots \quad \frac{1}{n} \sum_{i=0}^{n-1} x_i^{d-1} \right]^T$$

- Apply the `mean` function on the trainset

# MVN

- ▶ The covariance matrix  $\Sigma$  is a  $d \times d$  matrix
- ▶ Every matrix element  $s_{ij}$  of  $\Sigma$  is computed as follows

$$s_{ij} = \frac{1}{n-1} \sum_{t=0}^{n-1} (x_t^i - \bar{x}^i)(x_t^j - \bar{x}^j)$$

- ▶ Apply the cov function on the trainset

## Testing the multivariate normal distribution

- ▶ As in the univariate case, we will use the pdf
- ▶ The outlier score of datapoint  $\mathbf{o}$  is:

$$\text{score}(\mathbf{o}) = f(\mathbf{o}) = \frac{1}{\sqrt{(2\pi)^d \det(\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(\mathbf{o} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{o} - \boldsymbol{\mu})\right)$$

## Testing the multivariate normal distribution

- ▶ As in the univariate case, we will use the pdf
- ▶ The outlier score of datapoint  $\mathbf{o}$  is:

$$\text{score}(\mathbf{o}) = f(\mathbf{o}) = \frac{1}{\sqrt{(2\pi)^d \det(\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(\mathbf{o} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{o} - \boldsymbol{\mu})\right)$$

- ▶ Higher score implies higher likelihood that the datapoint  $\mathbf{o}$  has normal behavior
- ▶ MATLAB can use the `mvnpdf(o, mean_vector, covariance_matrix)` function
- ▶ We can use `score(o)` in conjunction with a threshold (threshold IDS) or we could also train a multivariate normal distribution with the anomalies (supervised IDS)

# MVN

**Input:** testset  $\mathcal{T}_I = \{\mathbf{o}_0, \mathbf{o}_1, \dots, \mathbf{o}_{q-1}\}$

**Input:** distribution  $\mathcal{N}(\mu, \Sigma)$ , threshold  $th$

**Output:**  $TP, FN$  rates

```
1
2  $TPcount = 0 ; FNcount = 0$ 
3 // iterate over the elements of the testset
4 for  $i=1$  until  $q$  do
5     // compute likelihood score
6      $score = mvnpdf(\mathbf{o}_i, \mu, \Sigma)$ 
7     // count match/mismatch
8     if  $score < th$  then
9          $TPcount = TPcount + 1$ 
10    else
11         $FNcount = FNcount + 1$ 
12    end
13 end
14 // compute the rates
15  $TP = TPcount/q ; FN = FNcount/q$ 
```

- ▶ The process is repeated for the normal behavior testset  $\mathcal{T}_{\neg I}$  and results in  $TN, FP$  rates

## Curse of dimensionality

- ▶ Increasing the number of features in every datapoint (dimension  $d$ ) can cause several computational problems

## Curse of dimensionality

- ▶ Increasing the number of features in every datapoint (dimension  $d$ ) can cause several computational problems
- ▶ Computing the scores using the density function `mvnpdf` has significant computational cost and results in very small values that could underflow

e.g. `mvnpdf(zeros(1, 20), zeros(1,20), eye(20)) = 1.0428e-08`

## Curse of dimensionality

- ▶ Increasing the number of features in every datapoint (dimension  $d$ ) can cause several computational problems
  
- ▶ Computing the scores using the density function `mvnpdf` has significant computational cost and results in very small values that could underflow
  - e.g. `mvnpdf(zeros(1, 20), zeros(1,20), eye(20))` = 1.0428e-08
  
- ▶ Estimating well the mean vector  $\mu$  and covariance matrix  $\Sigma$  parameters requires a very large dataset for high dimension  $d$ 
  - e.g. if your dataset has only a small number of datapoints, would you train a univariate or a multivariate model? Undertraining may imply large estimation errors on the model parameters.

# MVN

**Using the exponent as score.** Taking the log function of the p.d.f. we reach:

$$\begin{aligned} \log(f(\mathbf{o})) &= \log\left(\frac{1}{\sqrt{(2\pi)^d \det(\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(\mathbf{o} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{o} - \boldsymbol{\mu})\right)\right) = \\ &\log\left(\frac{1}{\sqrt{(2\pi)^d \det(\boldsymbol{\Sigma})}}\right) + \log\left(\exp\left(-\frac{1}{2}(\mathbf{o} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{o} - \boldsymbol{\mu})\right)\right) = \\ &\text{constant} - \frac{1}{2}(\mathbf{o} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{o} - \boldsymbol{\mu}) \end{aligned}$$

# MVN

**Using the exponent as score.** Taking the log function of the p.d.f. we reach:

$$\begin{aligned} \log(f(\mathbf{o})) &= \log\left(\frac{1}{\sqrt{(2\pi)^d \det(\Sigma)}} \exp\left(-\frac{1}{2}(\mathbf{o} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{o} - \boldsymbol{\mu})\right)\right) = \\ &\log\left(\frac{1}{\sqrt{(2\pi)^d \det(\Sigma)}}\right) + \log\left(\exp\left(-\frac{1}{2}(\mathbf{o} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{o} - \boldsymbol{\mu})\right)\right) = \\ &\text{constant} - \frac{1}{2}(\mathbf{o} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{o} - \boldsymbol{\mu}) \end{aligned}$$

- ▶ Thus instead of  $\text{score}(\mathbf{o}) = f(\mathbf{o})$  we can use the following alternative score

$$\text{score}(\mathbf{o}) = (\mathbf{o} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{o} - \boldsymbol{\mu})$$

*drop constant cause we don't care  
absolute scores.*

**Identity covariance matrix.** If we assume the data to be homoscedastic we can avoid the covariance matrix estimation and inversion.

- ▶ This assumption implies an identity covariance matrix:

$$\Sigma = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

**Identity covariance matrix.** If we assume the data to be homoscedastic we can avoid the covariance matrix estimation and inversion.

- ▶ This assumption implies an identity covariance matrix:

$$\Sigma = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

- ▶ Then the score can be reduced to:

$$score(\mathbf{o}) = (\mathbf{o} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{o} - \boldsymbol{\mu}) = (\mathbf{o} - \boldsymbol{\mu})^\top (\mathbf{o} - \boldsymbol{\mu})$$

**Identity covariance matrix.** If we assume the data to be homoscedastic we can avoid the covariance matrix estimation and inversion.

- ▶ This assumption implies an identity covariance matrix:

$$\Sigma = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

- ▶ Then the score can be reduced to:

$$score(\mathbf{o}) = (\mathbf{o} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{o} - \boldsymbol{\mu}) = (\mathbf{o} - \boldsymbol{\mu})^\top (\mathbf{o} - \boldsymbol{\mu})$$

- ▶ Thus instead of a score that uses matrix multiplication, we can simply compute an inner product, which is much faster
- ▶ Naturally, this assumption must be checked

**Dimensionality reduction.** Not all features of a dataset are crucial to anomaly detection. It is possible to either **ignore** certain features or **compress** them in order to reduce the dimension  $d$ .

**Dimensionality reduction.** Not all features of a dataset are crucial to anomaly detection. It is possible to either **ignore** certain features or **compress** them in order to reduce the dimension  $d$ .

Subsequently the multivariate normal distribution becomes easier to train and more efficient computationally.

**Dimensionality reduction.** Not all features of a dataset are crucial to anomaly detection. It is possible to either **ignore** certain features or **compress** them in order to reduce the dimension  $d$ .

