



Detection and Explanation of Contextual Anomalies in Attributed Graphs

Rémi Vaudaine

► To cite this version:

Rémi Vaudaine. Detection and Explanation of Contextual Anomalies in Attributed Graphs. Cryptography and Security [cs.CR]. Université de Lyon, 2021. English. NNT : 2021LYSES052 . tel-03726521

HAL Id: tel-03726521

<https://theses.hal.science/tel-03726521>

Submitted on 18 Jul 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



École Doctorale ED488 Sciences, Ingénierie, Santé

Numéro d'ordre NNT : 2021LYSES052

Detection and Explanation of Contextual Anomalies in Attributed Graphs

Détection et Explication des Anomalies Contextuelles dans les Graphes Attribués

Thèse préparée par **Rémi Vaudaine**
au sein de l'**Université Jean Monnet de Saint-Étienne**
pour obtenir le grade de :

Docteur de l'Université de Lyon
Spécialité : **Informatique**

Univ Lyon, Université Jean Monnet St-Etienne, CNRS, Institut d'Optique Graduate School,
Laboratoire Hubert Curien UMR 5516, F-42023, SAINT-ETIENNE, France.

Thèse soutenue publiquement le date 15/12/2021 devant le jury composé de :

Eric GAUSSIER	Professeur, Université Grenoble Alpes	Rapporteur
Christophe GRAVIER	Professeur, Université de Saint-Étienne	Examinateur
Nidhi HEGDE	Associate Professor, University of Alberta	Examinateur
Baptiste JEUDY	Maître de conférences, Université de Saint-Étienne	Co-Encadrant
Márton KARSAI	Associate Professor, Central European Univ. (Vienna)	Rapporteur
Christine LARGERON	Professeure, Université de Saint-Étienne	Directrice

Table of Contents

Introduction	1
List of Publications	5
List of Notations	7
1 Background	9
1.1 Graph mining	9
1.1.1 A few definitions	9
1.1.2 Different types of graphs	11
1.1.3 Network analysis	12
1.1.4 Mining tasks	16
1.2 Machine learning for graph mining	18
1.2.1 Machine learning basics	18
1.2.2 Deep neural networks	21
1.2.3 Graph embeddings	23
1.2.4 Graph neural networks	29
1.3 Anomaly detection	34
1.4 Explainable Artificial Intelligence	37
2 Graph contextual anomalies detection	41
2.1 Related work	44
2.1.1 Anomaly detection with vectorial data	44
2.1.2 Anomaly detection in relational data	53
2.2 Problem definition	60
2.3 Our method: CoBaGAD	62
2.4 Datasets generation	64
2.4.1 Generation of a graph with contextual anomalies	64
2.4.2 An illustrative example	65
2.5 Experimental evaluation of CoBaGAD	66
2.5.1 Datasets	67
2.5.2 Experimental setup	68

2.5.3	Results	69
2.5.4	Number of layers: hyper-parameter tuning	72
2.6	Conclusion	73
3	Explaining anomaly classification in graphs.	75
3.1	Related work	76
3.1.1	Explainability in the context of vector data	76
3.1.2	Explainability in the context of relational data	80
3.2	Problem definition	84
3.2.1	Contextual anomaly detection	84
3.2.2	Classification problem formalization	84
3.3	Our method	85
3.3.1	Local sample as perturbations of the neighborhood	85
3.3.2	Learning the local model	87
3.3.3	Feature importance in a decision tree	87
3.3.4	Explaining the prediction of a contextual anomaly by the black-box model	88
3.4	Measures: Fidelity, efficiency, and quality of an explanation	89
3.4.1	Fidelity	89
3.4.2	Efficiency	90
3.4.3	Precision and recall	90
3.5	Experiments	91
3.5.1	Datasets	91
3.5.2	Experimental setup	91
3.6	Results	93
3.6.1	Fidelity	93
3.6.2	Efficiency	93
3.6.3	Quality of the explanation	94
3.6.4	Hyper-parameter tuning	97
3.7	Conclusion	98
	Conclusion	101
	Acknowledgement	103
	Bibliography	105

Introduction

In the recent few years, network data has become ubiquitous and has attracted great interest in the data mining community. It is used in many different areas. Social networks such as Facebook, Twitter, or real-life social circles are inherently graphs with a lot of potential applications like recommending friends, finding groups of people, bots or spammers, malicious accounts that spread fake news, hate speakers, the most influential people and even studying the spread of information or diseases. The Web is also a network in which we want to identify hubs and try to improve the accuracy of searches. In telecommunication networks such as 4G and 5G or e-mails networks, one may want to identify entities whose behavior deviates from normality or the ones who spread malicious information. The distribution of electricity, gas, and water are also networks in which we want to find anomalies, prevent breakdowns or peaks in consumption. On the digital security side, virus propagation, intrusion of machines, and detection of unauthorized computers are applications of network analysis. Graphs keep on growing bigger and bigger with roughly 3 billion Facebook users, 200 million daily Twitter users and 45 billion webpages. New methods have been needed to deal with such datasets. Recently, graph mining has been revolutionized by machine learning models. Graph embeddings and graph neural networks are very efficient tools to reduce the complexity of analyzing graphs. Moreover, they improve a lot the ability to perform certain tasks on graphs.

On the other hand, anomaly detection is an important problem in many application domains. Anomaly detection aims at finding abnormal instances of the data. There are a lot of possible applications such as health monitoring with the detection of anomalies in radiographies or electrocardiograms, fraud detection, special event detection, defects in industry, or anomalous behaviors. For example, it is estimated that there are between 25 and 100 billion euros of tax fraud in France. In 2020, 25% of the tax audits were automated with the use of data analysis tools. The receipts of the tax have grown by 30% between 2019 and 2020. The rise of anomaly detection tools is leading to major improvements in a lot of places. It is necessary to further these developments and to improve the current methods. Another challenge to anomaly detection is the fact that it deals with the occurrence of rare instances in the data. This often implies dealing with very unbalanced data. Unbalanced data processing also complicates the analysis of the data.

Finally, explainability in the domain of artificial intelligence, XAI, has become very important with the rise of deep learning models. Indeed, deep learning has brought huge improvements

in text mining, computer vision or signal processing. But it has also brought a lack of understandability since most of the best models are not interpretable. For instance, convolutions or attention mechanisms are not inherently understandable. Thus, an emerging field of research is Explainable Artificial Intelligence (XAI) that focuses on how to generate meaningful explanations that can improve trust in the different models, improve the transparency of decisions, avoid biases from models or datasets or simply understand the output of a machine learning tool.

This thesis lies at the intersection of these three domains: graph mining for anomaly detection and explainability.

Outline of the thesis. This manuscript is composed of a background chapter followed by two chapters each presenting one of our contributions.

- Chapter 1 is an introduction to the different topics discussed in this thesis. First, we introduce the concept of graph and the tools to deal with them which are graph mining notions. A few definitions are given and a general presentation to network analysis, graph embedding techniques, and graph neural networks is provided. Second, we introduce the notion of anomaly detection and discuss one of its key limitations: the curse of dimensionality. Finally, we give insights of the notion of explainable artificial intelligence. The linear models and decision trees are introduced to, then, present their limitations. More accurate models are also introduced such as deep neural networks. While these new machine learning models improved a lot of different aspects of the decision process, they also are not inherently understandable. To solve this issue, new methods to explain are necessary.
- Chapter 2 is dedicated to the definition and the detection of contextual anomalies in attributed graphs which are a new kind of anomalies. Anomalies in graphs are not always clearly defined and often depend on the application. For example, power laws have been largely used in graph mining and anomalies could be those nodes that deviate a lot from the laws [Akoglu et al., 2010]. While this definition is easy to understand, it is model-specific as the choice of the laws is crucial. Moreover, such methods do not provide a clear definition of the anomalies as they do not focus on specific nodes of the graph. In an unsupervised setting, where no labeled data is available during the training, a model cannot focus on specific nodes of the graph since it cannot have user-guided information. On the other hand, if a few labeled examples are fed to the model during the training, it can focus on nodes of the same type. This approach allows detecting nodes of the graph that share the same kind. Thus, if we already know a few anomalies in the graph, then we can detect all the anomalies of the same type. In this thesis, we defined contextual anomalies as nodes of the graph depicted by a special local context around that node. Then, in a semi-supervised way, this type of anomaly can be detected thanks to graph neural networks with attention mechanisms. Intensive experiments prove the efficiency of

the approach and show that it outperforms both semi-supervised and unsupervised state-of-the-art models in the detection of contextual anomalies. Tuning of the parameters is discussed too so that any user can easily detect its own anomalies.

- Chapter 3 is devoted to the presentation of our graph classification explanatory model. Explainability in the context of machine learning has significantly grown in the past few years. The improvement of machine learning models in terms of computer vision and natural language processing have brought a lack of understandability of the methods. This is mostly due to the use of neural networks. In the context of node classification in graphs, explanatory models often rely on finding either important nodes of the graph that led to a specific classification or important features describing the nodes. A new method is introduced, that is simple, understandable and that can distinguish between both nodes and features to extract only the relevant nodes and their features that imply the anomaly detection. Experiments demonstrate that this new model can accurately find the relevant information that leads a deep model to a specific prediction. It also greatly outperforms state-of-the-art methods since it is more precise. Finally, hyper-parameter tuning is discussed.

List of Publications

Publications in International Conferences

Rémi Vaudaine, Christine Largeron and Baptiste Jeudy. Explaining anomaly classification in graphs. In *submitted*, 2021.

Rémi Vaudaine, Christine Largeron and Baptiste Jeudy. Detection of contextual anomalies in attributed graphs. In *Intelligent Data Analysis (IDA)*, Porto, 2021, 338-349, [Vaudaine et al., 2021a].

Rémi Vaudaine, Christine Largeron and Rémy Cazabet. Comparaison de la capacité des plongements de graphes à capturer les propriétés des réseaux. In *Extraction et Gestion de Connaissances (EGC)*, 2021, 517-518, [Vaudaine et al., 2021b].

Rémi Vaudaine, Christine Largeron and Rémy Cazabet. Comparing the preservation of network properties by graph embeddings. In *Intelligent Data Analysis (IDA)*, Konstanz, 2020, 522-534, [Vaudaine et al., 2020].

Aakash Sinha, Rémy Cazabet and Rémi Vaudaine. Systematic Biases in Link Prediction: comparing heuristic and graph embedding based methods. In *Complex Network*, Cambridge, 2018, 81-93 [Sinha et al., 2018].

Rémi Vaudaine, Christine Largeron and Rémy Cazabet. Comparing the preservation of network properties by graph embeddings. In *Complex Network*, Cambridge, 2018, 168-170 [Vaudaine et al., 2018].

List of Notations

$\mathcal{G}(\mathcal{V}, \mathcal{E})$	A graph on a set of vertices \mathcal{V} with a set of edges \mathcal{E}
$\mathcal{G}(\mathcal{V}, \mathcal{E}, X)$	A graph on a set of vertices \mathcal{V} with a set of edges \mathcal{E} and a feature matrix X
$\mathcal{V} = \{v_i\}$	The set of nodes
$\mathcal{E} = \{e_{ij}\}$	The set of edges
n	Number of nodes of a graph
m	Number of edges of a graph
$\mathcal{N}(v_i)$	The set of neighbors of the node v_i
d_i	The degree of the node v_i
A	The adjacency matrix of a graph
Z	The embedding matrix of a graph
<hr/>	
\mathcal{X}	A set
X	A matrix
\mathbf{x}	A vector
x	A scalar

Chapter 1

Background

Three main themes will be discussed throughout this thesis. First, graph mining has seen a renewal of interest in the last few years with the arrival of machine learning tools. Many approaches have been developed to reduce the inherent complexity of graphs and to improve the management of such complex data. The second theme is anomaly detection and more specifically in the context of relational data. Anomaly detection has been dealt with distance-based or density-based metrics. Nowadays, machine learning also took place in the anomaly detection domain and those techniques improved the detection of anomalies in both regular and graph data. In the end, anomaly detection and graph mining have been radically changed with the input of machine learning models. But machine learning and especially deep neural networks are rarely human-understandable because of their intrinsic complexity. Thus, the third theme is explainable artificial intelligence, *XAI*. It consists of developing new models that are either inherently understandable or that make sense of a non-interpretable model. This chapter is dedicated to background knowledge of these three topics that are essential to understand the rest of the thesis. We also address their mixing into the framework of contextual anomalies detection in chapter 2 and explanation in chapter 3.

1.1 Graph mining

1.1.1 A few definitions

Definition 1.1.1 (Graph). *A static, unweighted graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is a structure built on a set of nodes $\mathcal{V} = \{v_i\}_{i=0}^{n-1}$, the vertices and a set of edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ which is a set of unordered pairs of nodes.*

Example 1.1.1. *For instance, in a social network (or social graph), nodes can be people and edges exist whether people know each other. But there can be many other graphs: nodes are people and edges exist whether someone liked another one's post.*

Graphs grow bigger every day with new users joining Facebook or Twitter for example. Such graphs have billions of nodes and thousands of billions of edges. Thus, new tools are necessary

to study them and to be able to extract relevant information from them. Due to the important size of such datasets, efficient storage methods had to be proposed. Two criteria are important to compare the different ways to store a graph: the amount of memory necessary to store the graph and the time to compute some specific values. A typical time-consuming task is to find the list of neighbors of a node. There are three main ways to read a graph. First, there is the edge list where every edge is listed such that we know every connection in the graph. The nodes are the extremities of the edges. This is a very efficient way to store a graph in terms of space complexity but it is not very efficient in terms of time of search to get the list of neighbors of a node as it may be necessary to read the whole edge list to find every neighbor. Secondly, the adjacency list gives, for every node, the set of its neighbors. It is a lot faster to look for the list of neighbors of a specific node since it is directly stored this way. Last, the third way to store a graph is through adjacency matrix A .

Definition 1.1.2 (Adjacency matrix). *The adjacency matrix A of a graph is a square matrix where nodes are in rows and columns. Each element of the matrix a_{ij} is 1 if and only if there is an edge between the nodes v_i and v_j and 0 otherwise.*

Moreover, this matrix is usually sparse and can be stored accordingly. Throughout this thesis, unless specified otherwise, graphs will be stored as a sparse adjacency matrix. With the sparse version, the space complexity is the same as the edge list and the time to get the set of neighbors of a node is nearly the same as the adjacency list. Furthermore, the adjacency matrix framework offers the possibility to do matrix operations on the graph which will be very useful.

Example 1.1.2. *The undirected unweighted graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \{1, 2, 3, 4, 5, 6\}$ and $\mathcal{E} = \{(1, 2), (1, 5), (2, 3), (2, 4), (3, 4), (3, 5), (3, 6)\}$ is represented on Figure 1.1 on the left and its adjacency matrix is on the right.*

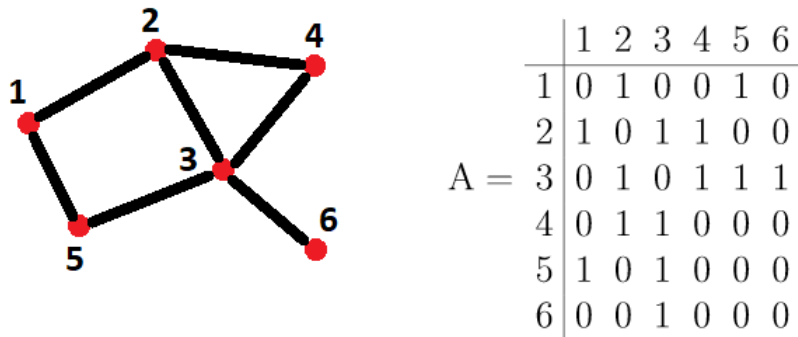


Figure 1.1: Illustration of an undirected unweighted example graph. A small graph on the left and its adjacency matrix on the right.

1.1.2 Different types of graphs

Graphs can be of many different types according to their composition. This is a non-exhaustive list of the most important types of graphs:

- Oriented graphs
- Bipartite graphs
- Attributed graphs
- Weighted graphs
- Temporal graphs

Definition 1.1.3 (Oriented graph). *An oriented graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is built on a set of nodes V and a set of edges E which is a set of ordered pairs of nodes. Let e_{ij} be the edge between node v_i and node v_j . Contrary to the undirected case, $e_{ij} \in \mathcal{E} \not\Rightarrow e_{ji} \in \mathcal{E}$.*

Example 1.1.3. *A typical example of an oriented graph is an e-mail network where nodes of the graph are e-mail addresses and edges are whether an e-mail has been sent. In this case, an e-mail is sent by someone to someone else indicating that the action is from a certain node towards another. This is the signature of oriented graphs.*

Definition 1.1.4 (Bipartite graph). *In a bipartite graph, the set of nodes \mathcal{V} can be split into two disjoint subsets \mathcal{V}_1 and \mathcal{V}_2 such that $\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2$, $\mathcal{V}_1 \cap \mathcal{V}_2 = \emptyset$ and each edge has one extremity in each subset.*

Example 1.1.4. *For instance, let us consider a graph with two sets of nodes. We have a set of people \mathcal{V}_1 and a set of videos \mathcal{V}_2 . Edges depict the fact that a specific user $v_i \in \mathcal{V}_1$ watched a specific video $v_j \in \mathcal{V}_2$. This case of a bipartite network can be, for example, used to build a recommender system in environments like YouTube, Netflix or Amazon.*

Definition 1.1.5 (Attributed graph). *An attributed graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, X)$ has additional information on their nodes and/or their edges. Nodes and edges of the graph can have their own feature vector that describes them. The set of node's features is represented by the feature matrix $X \in \mathbb{R}^{n \times f}$ where f is the number of features per node.*

Example 1.1.5. *In a social network, people can be described by a lot of features that can be numerical such as height, age, weight, or categorical like gender. Such graphs are nowadays at the core of many research activities as they stir together both graph mining and tabular data mining.*

Definition 1.1.6 (Weighted graph). *A weighted graph is a particular instance of an attributed graph where the additional information only lies on the edges. In a weighted graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, W)$, $w_{ij} \in \mathbb{R}$ is the weight associated with the edge e_{ij} . For a node v_i , the set of its neighbors is $\mathcal{N}(v_i) = \{v_j \in V | e_{ij} \in \mathcal{E}\}$ and the total weight of this node v_i is $w_i = \sum_{v_j \in \mathcal{N}(v_i)} w_{ij}$.*

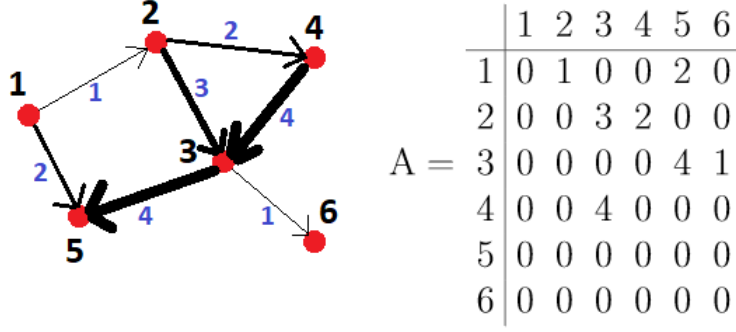


Figure 1.2: Illustration of a directed and weighted example graph. A small graph on the left and its adjacency matrix on the right.

Example 1.1.6. The directed weighted graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \{1, 2, 3, 4, 5, 6\}$ and $\mathcal{E} = \{(1, 2), (1, 5), (2, 3), (2, 4), (3, 4), (3, 5), (3, 6)\}$ is represented on Figure 1.2 on the left and its adjacency matrix is on the right.

Definition 1.1.7 (Temporal graphs). Temporal graphs are graphs that evolve over time. Nodes and edges can be created or deleted and their features can change. A temporal graph can be represented by a set of static graphs $\{\mathcal{G}_t(\mathcal{V}_t, \mathcal{E}_t, X_t)\}_{t=0}^{T-1}$ at different timesteps $t \in \{0, 1, \dots, T-1\}$ where T is the final timestep. Graphs that do not evolve over time are called static graphs.

Example 1.1.7. For instance, the Facebook network where the nodes are people and edges are friendship's relations evolves over time. Each day, some users sign up on the platform creating new nodes on the graph, new edges are created if some users become friends and the user's features, like age or place of residence, may also change.

To conclude, we presented several kinds of graphs: oriented, bipartite, attributed, weighted and temporal graphs. Our work focuses on undirected static attributed graphs. While temporal graphs are a natural extension of the proposed work, those will not be discussed here. To deal with this type of relational data, a lot of work has been done since the first experiments of Milgram [Milgram, 1967].

1.1.3 Network analysis

1.1.3.1 Visualization

Some of the first works on network analysis consisted in visualization. Visualization is the process of representing the network in a way that we can look at it with relevant information. Some layouts have been proposed to tackle this issue [Fruchterman and Reingold, 1991] [Kamada and Kawai, 1989]. For its part, graph dimension reduction consists of reducing the inherent structural complexity of a graph and, in two dimensions, joins up with visualization. While it has seen significant improvements in the early 2000s [Belkin and Niyogi, 2003] [Roweis and

Saul, 2000] by improving the complexity of state-of-the-art methods, it is still an ongoing field of research due to the immense growth of graphs' size and complexity.