Subsequently the multivariate normal distribution becomes easier to train and more efficient computationally.

- ▶ mean/variance statistic, correlation-based techniques
- ▶ principal component analysis, linear discriminant analysis
- ▶ classifiers that automatically perform dimensionality reduction such as neural networks

# Distance Based Outliers

## DBO idea

- ▶ Assuming that the normal behavior follows a certain probability distribution is often restrictive
- ▶ In addition, distribution-based techniques can often be computationally hard to train, especially with high-dimensional data

## DBO idea

- ▶ Assuming that the normal behavior follows a certain probability distribution is often restrictive
- ▶ In addition, distribution-based techniques can often be computationally hard to train, especially with high-dimensional data
- ▶ We will simplify the anomaly detection problem and define anomaly scores for the datapoints on the basis of the **nearest neighbour problem**

# DBO

- ▶ **Distance-based outlier.** A datapoint  $\mathbf{o} \in \mathcal{D}$  is a  $DB(p, d)$  outlier if at least fraction  $p$  of the datapoints in  $\mathcal{D}$  lie in distance greater than  $d$  from  $\mathbf{o}$

# DBO

- ▶ **Distance-based outlier.** A datapoint  $\mathbf{o} \in \mathcal{D}$  is a  $DB(p, d)$  outlier if at least fraction  $p$  of the datapoints in  $\mathcal{D}$  lie in distance greater than  $d$  from  $\mathbf{o}$ 
  1. Construct the following set  $\mathcal{S}$

$$\mathcal{S} = \{\mathbf{x} \in \mathcal{D} \text{ s.t. } dist(\mathbf{o}, \mathbf{x}) \leq d\}$$

where  $dist(\mathbf{o}, \mathbf{x})$  denotes the Euclidean distance between datapoints  $\mathbf{o}$  and  $\mathbf{x}$

- ▶ **Distance-based outlier.** A datapoint  $\mathbf{o} \in \mathcal{D}$  is a  $DB(p, d)$  outlier if at least fraction  $p$  of the datapoints in  $\mathcal{D}$  lie in distance greater than  $d$  from  $\mathbf{o}$

1. Construct the following set  $\mathcal{S}$

$$\mathcal{S} = \{\mathbf{x} \in \mathcal{D} \text{ s.t. } dist(\mathbf{o}, \mathbf{x}) \leq d\}$$

where  $dist(\mathbf{o}, \mathbf{x})$  denotes the Euclidean distance between datapoints  $\mathbf{o}$  and  $\mathbf{x}$

2. Compute the set cardinality  $|\mathcal{S}|$

- ▶ **Distance-based outlier.** A datapoint  $\mathbf{o} \in \mathcal{D}$  is a  $DB(p, d)$  outlier if at least fraction  $p$  of the datapoints in  $\mathcal{D}$  lie in distance greater than  $d$  from  $\mathbf{o}$

1. Construct the following set  $\mathcal{S}$

$$\mathcal{S} = \{\mathbf{x} \in \mathcal{D} \text{ s.t. } dist(\mathbf{o}, \mathbf{x}) \leq d\}$$

where  $dist(\mathbf{o}, \mathbf{x})$  denotes the Euclidean distance between datapoints  $\mathbf{o}$  and  $\mathbf{x}$

2. Compute the set cardinality  $|\mathcal{S}|$
3. If  $|\mathcal{S}| \leq (1 - p)|\mathcal{D}|$  then the datapoint  $\mathbf{o}$  is an outlier
4. If  $|\mathcal{S}| > (1 - p)|\mathcal{D}|$  then the datapoint  $\mathbf{o}$  is normal behavior

## Example 1

- ▶ Assume the univariate dataset  $\mathcal{D}$

$$\mathcal{D} = \begin{bmatrix} 0.11 & 0.13 & 0.09 & 0.63 & 0.07 & 0.12 \end{bmatrix}$$

- ▶ Is the datapoint  $\mathbf{o} = 0.63$  a  $DB(0.6, 0.2)$ -outlier?

**Example 1**

- ▶ Assume the univariate dataset  $\mathcal{D}$

$$\mathcal{D} = \begin{bmatrix} 0.11 & 0.13 & 0.09 & 0.63 & 0.07 & 0.12 \end{bmatrix}$$

- ▶ Is the datapoint  $\mathbf{o} = 0.63$  a  $DB(0.6, 0.2)$ -outlier?

1. Construct the set  $\mathcal{S}$

$$\mathcal{S} = \{\mathbf{x} \in \mathcal{D} \text{ s.t. } dist(0.63, \mathbf{x}) \leq 0.2\} = \{0.63\}$$

2.  $|\mathcal{S}| = 1$

3. We compute  $(1 - p)|\mathcal{D}| = (1 - 0.6) * 6 = 2.4$

It holds that  $|\mathcal{S}| = 1 \leq 2.4$

**Example 1**

- ▶ Assume the univariate dataset  $\mathcal{D}$

$$\mathcal{D} = [0.11 \quad 0.13 \quad 0.09 \quad 0.63 \quad 0.07 \quad 0.12]$$

- ▶ Is the datapoint  $\mathbf{o} = 0.63$  a  $DB(0.6, 0.2)$ -outlier?

1. Construct the set  $\mathcal{S}$

$$\mathcal{S} = \{\mathbf{x} \in \mathcal{D} \text{ s.t. } dist(0.63, \mathbf{x}) \leq 0.2\} = \{0.63\}$$

2.  $|\mathcal{S}| = 1$

3. We compute  $(1 - p)|\mathcal{D}| = (1 - 0.6) * 6 = 2.4$

It holds that  $|\mathcal{S}| = 1 \leq 2.4$

Thus the datapoint  $\mathbf{o} = 0.63$  is a  $DB(0.6, 0.2)$ -outlier

i.e at least 60% of the datapoints in dataset  $\mathcal{D}$  have distance from 0.63 that is greater than 0.2

## Example 2

- ▶ Assume the univariate dataset  $\mathcal{D}$

$$\mathcal{D} = \begin{bmatrix} 0.11 & 0.13 & 0.09 & 0.63 & 0.07 & 0.12 \end{bmatrix}$$

- ▶ Is the datapoint  $\mathbf{o} = 0.11$  a  $DB(0.7, 0.02)$ -outlier?

## Example 2

- ▶ Assume the univariate dataset  $\mathcal{D}$

$$\mathcal{D} = \begin{bmatrix} 0.11 & 0.13 & 0.09 & 0.63 & 0.07 & 0.12 \end{bmatrix}$$

- ▶ Is the datapoint  $\mathbf{o} = 0.11$  a  $DB(0.7, 0.02)$ -outlier?

1. Construct the set  $\mathcal{S}$

$$\mathcal{S} = \{\mathbf{x} \in \mathcal{D} \text{ s.t. } dist(0.11, \mathbf{x}) \leq 0.02\} = \{0.13, 0.09, 0.12\}$$

2.  $|\mathcal{S}| = 3$

3. We compute  $(1 - p)|\mathcal{D}| = (1 - 0.7) * 6 = 1.8$

It holds that  $|\mathcal{S}| = 3 > 1.8$

**Example 2**

- ▶ Assume the univariate dataset  $\mathcal{D}$

$$\mathcal{D} = [0.11 \quad 0.13 \quad 0.09 \quad 0.63 \quad 0.07 \quad 0.12]$$

- ▶ Is the datapoint  $\mathbf{o} = 0.11$  a  $DB(0.7, 0.02)$ -outlier?