Example 1.1.8. *The Zachary Karate Club is a famous graph of 34 nodes and 78 edges [Zachary, 1977]. Its visualization through the Fruchterman-Reingold [Fruchterman and Reingold, 1991] layout is shown in Figure 1.3.*

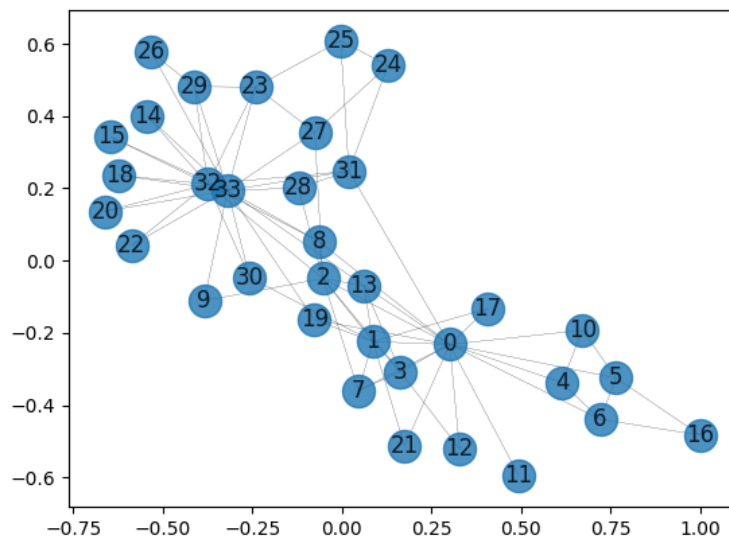


Figure 1.3: Visualization of the Zachary Karate Club graph using Fruchterman-Reingold layout.

1.1.3.2 Metrics and centralities

Moreover, different metrics and measures have been proposed to better understand the inner workings of the graphs.

Definition 1.1.8 (Shortest path). *A path in a graph is a sequence of vertices such that every consecutive pair of vertices in the sequence is connected by an edge in the network. The shortest path between two nodes is the path with the minimum number of edges, weighted if necessary. The geodesic distance is the length of this shortest path between two nodes.*

The shortest path between two nodes is the easiest way to go from a specific node to another one. For a whole graph, it is possible to compute two different metrics. The average path length is the mean of all the geodesic distances and the diameter is the maximum geodesic distance over every pair of nodes. There are two famous experiments on the computation of the average path length in our world. First, Stanley Milgram planned to send some letters from Omaha or Wichita in the USA to Boston. Some random people of the starting cities received the letters. These letters had to be handed to a specific person in Boston. In the more likely case, the first carriers did not know the target. Thus, they had to give the letter to a friend who was

likely to know the target. If the letter reached the contact in Boston, then the researcher could examine the length of the path, *i.e.* the number of people that held the letter before coming to Boston. The results suggested an average path length close to 6. This is known as the "six degrees of separation" or the small-world phenomenon. The second experiment has been conducted by Leskovec and Horvitz more recently [Leskovec and Horvitz, 2008]. They compute the average path length of the communication network of the MSN Messenger software. With the knowledge of the whole graph, it is possible to compute the shortest path between any pair of nodes and average it to have the average path length. This experiment concluded that the average path length was 6.6 in this graph. This value confirms the small-world phenomenon. Nowadays, common social networks such as Facebook or Twitter have higher connectivity which leads to lower values of the average path length.

A typical metric to measure the connectivity of a graph is density. For a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with n nodes and m edges, the density is defined as the ratio between the total number of edges m and the maximum possible number of edges if the graph were fully connected.

$$\text{density}(\mathcal{G}) = \frac{2 \times m}{n(n-1)} \quad (1.1)$$

Moreover, the local clustering coefficient of a node v_i is the density of its ego-network which is the subgraph \mathcal{EN}_i induced by the set of nodes \mathcal{N}_i , the set of neighbors of v_i and by the set of edges between those nodes. At the scale of the whole graph, it is possible to compute the average clustering coefficient. A high clustering coefficient indicates a tendency of the neighbor of a node to be connected together and a tendency of the network to exhibit a community structure.

Furthermore, centralities are measures of importance of nodes in a graph. Four centralities are commonly used. First, the degree centrality of a node is the simplest centrality. For a specific node v_i , it is defined as its degree $d_i = |\mathcal{N}_i|$, or its number of neighbors. Then, the betweenness centrality is defined as:

$$\text{betweenness}(v_i) = \sum_{v_s \neq v_i \neq v_t} \frac{\sigma_{st}(v_i)}{\sigma_{st}} \quad (1.2)$$

where σ_{st} is the total number of shortest paths from v_s to v_t and $\sigma_{st}(v_i)$ is the number of those paths passing by v_i . Thirdly, the closeness centrality of a node is the inverse of its farness. The farness is defined as the average of the geodesic distances to the other nodes of the graph. Finally, the eigenvector centrality is defined thanks to the eigenvalue problem $A\phi = \lambda\phi$ where A is the adjacency matrix, ϕ is the eigenvector associated to the largest eigenvalue λ . Then, the eigenvector centrality of a node v_i is:

$$\phi_i = \frac{1}{\lambda} \sum_{j=0}^{n-1} a_{ij} \phi_j \quad (1.3)$$

These centralities give insights into the behavior of a graph. They can have different interpretation as illustrated in Table 1.1.

Metric	Meaning
Degree centrality	How many people can this person reach directly?
Betweenness centrality	How likely is this person to be in communication paths?
Closeness centrality	How fast can this person spread information in the graph?
Eigenvector centrality	How well is this person connected to other well-connected people?

Table 1.1: Different metrics that have been proposed to make sense of graphs and their interpretations. Source: Giorgos Cheliotis - Social Network Analysis Lecture.

1.1.3.3 Heavy-tailed distributions

According to [Aggarwal, 2011], the Gaussian distribution is common in nature but there are many cases where the probability of events far to the right of the mean is significantly higher than in Gaussians. For example, on Instagram, only a small number of users have more than a million followers whereas a very large majority of people have less than a thousand followers. Heavy-tailed distributions attempt to model this. They are known as "heavy-tailed" because they decay polynomially quickly instead of exponentially creating a "fat tail" from extreme values on the probability density function plot. Power laws are well-known distributions used to fit observations. For example, in Figure 1.4, the distribution of the degrees among the nodes is plotted and, in a log-log scale, can be approximated by a straight line.

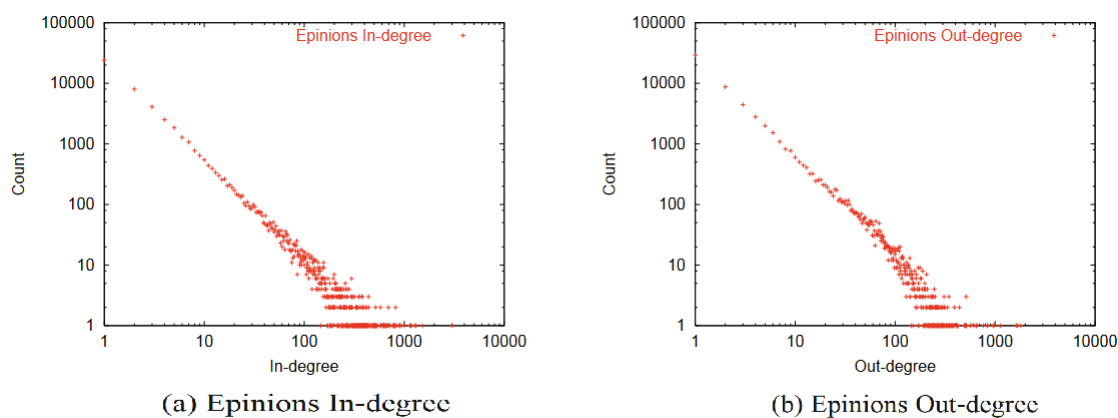


Figure 1.4: Power laws and deviations: in-degree and out-degree distributions on a log-log scale for the Epinions graph (an online social network of 75888 nodes and 508960 edges [Domingos and Richardson, 2001]).

1.1.3.4 Network models

One of the first network models has been proposed by P. Erdős, and A. Rényi [Erdős and Rényi, 1959]. It consists in creating a graph with n nodes and a probability p that each edge is present, independently of the other edges. The distribution of the degree of any particular vertex is binomial. The network has different properties depending on n and p . If the product

np is less than 1, then the graph will almost surely have no connected components of size larger than $O(\log(n))$. On the contrary, if $np > 1$, then the graph will almost surely have a unique giant component containing a positive fraction of the vertices. And if $p > \frac{(1+\epsilon)\ln(n)}{n}$, then the generated graph will almost surely be connected.

The Barabasi-Albert model begins with a connected network of m_0 nodes [Barabasi and Albert, 1999]. Then, new nodes are added one at a time and connected to $m < m_0$ nodes. The probability p_i that the new node is connected to node v_i is:

$$p_i = \frac{d_i}{\sum_j d_j} \quad (1.4)$$

where d_i is the degree of the node v_i and the sum is computed over all pre-existing nodes. Thus, heavily linked nodes tend to accumulate even more links while nodes with few links are unlikely to be chosen. This phenomenon is called preferential attachment since new coming nodes prefer to attach to high-degree nodes. The degree distribution of a Barabasi-Albert graph is a power-law.

Many other network models have been proposed. Some of them try to mimic real-world network's behavior. The LFR benchmark [Lancichinetti et al., 2008] is a network model that can create graphs with communities while Dancer [Largerion et al., 2017] can generate dynamic attributed graphs with communities.

1.1.4 Mining tasks

1.1.4.1 Community detection

Community detection aims at partitioning the nodes of a graph in such a way that nodes belonging to the same community are densely connected to each other while sparsely connected to the rest of the network [Fortunato and Hric, 2016] [Fortunato, 2010]. Thus, a community in a network is a group of nodes with high internal density and low external density. For instance, a community can be described as a group of people working at the same company or a group of people belonging to the same family. This informal definition has been quantified in many different ways and usually by quality functions. A quality function measures the goodness of a partition of a network into several communities. The modularity [Newman, 2006] is one of the most popular quality functions even though none has gained universal acceptance. One of its advantages is that it does not depend on the number of clusters that the graph is divided into. The basic idea of modularity is to compare the strength of the relationship in the community with those equivalent to a random subgraph with the same number of nodes, the same number of edges, and the same degree distribution. The farther the community is from a random subgraph, the better the community structure is. The modularity Q of a partition of the graph

into k communities $\{\mathcal{V}_1, \dots, \mathcal{V}_k\}$ is given by:

$$Q = \frac{1}{2m} \sum_{l=1}^k \sum_{v_i \in \mathcal{V}_l, v_j \in \mathcal{V}_l} \left(a_{ij} - \frac{d_i d_j}{2m} \right) \quad (1.5)$$

where m is the number of edges of the graph, d_i is the degree of v_i and a_{ij} is the element ij of the adjacency matrix.

The modularity takes values between -1 and 1 . A partition exhibits a good community structure when the modularity is greater than 0.3 [Newman, 2004] and a very good community structure when the modularity is greater than 0.6 [Largeron et al., 2015]. While modularity can be used to measure the quality of a partition, it is also possible to take advantage of it to detect the communities. Newman proposed a greedy clustering algorithm for optimizing modularity [Newman, 2004]. The nodes start each in their own community. Then, at each step, one chooses the two communities whose merger leads to the biggest increase in modularity. Another method to detect communities has been proposed by [Newman and Girvan, 2004]. The basic assumption is that there are only a small number of edges between communities. Thus, their betweenness centrality will be high. The definition of this centrality for edges is the same as the one in equation 1.2. The main idea of this method is to find the edges with the highest betweenness centrality and remove them so that communities are disconnected from each other.

1.1.4.2 Node Classification

In many applications, nodes are often characterized by contextual information. For instance, in a social network describing interactions between users, characteristics such as genre, interest, can be assigned to the nodes and thus, the network can be represented by an attributed graph. These labels typically appear on the user's profile within the network but frequently only part of them are available. When the labels of some nodes are known, but others are unknown, node classification or semi supervised label prediction consists in determining the unknown labels [Bhagat et al., 2011]. In the literature, two main approaches have been proposed to solve this task: on the one hand, methods which propagate the existing labels via random walks and on the other, methods based on traditional classifiers using graph information as features.

1.1.4.3 Link prediction

Another common task is link prediction. Consider a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ where \mathcal{V} is the set of nodes and \mathcal{E} is the set of observed edges. Link prediction aims at inferring new interactions among the nodes of the graph based on their attributes and observed edges [Liben-Nowell and Kleinberg, 2007] [Getoor and Diehl, 2005]. Attributes can be hand-crafted or the feature vector of a node in the case of an attributed graph. Initially, link prediction was based on different graph similarity measures such as Common Neighbors (CN) which is, for a pair of nodes (v_i, v_j) , the number of neighbors that they have in common.

$$CN(v_i, v_j) = |\mathcal{N}(v_i) \cap \mathcal{N}(v_j)| \quad (1.6)$$

The main issue of Common Neighbors is that it does not take into account the relative number of neighbors. Thus, the Jaccard measure (J) has been proposed to deal with it. For a pair of nodes (v_i, v_j) , it is defined as:

$$J(v_i, v_j) = \frac{|\mathcal{N}(v_i) \cap \mathcal{N}(v_j)|}{|\mathcal{N}(v_i) \cup \mathcal{N}(v_j)|} \quad (1.7)$$

In addition, many other heuristics such as Adamic-Adar, Katz index, or preferential attachment have been proposed to measure the proximity of two nodes in the graph [Liben-Nowell and Kleinberg, 2007]. Recently, some machine learning models, such as graph embeddings, have been proposed for link prediction [Trouillon et al., 2016] [Sinha et al., 2018].

To sum up, graphs can be visualized with tools such as the Fruchterman-Reingold layout. A lot of metrics and measures have been defined to quantify the importance of the nodes or to characterize the behavior of a graph. Moreover, the typical distributions in graphs follow heavy-tailed distributions and not the usual Gaussian ones. Finally, some quality measures such as modularity have been proposed to tackle the issue of community detection. This modularity can be maximized to find communities in a graph. While these metrics are very useful to study graphs, it is not efficient enough for solving new tasks such as node classification in comparison with more recent machine learning methods.

1.2 Machine learning for graph mining

Machine learning has brought two main kinds of models. First, graph embedding techniques consist of graph dimension reduction by describing each node (or edge or part of the graph) by a vector of the specified dimensions. The second type of method is graph neural networks (GNNs). They are deep neural networks on a non-homogeneous topology that aggregates information of the nodes. Both kinds of models use elements of deep learning. We will first discuss some key points of machine learning and deep neural networks to, then, study graph embeddings and graph neural networks.

1.2.1 Machine learning basics

Parts of this section are largely inspired by [Goodfellow et al., 2016].

1.2.1.1 The task

Machine learning allows tackling **tasks** that are too difficult to solve with fixed programs written by human beings. Learning is a mean of attaining the ability to perform the task. Usually, machine learning tasks are defined depending on how to process **an example**. An example is a collection of features that have been quantitatively measured from some objects. It is represented by a vector $\mathbf{x} \in \mathbb{R}^f$ where f is the number of features describing the example. There exists many kinds of machine learning tasks such as classification, prediction or anomaly

detection are some of the most common ones. For classification, the computer program is asked to which of k categories an example belongs to. To solve this, the algorithm produces a mapping function $\phi : \mathbb{R}^f \rightarrow \{1, \dots, k\}$. When $y = \phi(\mathbf{x})$, the model assigns the example described by the vector \mathbf{x} to the category identified by the label y .

1.2.1.2 The performance

To evaluate the ability of an algorithm to perform certain tasks, we must design quantitative measures of its performance. Usually, the measure of the performance depends on the task being carried out by the system. For tasks such as classification or transcription, we often measure the accuracy of the model. It is the proportion of examples for which the model produces correct output. Usually, we are interested in how well an algorithm performs on data that it has not seen before. Thus, the performances are evaluated on a **test set** that is separate from the data used for training the machine learning model, the **training set**.

1.2.1.3 The experience

Most machine learning algorithms can be categorized as either unsupervised or supervised. **Unsupervised** methods experience a dataset containing many features, then learn useful properties of the structure of this dataset. For example, clustering, which consists of dividing the dataset into clusters of similar examples, is an unsupervised learning task. Unsupervised learning involves observing several examples of a random vector \mathbf{x} and attempting to learn the probability distribution $p(\mathbf{x})$ from which the examples have been drawn. On the other hand, **supervised** learning algorithms experience a dataset containing many features but each example is associated with a **label**. For example, a flower dataset is annotated with the species of each plant. A supervised learning algorithm can learn to classify the plants based on their measurements. This time, the learning algorithm involves observing several examples of a random vector \mathbf{x} and an associated label value or label vector \mathbf{y} , then learning to predict \mathbf{y} from \mathbf{x} usually by estimating $p(\mathbf{y}|\mathbf{x})$. The third type of learning algorithm is semi-supervised algorithms. A **semi-supervised** learning algorithm experiences a dataset containing instances described by features but only some of them are associated with a label. Semi-supervision can be seen as weak supervision but can produce considerable improvement in accuracy.

1.2.1.4 Example: linear regression

Linear regression is, of course, a regression problem. The goal is to build a model that can take a vector $\mathbf{x} \in \mathbb{R}^f$ as input and predict the value of a scalar $y \in \mathbb{R}$ as its output. Let \hat{y} be the value that the model predicts y should take on. Then, the output of the model is:

$$\hat{y} = \mathbf{w}^T \mathbf{x} \tag{1.8}$$

where $\mathbf{w} \in \mathbb{R}^f$ is a vector of **parameters**.

Parameters are values that control the behavior of the model. Here, w_i is the coefficient that we multiply by x_i before summing up the contributions from all the features. \mathbf{w} can be seen as a set of weights that determine how each feature affects the final prediction. Suppose that we have n examples for training the model and m examples for evaluating the model. One way of measuring the performance of the model is to compute a **loss function**. For example, the mean squared error on the test set is given by:

$$MSE_{test} = \frac{1}{m} \sum_{i=1}^m (\hat{y}_i^{(test)} - y_i^{(test)})^2 \quad (1.9)$$

To improve the weights \mathbf{w} of our model in a way that reduces MSE_{test} , the algorithm is allowed gaining experience by observing examples from the training set $(\mathbf{x}^{(train)}, y^{(train)})$. One way to do it is to minimize the mean squared error. This is done by solving for where its gradient is $\mathbf{0}$.

A learning algorithm is trained with a training set and evaluated on a test set. The training set can be split into two parts. The first part will be used to train the learning algorithm. The second part is called the **validation set**. It is used to evaluate the performance of the algorithm during the training and for the tuning of the parameter of the algorithm.

1.2.1.5 Common evaluation metrics

Throughout this thesis, several evaluation metrics will be used. For an example $\mathbf{x} \in \mathbb{R}^f$ with label y , the model predicts a scalar $\hat{y} \in \mathbb{R}$. In the context of classification, when the true label of the instances and consequently the label predicted by the algorithm are categorical, taking k modalities, a confusion matrix can be computed to evaluate the algorithm. Each term $[l, l']$ of the matrix indicates the number of instances for which the true label equals l and the predicted label equals l' . An example with k set to 2 is given in Table 1.2. The number of examples predicted in the group 1 whose label is 1 is called the number of True Positives (TP). The examples predicted in the second group whose label is 2 are called the True Negative (TN). Those are the examples correctly classified by the learning algorithm. The examples predicted in the group 1 whose label is 2 are the False Positive (FP) and the examples predicted in the group 2 whose label is 1 are the False Negative (FN).

	$\hat{y} = 1$	$\hat{y} = 2$
$y = 1$	TP	FN
$y = 2$	FP	TN

Table 1.2: Example of a confusion matrix with two classes. TP: true positive, TN: true negative, FN: false negative, FP: false positive

From this matrix, many different measures can be defined to evaluate the performance of a learning algorithm. The accuracy is the proportion of examples predicted correctly.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1.10)$$

Precision and recall focus on a specific class. The precision for the class 1 is:

$$precision = \frac{TP}{TP + FP} \quad (1.11)$$

and the recall is:

$$recall = \frac{TP}{TP + FN} \quad (1.12)$$

Finally, the F_1 -measure is defined by:

$$F_1 = 2 \frac{precision \times recall}{precision + recall} \quad (1.13)$$

To sum up, we have seen that a learning algorithm performs a task. It experiences a dataset in an unsupervised, semi-supervised or supervised manner. The training process is done via optimizing a loss function on a set of training examples. Finally, the model's performance is evaluated with metrics such as the accuracy, or the F_1 -measure. There is a lot of learning algorithms. Nowadays, deep neural networks have proven to be very efficient in several tasks and are state-of-the-art models.

1.2.2 Deep neural networks

Deep learning is a machine learning approach based on artificial neural networks. These artificial neural networks are computing systems inspired by the functioning of the human brain.

For example, let us have a look at the simplest neural network: the perceptron [Rosenblatt, 1958]. This network only has one hidden layer as in Figure 1.5. For an input vector $\mathbf{x} \in \mathbb{R}^f$, it is defined as:

$$h(\mathbf{x}) = g(W \cdot \mathbf{x} + \mathbf{b}) \quad (1.14)$$

where g is an activation function and the matrix W and the bias b are weights to learn. Weights can be shared among neurons.