1. Construct the set  $\mathcal{S}$

$$\mathcal{S} = \{\mathbf{x} \in \mathcal{D} \text{ s.t. } dist(0.11, \mathbf{x}) \leq 0.02\} = \{0.13, 0.09, 0.12\}$$

2.  $|\mathcal{S}| = 3$

3. We compute  $(1 - p)|\mathcal{D}| = (1 - 0.7) * 6 = 1.8$

It holds that  $|\mathcal{S}| = 3 > 1.8$

Thus the datapoint  $\mathbf{o} = 0.11$  is not a  $DB(0.7, 0.02)$ -outlier

# DBO

## DBO detection algorithm

- ▶ The detection process takes as input  $n$  datapoints (from dataset  $\mathcal{D}$ ) and must decide if a certain datapoint  $\mathbf{o}$  is a  $DB(p, d)$ -outlier or not

```
1 DBO({x0, x1, ..., xn-1}, o, p, d)
2 count = 0
3 max_points = (1 - p) * n
4 for i=0 until n-1 do
5     if dist(xi, o) ≤ d then
6         | count = count + 1
7     end
8     if count > max_points then
9         | outlier = False
10        return outlier
11    end
12 end
13 outlier = True
14 return outlier
```

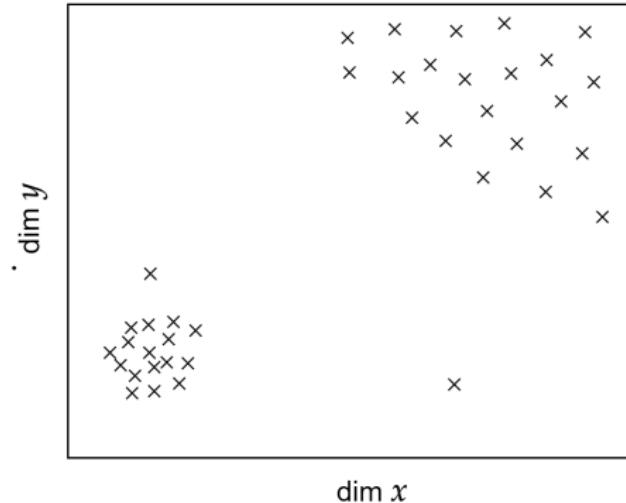
- ▶ Solving the  $DB(p, d)$ -outlier problem is equivalent to answering the **nearest-neighbour** problem
- ▶ We search in radius  $d$  around  $\mathbf{o}$  and if we find more than  $max\_points$ , then  $\mathbf{o}$  is not an outlier

## Final notes on DBO:

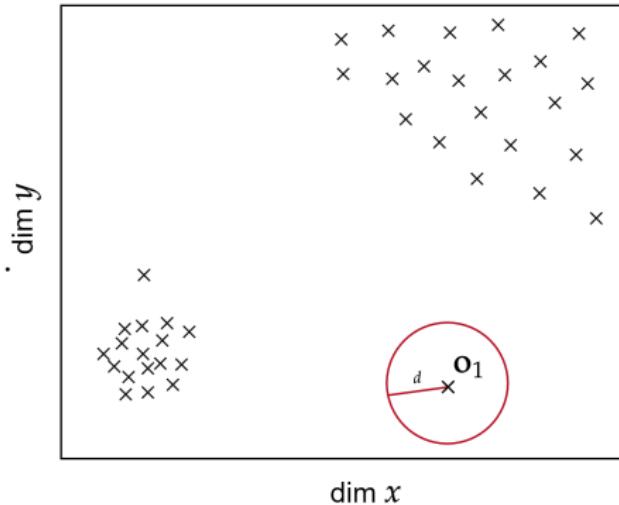
- ▶ DBO uses a generic distance-based definition of outliers, making it flexible and model-free
- ▶ Since it does not use probabilistic models it avoids slow computations like the pdf of a univariate/multivariate distribution
- ▶ The distance metric used reduces multi-dimensional datapoints to a single scalar, thus the technique can cope with a large number of dimensions

# Local Outlier Factor

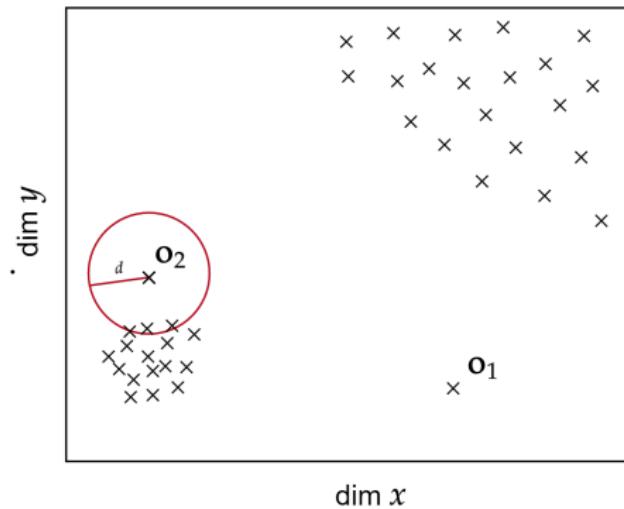
# LOF



- ▶ To introduce LOF we will consider a special dataset as example
- ▶ The 2-dimensional dataset  $\mathcal{D}$  has two main clusters

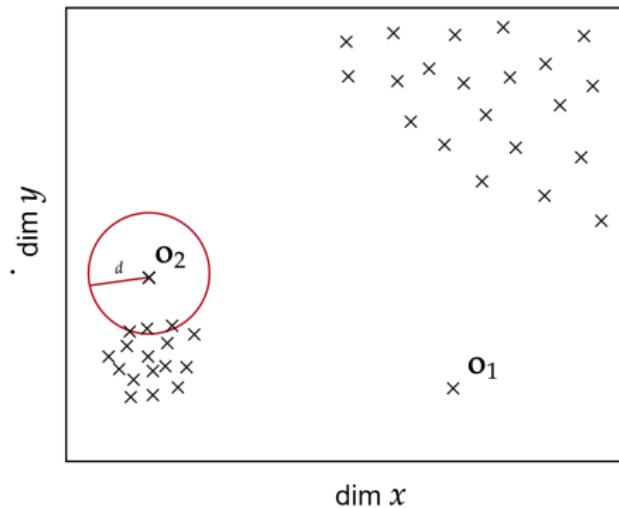


- ▶ If we search in radius  $d$  around  $\mathbf{o}_1$  then we will find no other datapoints
- ▶ Thus DBO will easily mark  $\mathbf{o}_1$  as an outlier



- ▶ What about datapoint  $\mathbf{o}_2$  though?
- ▶ If we search in radius  $d$  around  $\mathbf{o}_2$  we will find datapoints from cluster 2
- ▶ Thus, unless we pick a very small radius, DBO will not mark  $\mathbf{o}_2$  as an outlier

# LOF



- ▶ For DBO  $\text{o}_1$  is a **global outlier**
- ▶ But  $\text{o}_2$  is not a global outlier
- ▶ Still, it is an outlier **relative to the local neighbourhood**

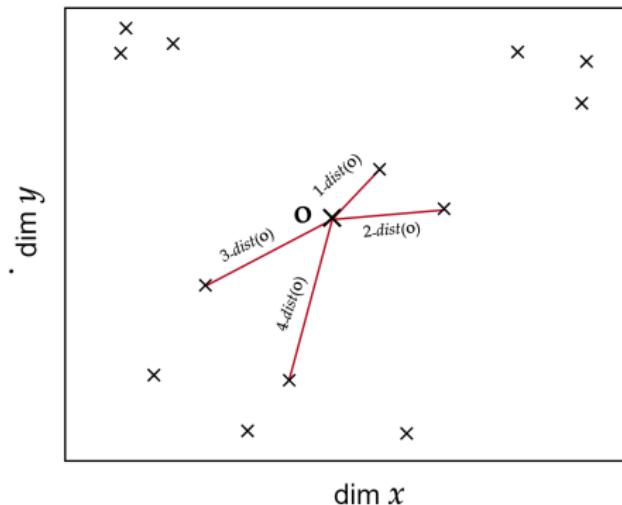
# LOF

- ▶ When clusters of different densities exist, outliers can be merged to a cluster
- ▶ This results in undetected outliers and is known as **masking**
- ▶ The **Local Outlier Factor (LOF)** is a technique that can deal with such cases

# LOF

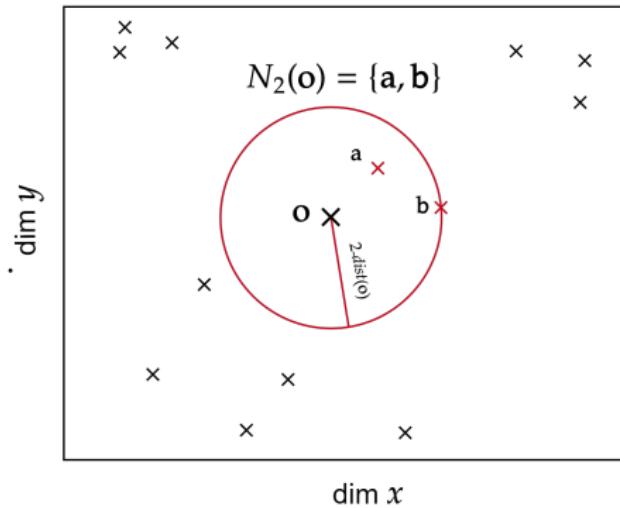
- ▶ When clusters of different densities exist, outliers can be merged to a cluster
- ▶ This results in undetected outliers and is known as **masking**
- ▶ The **Local Outlier Factor (LOF)** is a technique that can deal with such cases

**$k$ -distance.** For any integer  $k > 0$ , we define the the  $k$ -distance of datapoint  $\mathbf{o}$  as the distance of  $\mathbf{o}$  to its  $k$ -th nearest neighbour. We refer to this distance as  $k\text{-dist}(\mathbf{o})$



**$k$ -neighbourhood.** We define the  $k$ -neighbourhood of datapoint  $\mathbf{o}$  as the set of all datapoints  $\mathbf{x}$  in dataset  $\mathcal{D}$  (except  $\mathbf{o}$ ) whose distance from  $\mathbf{o}$  is not greater than the  $k$ -distance of  $\mathbf{o}$ . We refer to this set as  $N_k(\mathbf{o})$ .

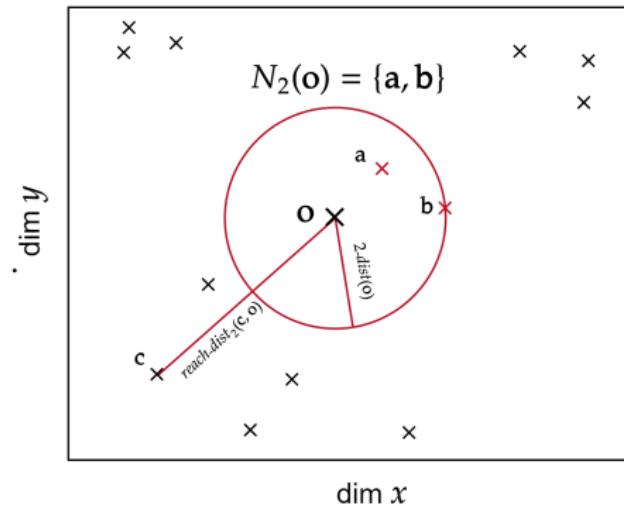
$$N_k(\mathbf{o}) = \{\mathbf{x} \in \mathcal{D} / \{\mathbf{o}\} \text{ s.t. } \text{dist}(\mathbf{x}, \mathbf{o}) \leq k\text{-dist}(\mathbf{o})\}$$



# LOF

**Reachability distance.** We define the reachability distance of datapoint  $x$  w.r.t. datapoint  $o$  as:

$$\text{reach-dist}_k(x, o) = \max\{k\text{-dist}(o), \text{dist}(x, o)\}$$

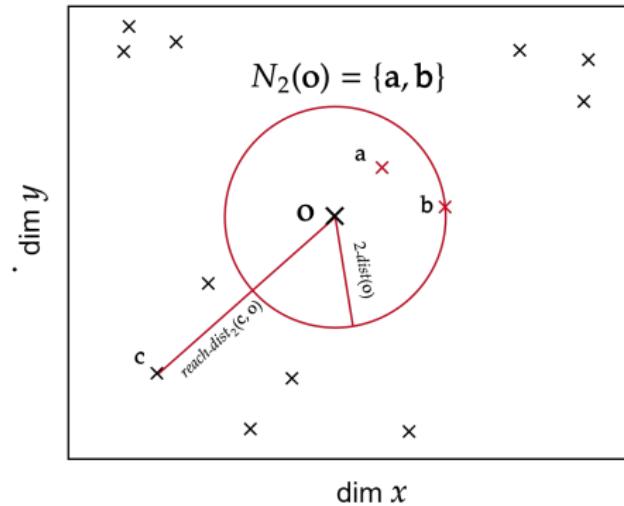


- If datapoint  $x$  is far away from datapoint  $o$  then their reachability distance is their actual distance between them  
e.g.  $\text{reach-dist}_2(c) = \text{dist}(c, o)$

# LOF

**Reachability distance.** We define the reachability distance of datapoint  $x$  w.r.t. datapoint  $o$  as:

$$\text{reach-dist}_k(x, o) = \max\{k\text{-dist}(o), \text{dist}(x, o)\}$$



- ▶ However if the datapoints are sufficiently close then the reachability distance is the  $k$ -distance of  $o$ .  
e.g.  $\text{reach-dist}_2(a) = 2\text{-dist}(o)$

# LOF

Note that  $k$  is a hyper-parameter of LOF i.e. we need to fine-tune the algorithm by trying various values of  $k$

**Local reachability density.** We define the  $lrd_k$  of datapoint  $\mathbf{o}$  as the inverse of the average reachability distances for a certain  $k$ -neighbourhood.

$$lrd_k(\mathbf{o}) = \left( \frac{\sum_{\mathbf{x} \in N_k(\mathbf{o})} \text{reach-dist}_k(\mathbf{o}, \mathbf{x})}{|N_k(\mathbf{o})|} \right)^{-1}$$

- ▶  $lrd_k(\mathbf{o})$  quantifies the density of datapoints in the neighbourhood of  $\mathbf{o}$