Definition 1.2.1 (Artificial neural network). *An artificial neural network consists of a set of neurons and a set of connections between them. Each neuron stores a value and the connections indicate whether a value is passed to another neuron or not. The weights of the network are the weights of the connection. The weight of a connection quantifies the importance of the source in the computation of the target of this connection.*

Definition 1.2.2 (Deep neural network). *A deep neural network is an artificial neural network made of multiple layers. The first layer is called the input layer, the final layer is the output layer and the other layers are called the hidden layers. A layer can be a perceptron or any more complex function of some parameters.*

There are three steps to create and use a deep neural network. First, designing his architecture is a key element to make it efficient. Secondly, it must be trained so it can, in a third place, infer predictions. Building the network architecture consists of arranging the neurons and their connections into layers. The input is usually a vector and the input layer must fit

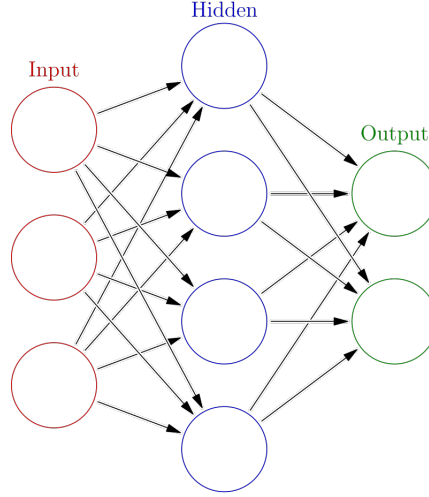


Figure 1.5: An example of an artificial neural network. Source: Wikipedia

the dimension of the input vectors. Thus, the design of this first layer is often very limited. Then, there is a wide variety of potential hidden layers. The most common ones are convolution layers [Goodfellow et al., 2016] in computer vision and attention layers [Vaswani et al., 2017] in natural language processing. The final layer often depends on the kind of loss function that is used to train the model.

Definition 1.2.3 (Loss function). *A loss function also called cost function or error function, is a function that takes as input some vectors (the representations of some data) Y_{pred} and their expected values (the labels of the data) Y_{true} to compare them. The less similar these two vectors are, the higher the cost.*

Example 1.2.1. *In the context of classification into k groups, a typical loss function is the standard categorical cross-entropy loss that takes as input a set of vectors of predictions Y_{pred} and their expected values Y_{true} . It is defined as:*

$$L(Y^{true}, Y^{pred}) = - \sum_{j=1}^k \sum_{i=0}^{n-1} (y_{ij}^{true} \times \log(y_{ij}^{pred})) \quad (1.15)$$

where k is the number of different classes, and y_{ij}^{true} and y_{ij}^{pred} are assumed to be positive.

Once built, the network must be trained. The forward pass is the process of passing the data through the network. It first goes in the input layer and gets transformed as it progresses to the output layer. The output layer gives a new vectorial representation of an input that is compared to its expected label thanks to the loss function. Then, there is the backpropagation step. Backpropagation is a widely used algorithm to train neural networks. It computes the gradient of the loss function with respect to the weights of the network for a specific input-output instance with the use of the chain rule. Thanks to gradient descent, it is possible to minimize the loss function by changing the weights of the layers. For more mathematical information, we refer the reader to [Goodfellow et al., 2016].

The final step is the inference. The training step computed the weights of the network. Then, when new data come in, it is fed to the network and the network outputs a new representation. In a classification problem, this new representation will be the class to whose the new data belongs.

To sum up, a neural network is made of layers of neurons that are connected together. It is trained thanks to a training set and a loss function that computes the error between the output of the network and the expected representation. Through backpropagation, the weights of the network are adjusted. Finally, it is possible to infer some predictions on new data.

1.2.3 Graph embeddings

Definition 1.2.4 (Graph embedding). *Let $\mathcal{G}(\mathcal{V}, \mathcal{E})$ be an unweighted and undirected graph where A is its binary adjacency matrix. Graph embedding consists of encoding the graph into a low-dimensional space $\mathbb{R}^{f'}$ with a function $\phi: \mathcal{V} \mapsto \mathcal{Z}$ which maps vertices to vector embeddings while preserving some properties of the graph. The embedding matrix is denoted Z where each row \mathbf{z}_i is the embedding vector of the node v_i .*

Example 1.2.2. *For a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, a trivial embedding is the adjacency matrix itself. Each node embedding is its corresponding row in the adjacency matrix. Thus, every node lies in a n -dimensional space, and pairs of nodes whose set of neighbors are the same are close together. Note that this embedding is not in low-dimensions.*

These techniques are deeply related to the problem of graph dimension reduction. According to the typology of [Ou et al., 2017], graph embeddings methods can be divided into three sub-categories: random-walks-based methods, matrix factorization methods, and deep learning methods.

1.2.3.1 Random-walks based methods

The first type of method makes use of the Skip-Gram [Mikolov et al., 2013] algorithm. The aim of the Skip-Gram model is to learn a shallow neural network with only a few layers which is called the embedding layer. It was first made for natural language processing to compute embeddings of words. Each word of the corpus is encoded as a one-hot vector whose dimension is the number n of words in the corpus. A one-hot vector will be used as input of the neural network. The network is made of an input layer followed by a linear layer, the embedding layer, where a linear transformation $W_{input} \in \mathbb{R}^{n \times f'}$ is learned with f' is the hidden dimension. Then, there is the output layer made of a linear transformation $W_{output} \in \mathbb{R}^{n \times f'}$ and a softmax. After the neural network has been trained, the embedding of the words can be read on the embedding layer. First, for a specific word w , find its context $C(w)$. Usually, it is a few words before and a few words after w in the sentence. Then, compute the product between the one-hot vector $\mathbf{w} \in \mathbb{R}^n$ representing w and the embedding layer:

$$\mathbf{h} = W_{input}^T \mathbf{w} \quad (1.16)$$

The next step consists of computing the dot-product with the output layer and applying a softmax:

$$\mathbf{y}_{pred} = \text{softmax}(W_{output}^T \mathbf{h}) \quad (1.17)$$

where the *softmax* of a vector $\mathbf{x} \in \mathbb{R}^v$ is defined as:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^v e^{x_j}} \quad (1.18)$$

For every pair of word (w, w_C) where $w_C \in C(w)$, the loss function is the difference between the representation \mathbf{y}_{pred} of w after the *softmax* and the one-hot vector \mathbf{y}_{true} representing w_C . The network is trained using stochastic gradient descent.

Example 1.2.3. *For example, assume we want to study the phrase "The man who passes the sentence should swing the sword" and focus on embedding the word "passes" as in Figure 1.6. The sentence has 8 words and they will be embedded in 3 dimensions.*

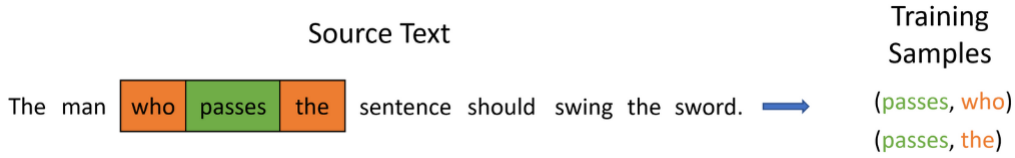


Figure 1.6: Training window of the word "passes". The context is made of "who" and "the".
 Source: aegis4048.github.io

Then, as in Figure 1.7, the word "passes" is encoded into a one-hot vector $\mathbf{w} \in \mathbb{R}^8$ in the input layer. It is fed to the embedding layer $W_{input} \in \mathbb{R}^{8 \times 3}$ and the resulting vector $\mathbf{h} = W_{input}^T \mathbf{w}$ is the embedding of the word "passes". Finally, it is multiplied by W_{output} and applied a *softmax*. The output \mathbf{y}_{pred} can be then compared with the one-hot vector representing the words "the" and "who" of the context.

The Skip-Gram model can also be used with graphs to learn an embedding of the nodes. Instead of using sentences (sequence of words) to define a context, it uses sequences of nodes generated by random-walks in the graph. The shallow neural network remains the same and every node, instead of word, is encoded by a one-hot vector. For an input node, its context can be defined by random-walks. A random-walk in a graph is a path where, at each step, the next node is chosen at random among the neighbors of the current node. The basic idea is to start on a specific node, then, go to one of its neighbors at random. Random-walks based approaches [Grover and Leskovec, 2016] [Perozzi et al., 2014a] rely on the Skip-Gram model discussed previously. The main difference between these methods is the way the context of every node is generated and some optimization tricks. For DeepWalk [Perozzi et al., 2014a], the context of a node is directly drawn thanks to a standard random-walk. The standard random-walk in a graph starts at a specific node v_i . The next state of the walk is entirely determined by the current state (its position at v_i). That is, the probability of the walk going to a neighbor of v_i

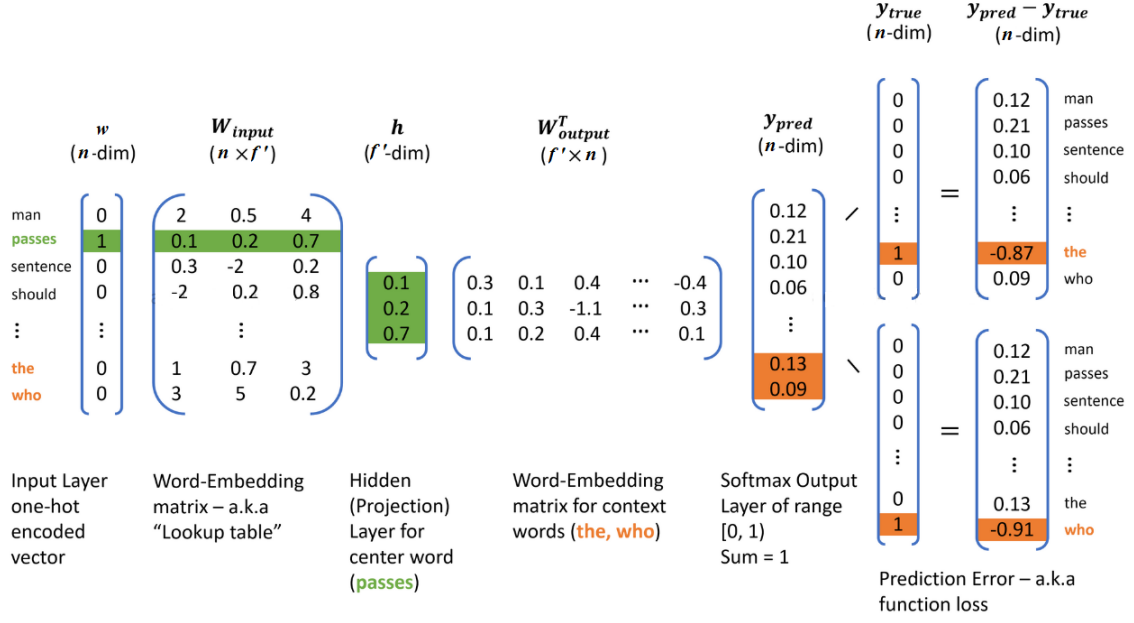


Figure 1.7: Skip-gram model structure. Current word is "passes". The context is made of "who" and "the". Source: aegis4048.github.io

is $1/d_i$ where d_i is the degree of v_i . The context of a node v_i is the set of the few previous and the few next nodes in the walk. With words, training samples were pairs made of the current word and a word of the context. With nodes, training samples are pairs of nodes made of the current node v_i and a node of the context. For their part, the authors of Node2vec [Grover and Leskovec, 2016] propose to use biased random-walks to generate the context of the nodes. The process to get the next step depends on the current state of the walk and the previous one. Assume the previous state was v_i and the walk is now on the node v_j . Then, as in Figure 1.8, the unnormalized probability of the walk to go back to v_i is $1/p$, the probability to go to a neighbor of both v_i and v_j is 1 and the probability to go to a neighbor of v_j but not a neighbor of v_i is $1/q$. p and q are hyper-parameters to tune the probability of the walk in two different ways to go far in the graph, Depth First Search (DFS), or to stay close to the starting point, Breadth First Search (BFS). These different types of random-walks allow capturing different properties of the graph.