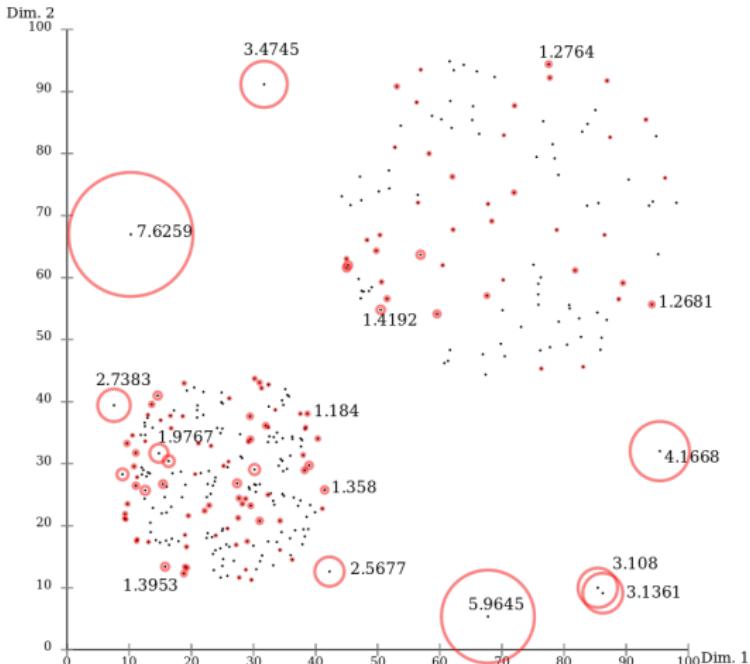
# LOF

**Local Outlier Factor (LOF).** We define the  $LOF_k$  as the average ratio of local reachabilities of  $\mathbf{o}$  and the local reachabilities of the  $k$ -neighbourhood of  $\mathbf{o}$ .

$$LOF_k(\mathbf{o}) = \frac{\sum_{x \in N_k(\mathbf{o})} lrd_k(\mathbf{x})}{|N_k(\mathbf{o})|}$$

- ▶ The LOF captures the degree to which the datapoint  $\mathbf{o}$  is an outlier

# LOF



- ▶  $LOF \approx 1$  means similar density as neighbours
- ▶  $LOF < 1$  means higher density than neighbours
- ▶  $LOF > 1$  means lower density than neighbours i.e. potential outlier

# Isolation Forest

# iForest

- ▶ Most IDSs profile the normal behavior of a system and then identify outliers
- ▶ The isolation forest algorithm (iForest) works by isolating anomalies instead of profiling the normal behavior

- ▶ Most IDSs profile the normal behavior of a system and then identify outliers
- ▶ The isolation forest algorithm (iForest) works by isolating anomalies instead of profiling the normal behavior
- ▶ iForest takes advantage of the following two properties of anomalies:
  1. Anomalies are few compared to the number of normal instances
  2. The features of an anomaly are very different from the features of normal instances
- ▶ Since anomalies are “few and different” they are more susceptible to **isolation**, compared to normal instances

## Constructing an isolation Tree (iTree)

Assume that we have the following training dataset with:

- ▶ 6 datapoints stored in vectors  $\{x_0, x_1, x_2, x_3, x_4, x_5\}$
- ▶ 5 features  $[x^0, x^1, x^2, x^3, x^4]$  for each datapoint  $x_i$

$$\mathcal{D} = \begin{bmatrix} x_0^T \\ x_1^T \\ x_2^T \\ x_3^T \\ x_4^T \\ x_5^T \end{bmatrix} = \begin{bmatrix} 4.74 & 0.71 & -0.26 & 2.79 & 5.11 \\ 4.71 & 0.75 & -0.30 & 2.50 & 5.58 \\ 4.72 & 0.82 & -0.28 & 2.61 & 5.26 \\ 4.66 & 0.72 & -0.31 & 2.62 & 5.30 \\ 4.80 & 0.71 & -0.27 & 2.71 & 5.33 \\ 4.81 & 0.77 & -0.29 & 2.72 & 5.40 \end{bmatrix}$$

# iForest

1. Randomly select a feature  $q$  from  $\{x^0, x^1, x^2, x^3, x^4\}$

e.g. we select feature  $q = x^2$

1. Randomly select a feature  $q$  from  $\{x^0, x^1, x^2, x^3, x^4\}$

e.g. we select feature  $q = x^2$

2. Randomly select a split point  $p$  between the maximum and the minimum values of the selected feature  $q$

e.g.  $\max(x^2) = -0.26$  and  $\min(x^2) = -0.31$

we select randomly  $p \in \{-0.31, -0.26\}$  and we pick  $p = -0.28$

$$\mathcal{D} = \begin{bmatrix} 4.74 & 0.71 & \textcolor{blue}{-0.26} & 2.79 & 5.11 \\ 4.71 & 0.75 & \textcolor{blue}{-0.30} & 2.50 & 5.58 \\ 4.72 & 0.82 & \textcolor{blue}{-0.28} & 2.61 & 5.26 \\ 4.66 & 0.72 & \textcolor{blue}{-0.31} & 2.62 & 5.30 \\ 4.80 & 0.71 & \textcolor{blue}{-0.27} & 2.71 & 5.33 \\ 4.81 & 0.77 & \textcolor{blue}{-0.29} & 2.72 & 5.40 \end{bmatrix}$$

3. We partition the dataset  $\mathcal{D}$  to the left and right sets  $\mathcal{D}_l$  and  $\mathcal{D}_r$  such that:

$$\mathcal{D}_l = \{\mathbf{x} \in \mathcal{D} \text{ s.t. } q < p\} \quad \text{and} \quad \mathcal{D}_r = \{\mathbf{x} \in \mathcal{D} \text{ s.t. } q \geq p\}$$

3. We partition the dataset  $\mathcal{D}$  to the left and right sets  $\mathcal{D}_l$  and  $\mathcal{D}_r$  such that:

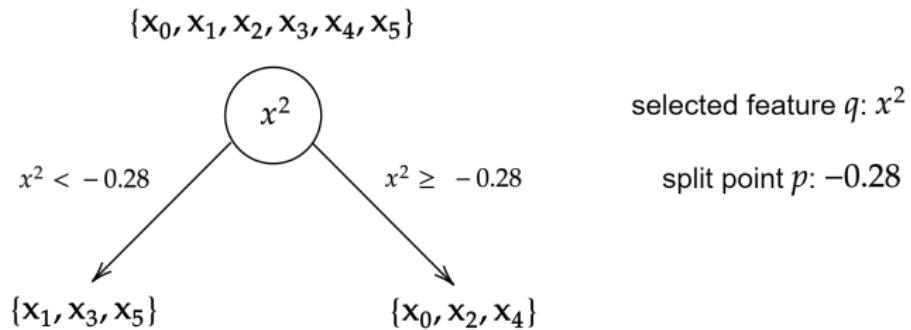
$$\mathcal{D}_l = \{\mathbf{x} \in \mathcal{D} \text{ s.t. } q < p\} \quad \text{and} \quad \mathcal{D}_r = \{\mathbf{x} \in \mathcal{D} \text{ s.t. } q \geq p\}$$

e.g. we split the dataset in the following way:

$$\mathcal{D}_l = \{\mathbf{x} \in \mathcal{D} \text{ s.t. } x^2 < -0.28\} \quad \text{and} \quad \mathcal{D}_r = \{\mathbf{x} \in \mathcal{D} \text{ s.t. } x^2 \geq -0.28\}$$

$$\mathcal{D} = \begin{bmatrix} 4.74 & 0.71 & -0.26 & 2.79 & 5.11 \\ 4.71 & 0.75 & -0.30 & 2.50 & 5.58 \\ 4.72 & 0.82 & -0.28 & 2.61 & 5.26 \\ 4.66 & 0.72 & -0.31 & 2.62 & 5.30 \\ 4.80 & 0.71 & -0.27 & 2.71 & 5.33 \\ 4.81 & 0.77 & -0.29 & 2.72 & 5.40 \end{bmatrix}$$

- ▶ We organize the partitioned sets using a binary tree structure
- ▶ We refer to this as the **isolation tree** (iTree)



# iForest

## 4. The construction of the iTree continues recursively

- We focus on the left set  $\mathcal{D}_l$

$$\mathcal{D}_l = \begin{bmatrix} 4.71 & 0.75 & -0.30 & 2.50 & 5.58 \\ 4.66 & 0.72 & -0.31 & 2.62 & 5.30 \\ 4.81 & 0.77 & -0.29 & 2.72 & 5.40 \end{bmatrix}$$

## 4. The construction of the iTree continues recursively

- We focus on the left set  $\mathcal{D}_l$

$$\mathcal{D}_l = \begin{bmatrix} 4.71 & 0.75 & -0.30 & 2.50 & 5.58 \\ 4.66 & 0.72 & -0.31 & 2.62 & 5.30 \\ 4.81 & 0.77 & -0.29 & 2.72 & 5.40 \end{bmatrix}$$

- We select randomly feature  $q = x^0$  and then select split point  $p \in \{\min(x^0), \max(x^0)\} \iff p \in \{4.66, 4.81\}$ . We pick  $p = 4.80$

$$\mathcal{D}_l = \begin{bmatrix} \textcolor{blue}{4.71} & 0.75 & -0.30 & 2.50 & 5.58 \\ \textcolor{blue}{4.66} & 0.72 & -0.31 & 2.62 & 5.30 \\ \textcolor{blue}{4.81} & 0.77 & -0.29 & 2.72 & 5.40 \end{bmatrix}$$

# iForest

## 4. The construction of the iTree continues recursively

- We focus on the left set  $\mathcal{D}_l$

$$\mathcal{D}_l = \begin{bmatrix} 4.71 & 0.75 & -0.30 & 2.50 & 5.58 \\ 4.66 & 0.72 & -0.31 & 2.62 & 5.30 \\ 4.81 & 0.77 & -0.29 & 2.72 & 5.40 \end{bmatrix}$$

- We select randomly feature  $q = x^0$  and then select split point  $p \in \{\min(x^0), \max(x^0)\} \iff p \in \{4.66, 4.81\}$ . We pick  $p = 4.80$

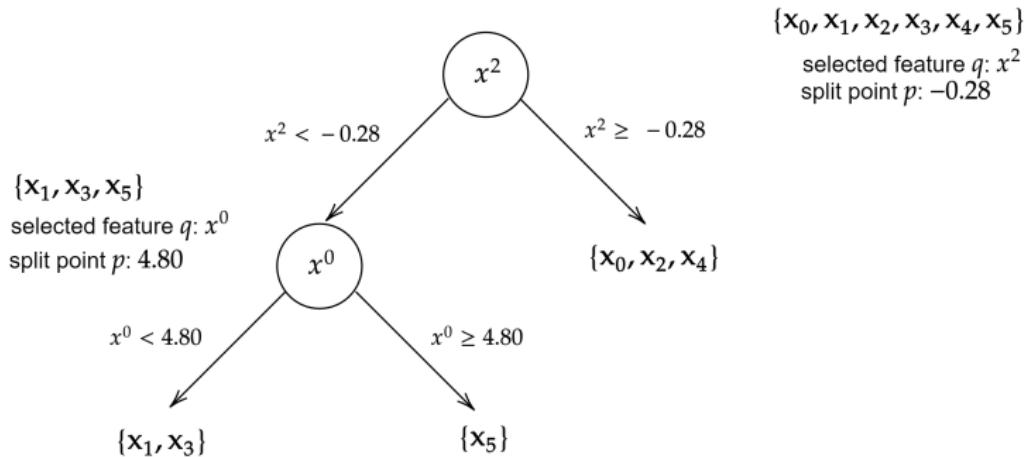
$$\mathcal{D}_l = \begin{bmatrix} \textcolor{blue}{4.71} & 0.75 & -0.30 & 2.50 & 5.58 \\ \textcolor{blue}{4.66} & 0.72 & -0.31 & 2.62 & 5.30 \\ \textcolor{blue}{4.81} & 0.77 & -0.29 & 2.72 & 5.40 \end{bmatrix}$$

- We split  $\mathcal{D}_l$  to  $\{\mathbf{x} \in \mathcal{D}_l \text{ s.t. } x^0 < 4.80\}$  and  $\{\mathbf{x} \in \mathcal{D}_l \text{ s.t. } x^0 \geq 4.80\}$

$$\mathcal{D}_l = \begin{bmatrix} \textcolor{red}{4.71} & \textcolor{red}{0.75} & \textcolor{red}{-0.30} & \textcolor{red}{2.50} & \textcolor{red}{5.58} \\ \textcolor{red}{4.66} & \textcolor{red}{0.72} & \textcolor{red}{-0.31} & \textcolor{red}{2.62} & \textcolor{red}{5.30} \\ \textcolor{red}{4.81} & \textcolor{red}{0.77} & \textcolor{red}{-0.29} & \textcolor{red}{2.72} & \textcolor{red}{5.40} \end{bmatrix}$$

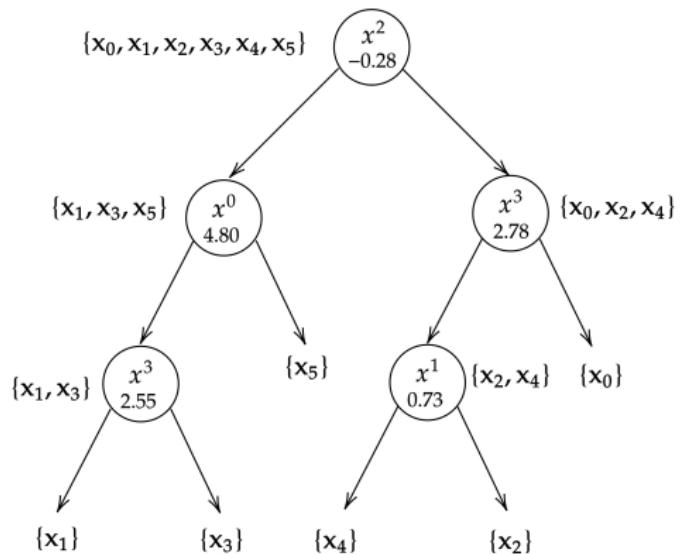
# iForest

- The iTree grows as we keep splitting the dataset



# iForest

- The full iTree is constructed once all nodes have set size 1 or when a predefined tree height limit is reached