As for DeepWalk, random-walks are used to create the context around a specific node to compute the embedding thanks to the Skip-Gram model. Another difference between DeepWalk and Node2vec is the way they optimize the loss function. Node2vec uses negative sampling while Deepwalk uses hierarchical softmax. Both methods have time complexity $O(nf')$ which can be a lot faster than matrix decomposition methods in the case of dense graphs.

Note that the embedding of an entire graph also exists but will not be discussed here. Thus, graph embedding and node embedding are interchangeable through this thesis. Also, the embedding itself can be either the mapping function or the output of the function: a low-dimension vector for each node of the graph.

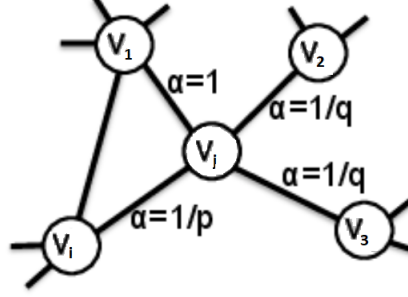


Figure 1.8: Biased random-walks generation process for Node2vec. Source: [Grover and Leskovec, 2016]

1.2.3.2 Matrix decomposition methods

The second family of methods is based on matrix decomposition. The main idea is to work directly on the adjacency matrix. As said previously, the most straightforward embedding is the matrix itself as it provides a natural approach without loss of information but it does not reduce the dimension of the problem. Thus, downstream applications will not be faster. It is possible to apply the wide range of matrix reduction techniques to the adjacency matrix to operate a graph embedding. Of course, the choice of decomposition has an impact on the quality of the embedding itself. Matrix factorization methods are, most of the time, based on the decomposition of the adjacency matrix and thus do not take into account node features [Belkin and Niyogi, 2003] [Ou et al., 2016] [Roweis and Saul, 2000].

In this family, we can mention Laplacian Eigenmaps, LE [Belkin and Niyogi, 2003], an algorithm that provides a computationally efficient approach to non-linear dimensionality reduction. For a weighted graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, W)$, the method aims to minimize the following cost function:

$$L(Z) = \frac{1}{2} \sum_{v_i \in \mathcal{V}, v_j \in \mathcal{V}} \|\mathbf{z}_i - \mathbf{z}_j\|^2 w_{ij} = \text{tr}(Z^T \hat{L} Z) \quad (1.19)$$

where $\text{tr}(\cdot)$ is the trace of a matrix and $\hat{L} = D - W$ is the Laplacian of the graph with $d_{ii} = \sum_{v_j \in \mathcal{V}} w_{ji}$ the diagonal weight matrix. This minimization is done with spectral analysis. In fact, solving this problem is the same as solving the generalized eigenvector problem:

$$\hat{L}\mathbf{z} = \lambda D\mathbf{z} \quad (1.20)$$

Let $\mathbf{y}_0, \mathbf{y}_1, \dots$ be the solutions of the generalized eigenvector problem ordered according to their eigenvalues by increasing order. Then, the embedding of a node v_i is given by $(y_0(i), y_1(i), \dots)$. This method only requires to solve an eigenvalue decomposition that can be quite fast depending on the number of edges with the total time complexity of the algorithm is $O(|E|f'^2)$.

For their part, [Roweis and Saul, 2000] presented LLE for Locally Linear Embedding. LLE is an unsupervised learning algorithm that computes neighborhood-preserving embeddings. For

a weighted graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, W)$, the authors assume that the weight w_{ij} represents the weight of node v_i in the representation of v_j :

$$\hat{\mathbf{z}}_i \approx \sum_{v_j \in \mathcal{V}} w_{ij} \mathbf{z}_j \quad (1.21)$$

Hence, the embedding is obtained by minimizing the following loss function:

$$L(Z) = \sum_{v_i \in \mathcal{V}} \|\mathbf{z}_i - \hat{\mathbf{z}}_i\|^2 \quad (1.22)$$

This is a quadratic cost function in terms of the coordinates \mathbf{z}_i with fixed weights w_{ij} . Let us denote I the identity matrix. Then, the loss function can be minimized by finding the eigenvectors of the matrix $(I - W)^T(I - W)$ whose bottom f' non zero eigenvectors provide the embedding of the nodes. This method is also quite efficient to compute an embedding of a graph since its complexity is also $O(|E|f'^2)$.

A more recent approach is HOPE which stands for High-Order Proximity preserving Embedding [Ou et al., 2016]. This embedding method aims to preserve high-order proximity measurements in terms of powers of the adjacency matrix A . They define a similarity matrix S such that s_{ij} is the similarity between v_i and v_j . They want to minimize the norm between this similarity matrix and the embedding matrix:

$$L(Z, S) = \|S - ZZ^T\|_2^2 \quad (1.23)$$

The authors propose different similarity matrices of the form $S = M_1^{-1}M_2$. For example, for Katz Index, $M_1 = I - \beta A$ and $M_2 = \beta A$ and for common neighbors, we have $M_1 = I$ and $M_2 = A^2$. In a similar fashion as LLE, finding the optimal approximation of the proximity matrix S is the same as performing a Singular Value Decomposition [Golub and Reinsch, 1970] on S and using the largest singular vectors to construct the embedding. This also can be done with truncated singular value decomposition to drastically reduce the complexity and bring it in line with LLE and LE so that the total complexity of the decomposition is also $O(|E|f'^2)$.

Matrix decomposition approaches are rather fast methods that allow a precise proximity to be preserved in the embedding space. Their major drawback is that they do not generalize well to unseen data. The second limit is the fact that they do not take into account the features of the nodes. Also, some methods have been proposed to tackle these issues.

1.2.3.3 Deep learning methods

Deep learning methods [Wang et al., 2016] [Cao et al., 2016] can also be used to encode a graph into a low-dimension space. These methods often rely on auto-encoder architecture to learn a mapping of the nodes into a low-dimensional space. An auto-encoder is made of an encoder that takes as input the graph, the nodes, and/or the features of the nodes to transform it into low-dimension vectors. Then, a decoder takes as input those vectors and tries to recompose

the input of the encoder. A standard method is Structural Deep Network Embedding, SDNE [Wang et al., 2016]. The goal of this method is to compute the embedding of each node of the graph while preserving the first and second-order proximities. The first-order proximity describes the pairwise proximity between vertices. For a node v_i , it can be described by the set of its neighbors $\mathcal{N}(v_i)$. Thus, the embedding aims to project near together in the embedding space linked nodes. Then, the second-order proximity between two nodes v_i and v_j describes the proximity of the pair's neighborhood structure. It is determined by the similarity between $\mathcal{N}(v_i)$ and $\mathcal{N}(v_j)$. Thus, nodes having the same neighbors should be near together in the embedding space. More technically, SDNE's input layer is $\mathbf{h}_i^0 = \mathbf{a}_i$. Initially, each node is described by its row in the adjacency matrix. Then, SDNE's encoder is a multi-layer perceptron where each layer c follows this equation:

$$\mathbf{h}_i^c = \sigma(W^c \mathbf{h}_i^{c-1} + b^c) \quad (1.24)$$

where W^c and b^c are parameters to learn. The final layer of the encoder outputs $\mathbf{enc}(v_i) = \mathbf{h}_i^C$ where C is the number of the last layer. Then, the first order proximity is preserved thanks to the use of a first loss function on the output of the encoder:

$$L_{1st} = \sum_{(v_i, v_j) \in \mathcal{E}} a_{i,j} \|\mathbf{enc}(v_i) - \mathbf{enc}(v_j)\|^2 \quad (1.25)$$

For any pair of linked nodes, this first loss function aims at keeping them close by in the embedding space by ensuring that their euclidean distance is minimized. After computing the embeddings $\mathbf{enc}(v_i)$ for each node, the decoder will try to reconstruct the input of the network: the rows of the adjacency matrix. Indeed, the authors proposed to build the decoder as another multi-layer perceptron whose output is $\hat{\mathbf{a}}_i = \mathbf{dec}(\mathbf{enc}(v_i))$. Then, the second-order proximity is preserved thanks to the use of a second loss function:

$$L_{2nd} = \sum_{v_i \in \mathcal{V}} \|(\mathbf{dec}(\mathbf{enc}(v_i)) - \mathbf{a}_i) \odot \beta \mathbf{a}_i\| \quad (1.26)$$

where $\beta \in \mathbb{R}$ is a parameter and \odot is the Hadamard product. The aim of this second loss function is that the output of the network must look like the input which is the rows of the adjacency matrix. The row \mathbf{a}_i of this matrix A is equivalent to the list of the neighbors $\mathcal{N}(v_i)$ for the node v_i since they contain the same information. Thus, reconstructing the rows of the adjacency matrix means preserving the second-order proximity as two nodes that have the same set of neighbors in the graph should now have the same nearest neighbors in the embedding space.

But, one major drawback of these methods is that they only consider node structural information *i.e.* the connectivity between pairs of nodes. They ignore the fact that graph can contain feature information that describes its nodes. Therefore, [Kipf and Welling, 2016] proposed the Graph Auto-Encoder, GAE, that can deal with attributed networks. This is a relatively simple architecture. For an attributed graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, X)$, the encoder consists of two GCN layers (defined in Section 1.2.4:

$$Z = \mathbf{enc}(X) = \mathbf{GCN}(\sigma(\mathbf{GCN}(X|W^1))|W^2) \quad (1.27)$$

where σ is the ReLU function and W^1 and W^2 are parameters to learn. The output of the encoder is the embedding matrix Z where each row \mathbf{z}_i is the embedding vector of a node v_i . As for SDNE, GAE tries to reconstruct the adjacency matrix A with the decoder:

$$\hat{a}_{ij} = \text{dec}(\mathbf{z}_i, \mathbf{z}_j) = \sigma'(\mathbf{z}_i^T \mathbf{z}_j) \quad (1.28)$$

where σ' is the logistic sigmoid function. To compare the real adjacency matrix A and the reconstructed one \hat{A} , the authors propose to use a cross-entropy loss.