- ▶ The iTree algorithm is initialized with dataset  $\mathcal{D}$  and possibly an upper limit on the tree height  $l$ . The function works recursively, keeping track of the current tree height  $e$ .
- ▶ During recursion, the function should also keep track of all selected features  $q$  and all the split values  $p$

```
1 iTree( $\mathcal{D}, e, l, q, p$ )
2 if  $e \geq l$  or  $|\mathcal{D}| \leq 1$  then
3   | return  $\emptyset$ 
4 end
5 select randomly feature  $q$ 
6 select randomly split point  $p \in [min(q), max(q)]$ 
7  $\mathcal{D}_l = \{\mathbf{x} \in \mathcal{D} \text{ s.t. } q < p\}$ 
8  $\mathcal{D}_r = \{\mathbf{x} \in \mathcal{D} \text{ s.t. } q \geq p\}$ 
9 return iTree( $\mathcal{D}_l, e + 1, l, p, q$ )
10 return iTree( $\mathcal{D}_r, e + 1, l, p, q$ )
```

## Testing data on an iTree

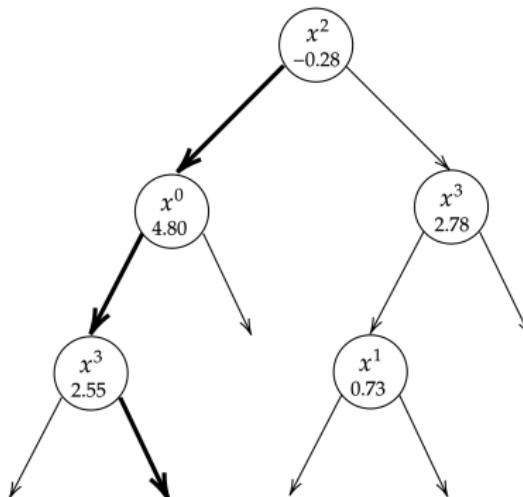
- ▶ We have used the training dataset  $\mathcal{D}$  to construct an iTree
- ▶ Let's test it with a datapoint  $x$  that is labeled as "normal behavior"

$$x = \begin{bmatrix} 4.70 & 0.73 & -0.29 & 2.65 & 5.41 \end{bmatrix}$$

## Testing data on an iTree

- ▶ We have used the training dataset  $\mathcal{D}$  to construct an iTree
- ▶ Let's test it with a datapoint  $x$  that is labeled as "normal behavior"

$$\mathbf{x} = [4.70 \quad 0.73 \quad -0.29 \quad 2.65 \quad 5.41]$$



- ▶ Passing  $\mathbf{x}$  through the iTree results in path length equal to 3

# iForest

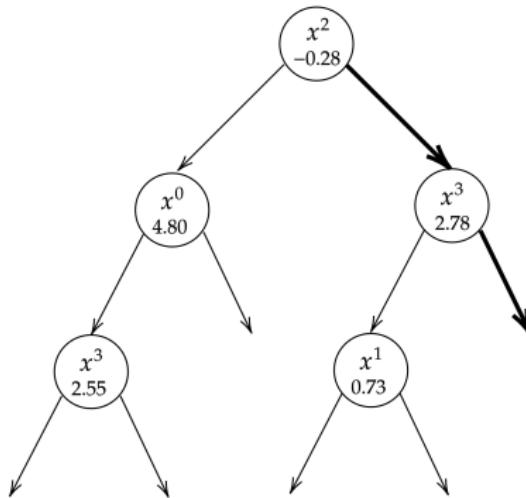
- ▶ Let's test it with a datapoint  $x'$  that is labeled as "anomaly"

$$x' = \begin{bmatrix} 5.70 & 0.71 & -0.09 & 3.51 & 5.39 \end{bmatrix}$$

# iForest

- ▶ Let's test it with a datapoint  $x'$  that is labeled as "anomaly"

$$x' = [5.70 \quad 0.71 \quad -0.09 \quad 3.51 \quad 5.39]$$



- ▶ Passing  $x'$  through the iTree results in path length equal to 2
- ▶ **iForest idea:** On average, the path length of anomalies is shorter compared to the path length of normal datapoints

## Training phase

- ▶ Constructing an iTree is a probabilistic process, since we choose randomly the feature  $q$  and the split point  $p$
- ▶ The iForest generates  $t$  iTrees (ensemble) to perform anomaly detection using the training dataset  $\mathcal{D}$
- ▶ Since the goal is not to model the normal behavior, every iTree can be constructed using a sub-sample  $\mathcal{D}'$  of the training dataset  $\mathcal{D}$  with size  $\psi < |\mathcal{D}|$

```
1 iForest( $\mathcal{D}, t, \psi$ )
2  $forest = \emptyset$ 
3 for  $i=1$  to  $t$  do
4    $\mathcal{D}' \xleftarrow{R} sample(\mathcal{D}, \psi)$ 
5    $forest = forest \cup iTree(\mathcal{D}')$ 
6 end
7 return  $forest$ 
```

## Testing phase

- ▶ Let datapoint  $x$  that we need to decide if it is “anomaly” or “normal”
- ▶ iForest gets the path lengths produced by  $x$  across all iTrees in the ensemble
- ▶ The average path length is computed and possibly normalized by factor  $c$  to produce the anomaly score

```
1 Score( $x$ ,  $forest$ )
2 for  $i=1$  to  $t$  do
3   |   tree =  $forest(i)$ 
4   |    $pl(i) = \text{PathLength}(x, tree)$ 
5 end
6  $\bar{pl} = \sum_{i=1}^t pl(i)$ 
7  $c = 2(\log(\psi - 1) + 0.5772156649) - 2(\psi - 1)/\psi$ 
8  $score = 2^{-\bar{pl}/c}$ 
9 return score
```

## Final notes on iForest

- ▶ Simple and efficient algorithm that matches the accuracy of more complex anomaly detection methods.
- ▶ iForest does not model the normal behavior, thus it can sub-sample the training dataset and work with partial models. This reduces the computational cost.
- ▶ iForest does not use a standard distance/density metric, reducing the computational cost.
- ▶ iForest has linear time complexity and low memory requirements, thus it can scale up to process large datasets and/or datasets with a large number of dimensions.
- ▶ Works effectively with high-dimensional datasets when several dimensions are not related to the anomalies.
- ▶ Deals well with masking effects

# Lightweight Online Detector of Anomalies

- ▶ The amount and dimensionality of data to process with an IDS is increasing rapidly, thus we need a method that is very efficient computationally
- ▶ We strive to react quickly to an intrusion, thus we are interested in **online** processing and detection
- ▶ Intrusions patterns change over time, thus the IDS must deal with **concept drift**
- ▶ An IDS may contain multiple observation sensors and if the sensors fail the dataset will contain missing features. Thus the trained IDS must be **robust** for such scenarios.

# LODA

## LODA idea:

- ▶ To achieve efficiency we must avoid using a single complex model/classifier
- ▶ Instead we should use an **ensemble** of simple detectors that we aggregate into a strong detector

# LODA

## LODA idea:

- ▶ To achieve efficiency we must avoid using a single complex model/classifier
- ▶ Instead we should use an **ensemble** of simple detectors that we aggregate into a strong detector
- ▶ The same idea formed also the basis for the iForest algorithm which is an ensemble of simple iTrees
- ▶ LODA simplifies the approach even more by using sparse **projections** and **histograms**

# LODA

## LODA projection:

- ▶ Very often we must process multi-dimensional datasets
- ▶ To achieve efficiency LODA applies a **projection** on every datapoint and reduces its dimension to 1
- ▶ Assume a  $d$ -dimensional datapoint  $\mathbf{x}$  and a projection vector  $\mathbf{w}$  of the same dimension

$$\mathbf{x} = \begin{bmatrix} x^0 \\ x^1 \\ \vdots \\ x^{d-1} \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} w^0 \\ w^1 \\ \vdots \\ w^{d-1} \end{bmatrix}, \quad \mathbf{x}, \mathbf{w} \in \mathbb{R}^d$$

# LODA

## LODA projection:

- ▶ To reduce the dimension of datapoint  $\mathbf{x}$  we compute the following inner product denoted as  $z$ :

$$z = \mathbf{x}^T \cdot \mathbf{w} = \begin{bmatrix} x^0 & x^1 & \dots & x^{d-1} \end{bmatrix} \begin{bmatrix} w^0 \\ w^1 \\ \vdots \\ w^{d-1} \end{bmatrix} = \sum_{i=0}^{d-1} x^i w^i$$

# LODA

## LODA projection:

- ▶ To reduce the dimension of datapoint  $\mathbf{x}$  we compute the following inner product denoted as  $z$ :

$$z = \mathbf{x}^T \cdot \mathbf{w} = \begin{bmatrix} x^0 & x^1 & \dots & x^{d-1} \end{bmatrix} \begin{bmatrix} w^0 \\ w^1 \\ \vdots \\ w^{d-1} \end{bmatrix} = \sum_{i=0}^{d-1} x^i w^i$$

- ▶ The projection vector  $\mathbf{w}$  is a sparse projection initialized in the following manner
  1.  $\mathbf{w} = [0 \quad 0 \quad \dots \quad 0]$ ,  $\mathbf{w} \in \mathbb{R}^d$
  2. Select randomly  $\lceil \sqrt{d} \rceil$  elements in vector  $\mathbf{w}$
  3. Initialize the selected elements with values sampled from the standard normal distribution  $\mathcal{N}(0, 1)$

# LODA

## LODA histogram:

- ▶ We create a **histogram**  $h$  that models the projected values  $z$
- ▶ The histogram  $h$  approximates the probability density function (pdf) of  $z$

# LODA

## LODA histogram:

- ▶ We create a **histogram**  $h$  that models the projected values  $z$
- ▶ The histogram  $h$  approximates the probability density function (pdf) of  $z$
- ▶ Assume that we project 7 datapoints  $\mathbf{x}_i$  to the respective scalars  $z_i$

$$\{\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_6\} \xrightarrow{w} \{z_0, z_1, \dots, z_6\} = [5.1 \quad -3.2 \quad 7.0 \quad 0.4 \quad 0.2 \quad -1.1 \quad 2.9]$$

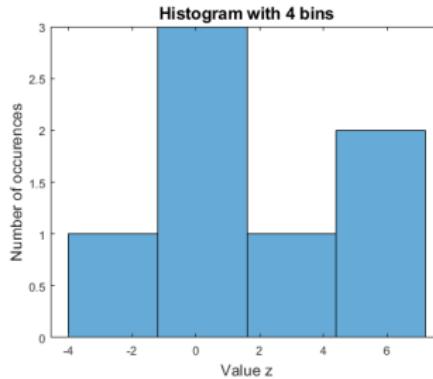
# LODA

## LODA histogram:

- ▶ We create a **histogram**  $h$  that models the projected values  $z$
- ▶ The histogram  $h$  approximates the probability density function (pdf) of  $z$
- ▶ Assume that we project 7 datapoints  $\mathbf{x}_i$  to the respective scalars  $z_i$

$$\{\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_6\} \xrightarrow{w} \{z_0, z_1, \dots, z_6\} = [5.1 \quad -3.2 \quad 7.0 \quad 0.4 \quad 0.2 \quad -1.1 \quad 2.9]$$

- ▶ Applying e.g. the `histogram(z, 4)` function we bin the  $z$  values using 4 bins



- ▶ LODA will use an ensemble of multiple histograms

# LODA

## LODA training algorithm

- ▶ Training LODA uses  $n$   $d$ -dimensional datapoints  $\{\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{n-1}\}$  as the training set
- ▶ The training procedure requires 2 hyper-parameters
  - ▶ The number of histograms  $k$  used, i.e. the ensemble size
  - ▶ The number of bins  $b$  in every histogram

```
1 LodaTrain( $\{\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{n-1}\}, k, b$ )
2 Initialize  $k$  projection vectors  $\{\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_k\}$ 
3 Initialize  $k$  histograms  $\{h_1, h_2, \dots, h_k\}$  with  $b$  bins
4 for  $j=0$  until  $n-1$  do
5   for  $i=1$  until  $k$  do
6      $z_i = \mathbf{x}_j^T \cdot \mathbf{w}_i$ 
7     update histogram  $h_i$  with  $z_i$ 
8   end
9 end
```

- ▶ The outputs are the  $k$  projection vectors  $\{\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_k\}$  and the  $k$  trained histograms  $\{h_1, h_2, \dots, h_k\}$

# LODA

## LODA score function:

- ▶ We want to compute the **anomaly score** of a  $d$ -dimensional datapoint  $\mathbf{x}$ 
  1. First we apply the projection  $\mathbf{w}$

$$z = \mathbf{x}^T \cdot \mathbf{w}$$

2. A trained histogram  $h$  provides an approximate pdf for value  $z$

$$p = h(z) \approx pdf(z)$$

# LODA

## LODA score function:

- ▶ We want to compute the **anomaly score** of a  $d$ -dimensional datapoint  $\mathbf{x}$ 
  1. First we apply the projection  $\mathbf{w}$

$$z = \mathbf{x}^T \cdot \mathbf{w}$$

2. A trained histogram  $h$  provides an approximate pdf for value  $z$

$$p = h(z) \approx \text{pdf}(z)$$

- ▶ The process is repeated for all  $k$  projections  $\mathbf{w}_i$  and trained histograms  $h_i$ .

$$z_i = \mathbf{x}^T \cdot \mathbf{w}_i, \quad p_i = h_i(z_i), \quad i = 1, 2, \dots, k$$

- ▶ The  $\log$  function is applied to the estimated likelihoods  $p_i$  and the score is computed as the average.

$$\text{score}(\mathbf{x}) = -\frac{1}{k} \sum_{i=1}^k \log(p_i)$$

## LODA Testing Algorithm

- ▶ Testing LODA uses  $m$   $d$ -dimensional datapoints  $\{\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{m-1}\}$  as the test set
- ▶ LodaTest also receives the  $k$  projection vectors  $\mathbf{w}_i$  and trained histograms  $h_i$  as inputs

```
1 LodaTest( $\{\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{m-1}\}, \{\mathbf{w}_1, \dots, \mathbf{w}_k\}, \{h_1, h_2, \dots, h_k\}$ )
2 for  $j=0$  until  $m-1$  do
3   for  $i=1$  until  $k$  do
4      $z_i = \mathbf{x}_j^T \cdot \mathbf{w}_i$ 
5      $p_i = h_i(z_i)$ 
6   end
7    $score(\mathbf{x}_j) = -\frac{1}{k} \sum_{i=1}^k \log(p_i)$ 
8 end
```

# LODA

## Final notes on LODA:

- ▶ LODA uses a computationally efficient one-dimensional histograms to detect intrusions. Histograms typically require only 'counting' operations
- ▶ The histogram can be updated on-the-fly enabling online detection and resistance to drifting
- ▶ In the case of a missing feature, LODA can apply only the projections  $w_i$  that ignore the particular feature. Thus we can use a part of the ensemble that guarantees robustness and avoids the failing sensors.