Name of the method	Graph sim.	Embedding sim.
Laplacian Eigenmaps (LE) - $O(N^2)$	1st-order prox.	Euclidean
Locally Linear Emb. (LLE) - $O(N^2)$	1st-order prox.	Euclidean
HOPE - $O(N^2)$	Katz-Index	Dot-product
SVD of the adjacency matrix - $O(N^2)$	2nd-order prox.	Dot-product
Struc2vec (S2V) - $O(N \log(N))$	co-occurrence proba.	Dot-product
Node2vec (N2V) - $O(N)$	co-occurrence proba.	Dot-product
Verse - $O(N)$	Perso. Page-Rank	Dot-product
Kamada-Kawai layout (KKL) - $O(N^2)$		Euclidean
Multi-dim Scaling (MDS)	1st-order prox.	Euclidean
SDNE - $O(N)$	1st & 2nd-order prox.	Euclidean

Table 1.3: Studied methods with complexity, their graph similarity (encoder) and their distance in the embedding space (decoder)

To sum up, there is a wide variety of graph embedding techniques that transform the nodes of a graph into low-dimension vectors as presented in Table 1.3. LE [Belkin and Niyogi, 2003], LLE [Roweis and Saul, 2000], HOPE [Ou et al., 2016], Struc2vec [Ribeiro et al., 2017], Node2vec [Grover and Leskovec, 2016], Verse [Tsitsulin et al., 2018], Kamada-Kawai layout [Kamada and Kawai, 1989], MDS [Kruskall, 1964] and SDNE [Wang et al., 2016] are all embedding techniques that can be classified into either matrix decomposition methods or Skip-Gram methods or deep neural network methods. The embeddings produced by these methods can be very different since they aim at preserving different aspects of the graph. These embeddings can then be used for any downstream task that requires vectors to perform inference. Most importantly, it should be noticed that, in an anomaly detection setup, as graph embedding allows the user to transform a graph into a set of vectors, usual detection methods suited for vectorial data can then be applied.

1.2.4 Graph neural networks

Deep neural networks have proven to be very efficient in both computer vision and natural language processing. Relational data can also be used with neural networks to perform inference

but the model should take into account the specificity of this type of data. Here, we detail the process of creating a neural network on a graph.

1.2.4.1 Building a graph neural network

The main idea of deep neural networks on graphs, or graph neural networks, is to aggregate the information of the nodes to infer a property. Let $\mathcal{G}(\mathcal{V}, \mathcal{E}, X)$ be a graph on a set of nodes \mathcal{V} , a set of edges \mathcal{E} and a feature matrix X where each row $\mathbf{x}_i \in \mathbb{R}^f$ is the feature vector of the node $v_i \in \mathcal{V}$. We want to learn a new representation of this node by aggregating the information of surrounding nodes, usually its neighbors. The input features of a neural network layer are called \mathbf{h}_i for the node v_i and its output of the layer is \mathbf{h}'_i for the same node. The information flow from its neighbors to the studied node v_i is a message as in Figure 1.9. Thus, this aggregation is called message passing.

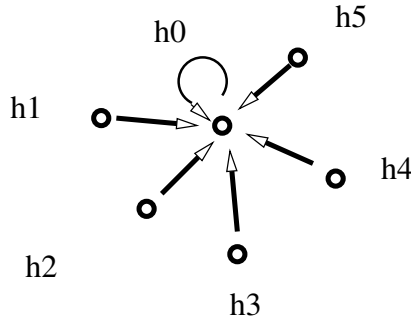


Figure 1.9: Message passing: aggregation of the information of surrounding nodes.

This new representation \mathbf{h}'_i of the node v_i is given by the definition ψ of the layer that we use to compute: $\mathbf{h}'_i = \psi(\{\mathbf{h}_j | v_j \in \mathcal{N}(v_i)\})$. A neural network with only one layer is a relatively small network. To build a bigger one, multiple layers can be placed on top of each other. Bigger neural networks are more complex, usually less understandable, and may or may not be more precise depending on the situation.

In image processing, many convolution layers are added to a deep neural network for computer vision purposes. At the pixel scale, a single convolution in an image provides a summary of the vicinity of this pixel. A second convolution will provide information about the pixels that are a bit further from this pixel in the image. Adding more and more convolutions allows aggregating the information with pixels that are still further. In natural language processing, attention mechanisms are used. Attention is another aggregation method that can be split into different heads or parallel computations. Each head is said to focus on different parts of the sentence to extract enough information for later inference. In the same manner as previously, many attention layers can be added in a single neural network such that the information necessary to do inference on a word is brought to it even if it is far in the corpus. The more attention layers, the more information is brought to inference.

The same process applies to graphs. As said before, a neural layer aggregates the local information around a node v_i . For the rest of this thesis, we will assume that the aggregation is

made on the set of the neighbors $\mathcal{N}(v_i)$ of this node. Moreover, the information of the node v_i itself may be added to this aggregation. We call it a self-loop since the features of a node will be used to predict new features of the same node. One layer of a graph neural network explores the neighborhood of a node, the 1-hop neighborhood more precisely. If we add a second layer, it will aggregate information from the 2-hop neighborhood and so on. The more layers, the further in the graph the information is looked at to process the inference. This is called the field-of-view which is addressed in Figure 1.10. It is clearly defined in the case of graph neural networks since a network with C layers aggregates information from the C -hop neighborhood of a node. The set of nodes on which the aggregation is made and the set of edges between them is the computation graph. In the case of graph mining, the field-of-view and the computation graph are deeply related.

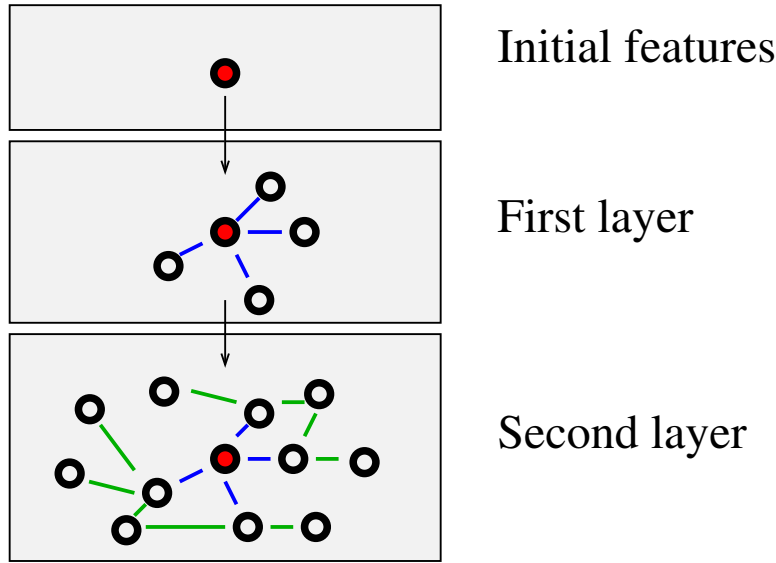


Figure 1.10: The field-of-view increases as the number of layers increases. With 0 layer, only the features of the nodes are used. The first layer aggregates information from the direct neighbors of the red node. The second layer aggregates information from the 2-hop neighbors of the red node.

Recently, some graph neural networks (GNN) methods have been proposed to tackle the issue of attributed graphs [Kipf and Welling, 2017] [Hamilton et al., 2017] [Veličković et al., 2018]. They define neural networks layers with graph convolution, attention mechanism, or different aggregation methods. They have shown a significant improvement in terms of node classification. These methods focus on attributed networks in a semi-supervised setting: some of the nodes have a label that corresponds to the group they belong to. The usually studied task is node classification which consists of assigning the remaining nodes into the right group.

Graph Convolutional Network, GCN [Kipf and Welling, 2017], is one of the first models to tackle the issue of semi-supervised attributed graph classification. They are an extension of the usual convolutional layers in computer vision [LeCun and Bengio, 1998]. For an attributed graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, X)$, let the input layer be defined as $\mathbf{h}_i^0 = \mathbf{x}_i$, then a graph convolutional layer is

given by:

$$\mathbf{h}_i^c = \sigma(W^c \cdot \sum_{v_j \in \mathcal{N}(v_i)} \mathbf{h}_j^{c-1} / \sqrt{d_i d_j}) \quad (1.29)$$

where σ is an activation function, W^c is the trainable matrix at layer c and d_i is the degree of the node v_i . Thus, each convolutional layer is a linear combination of the output of the previous one with weights depending on the degree of the nodes. Then, to train the parameters of the layers, the authors proposed to use a cross-entropy loss.

GraphSage is another method that has been proposed to deal with graphs in a deep learning fashion [Hamilton et al., 2017]. It is an inductive framework that leverages node feature information to generate node embeddings. The workflow of the algorithm is described in Figure 1.11. First, some nodes from the neighborhood are sampled. Then, the features of these nodes are aggregated together. Finally, this aggregated information is used for later inference.

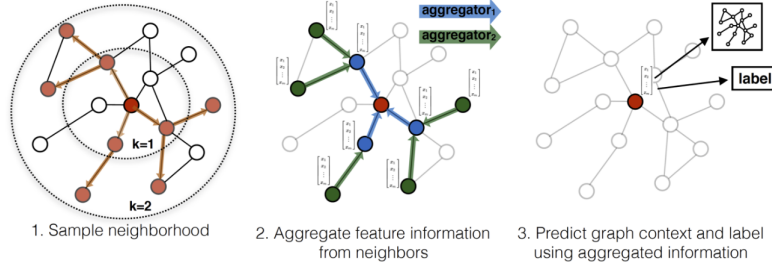


Figure 1.11: Illustration of the GraphSage framework - [Hamilton et al., 2017]

More precisely, the information of the sampled nodes is aggregated thanks to a deep neural network architecture. The input of the network is the adjacency matrix A and the features X of the node v_i with $\mathbf{h}_i^0 = \mathbf{x}_i$. Then, each layer of the network is defined as:

$$\mathbf{h}_i^c = \sigma(W^c \cdot AGGREGATE^c(\mathbf{h}_j^{c-1}, \forall v_j \in \mathcal{S}(v_i))) \quad (1.30)$$

where c is the number of the layer, W_c is the weight to learn for this layer, $AGGREGATE^c$ is an aggregation function discussed next and $\mathcal{S}(v_i)$ is the set of sampled nodes in the vicinity of the node v_i . The authors proposed many aggregation functions. The first example is the mean aggregator. It consists simply of taking the mean of the features of the sampled nodes. Another approach is the Long Short Term Memory, LSTM [Hochreiter and Schmidhuber, 1997], aggregator which has the advantage of the larger expressive capability. The last aggregator is a max pooling approach. Each sampled vector is fed through a fully-connected layer where the operation is defined as:

$$AGGREGATE_{pool}^c(\mathbf{h}_j^{c-1}) = \max_{v_j \in \mathcal{S}(v_i)} \sigma(W_{pool}^c \mathbf{h}_j^{c-1} + b^c) \quad (1.31)$$

where W_{pool}^c and b^c are trainable parameters. The most used and simplest aggregation function is the mean aggregator.

The last layer of the network outputs the embedding \mathbf{z}_i of each node v_i of the graph.

Then, the next step is the learning of the parameters. To do so, the authors defined a custom loss function as:

$$L(\mathbf{z}_i) = -\log(\text{dec}(\mathbf{z}_i, \mathbf{z}_j)) - Q \mathbb{E}_{v_n \sim P_n(v_i)} \log(-\text{dec}(\mathbf{z}_i, \mathbf{z}_n)) \quad (1.32)$$

where the decoding function is $\text{dec}(\mathbf{z}_i, \mathbf{z}_j) = \sigma(\mathbf{z}_i^T \mathbf{z}_j)$ and σ is the sigmoid function, Q is the number of negative samples, P_n is a negative sampling distribution, and v_j is a node that co-occurs in a random-walk around v_i . The parameters of the network are learned with stochastic gradient descent.

One of the best graph neural layers, in terms of performance, is the attention layer that is used to build Graph Attention Networks, GAT [Veličković et al., 2018]. It also deals with attributed graphs and, thus, takes as input the adjacency matrix A and the feature matrix X . GAT aggregates information from the neighborhood of a node by assigning more importance to some nodes. Let the input layer be defined by $\mathbf{h}_i^0 = \mathbf{x}_i$, then an attention layer is given by:

$$\mathbf{h}_i^c = \sigma(W^c \cdot \sum_{v_j \in \mathcal{N}(v_i)} \alpha_{ij}^c \mathbf{h}_j^{c-1}) \quad (1.33)$$

where σ is an activation function, W^c is learned and α_{ij}^c are parameters to weight the aggregation of the information defined as:

$$\alpha_{ij}^c = \text{softmax}(e_{ij}^c) \quad (1.34)$$

with

$$e_{ij}^c = \text{LeakyReLU}(\mathbf{a}_c^T [W^c \mathbf{h}_i^c || W^c \mathbf{h}_j^c]) \quad (1.35)$$

where \mathbf{a}_c is a trainable vector. In the end, a GAT layer learns the weights of a weighted linear combination of the features of the nodes by minimizing the cross-entropy loss. The key difference between the three methods is that GraphSage does not necessarily assign any weight to the different nodes in the aggregation process while GCN assigns a non-parametric weight $\frac{1}{\sqrt{d_i d_j}}$ to the message passing from v_j to v_i and GAT learns these weights such that the most important nodes have a larger weight. [Veličković et al., 2018] conducted experiments to evaluate the performances of the different graph neural network approaches. First, in a transductive setting, they learn to classify the nodes of three graphs. Cora has 2708 nodes divided into 7 classes, Citeseer has 3327 nodes in 6 groups and PubMed has 19717 nodes in 3 different classes. They train the different methods on 140, 120 and 60 nodes respectively with 500 nodes to validate and 1000 nodes to test. The results in terms of accuracy of the classification are presented in the table 1.4

The second type of experiment uses an inductive setting on PPI. The dataset PPI is composed of 24 graphs on 56944 nodes divided into 121 classes. The training is done on 20 graphs with 44906 nodes. The validation is made with 2 graphs on 6514 nodes and the test set consists of 2 graphs on 5524 nodes. The results in terms of micro-averaged F_1 scores are given in table 1.5.

Method / Graph	Cora	Citeseer	Pubmed
GAT	83.0 ± 0.7	72.5 ± 0.7	79.0 ± 0.3
GCN	81.4 ± 0.5	70.9 ± 0.5	79.0 ± 0.3

Table 1.4: Transductive setting - Results from [Veličković et al., 2018]

Method / Graph	PPI
GAT	0.768
GraphSage	0.973 ± 0.002

Table 1.5: Inductive setting - Results from [Veličković et al., 2018]

These results show that the GAT architecture seems to be the best for node classification. Moreover, the method is efficient and provides good results. Thus, GNNs seem a good opportunity to deal with attributed graphs.

To conclude, we have seen that there are many ways to do machine learning on graphs. First, through graph embeddings, the nodes of a graph can be represented in a low-dimensional space. The different techniques consist of using the Skip-Gram model, matrix factorization, or deep neural networks. Secondly, graph neural networks are a very powerful tool to deal with attributed graphs. They are particularly efficient for solving node classification. But, a specific case of node classification is anomaly detection where there are only two classes: the normal nodes and the anomalies. So, a promising path to explore seems to be graph-based anomaly detection using graph neural networks. That is the approach that we retained to design CoBaGAD presented in Chapter 2.

1.3 Anomaly detection

Anomaly detection consists of finding rare instances or patterns in the data that are abnormal. This definition does not depend on the type of data that is studied. In fact, there are anomalies in signal processing, computer vision, natural language processing, regular vectorial data, or even graphs. In the following section, we will focus on the detection of anomalies both in vectorial data and in graphs.

In regular tabular data, the goal is to find instances of the dataset \mathcal{D} that are far from the majority. These instances can be single points or small groups of points. The main idea to find instances that are far from the rest of the dataset is to compute a certain distance with respect to the other points. A first method consists of computing the minimum distance, Min_dist_i , of each data point i to any other data point j . Then, we can find those instances whose minimum distance is the highest.

$$Min_dist_i = \min_{j \in \mathcal{D}} D(i, j) \quad (1.36)$$

where D is a distance function and the minimum over j is over all instances in the dataset

except i . Such a method can be useful as a first try to look at abnormal data but it also has drawbacks. For example, in the unitary hypercube in dimension d , we uniformly draw 5000 samples. Then, we can compute the pairwise euclidean distance between any pair of data points and normalize it by the maximum distance which is \sqrt{d} . Finally, we can draw the density plot of these distances for any dimensions d .

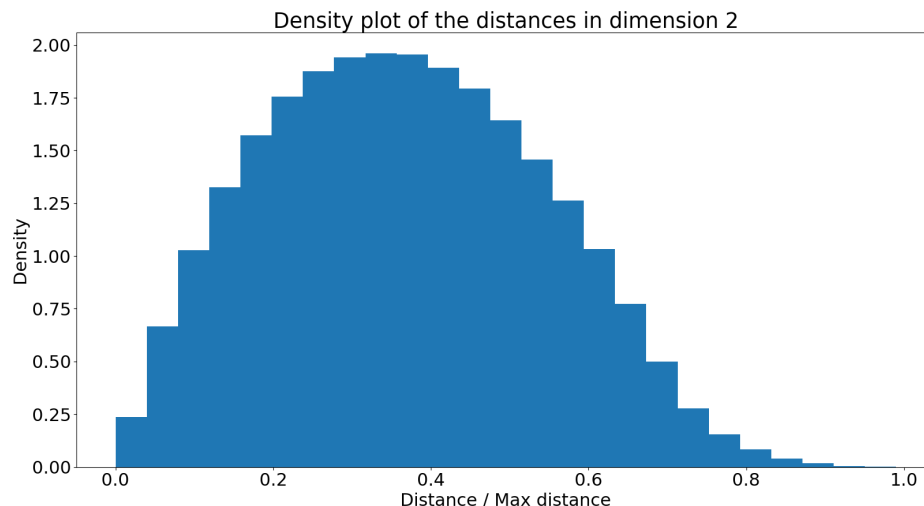


Figure 1.12: Density plot of the distances in two dimensions.

For $d = 2$, Figure 1.12 shows that the probability of frequency of occurrence of the normalized distance is widespread from 0.0 to 1.0. This implies that distances can take a lot of different values and it is thus possible to differentiate them and, subsequently, differentiate the data points to find anomalies. Let's see what happens in higher dimension.

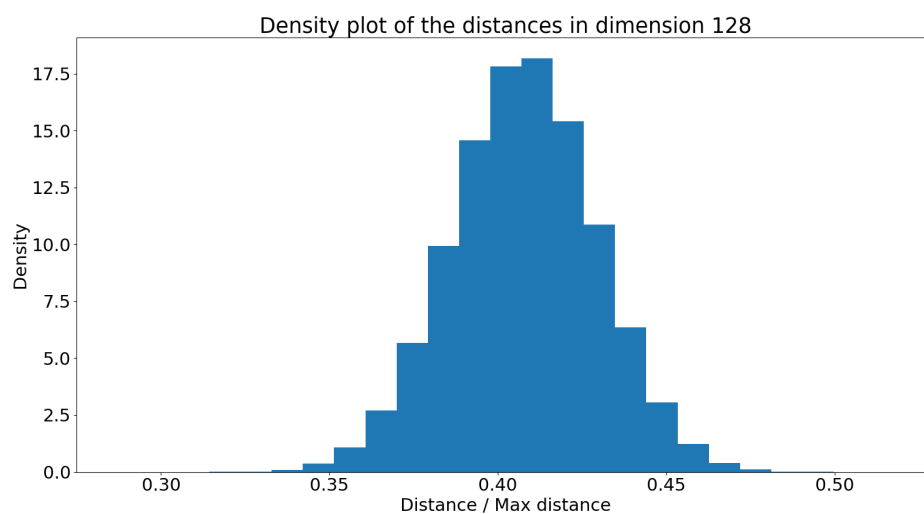


Figure 1.13: Density plot of the distances in 128 dimensions.

For $d = 128$, Figure 1.13 shows that the distances concentrates around a central value. This

means that there is less discrepancy between the values and, thus, it is harder to distinguish between the data points.

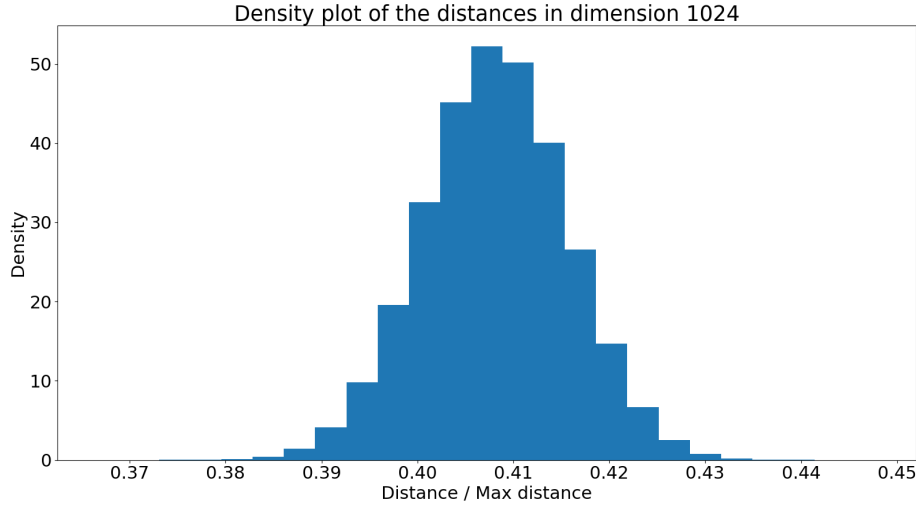


Figure 1.14: Density plot of the distances in 1024 dimensions.

For $d = 1024$, as shown in Figure 1.14, the notion of euclidean distance vanishes as the pairwise distance between any pair of instances is nearly the same. This is the effect of the so-called curse of dimensionality [Bellman, 1958]. The distances concentrate around a value. In anomaly detection, this phenomenon has a huge impact on the ability to detect anomalies. Indeed, it makes it difficult to find instances of the data that are "far" from the other points, since, in high dimension, euclidean distances are all the same. Instances cannot be far anymore since they are all as far from each other.

To solve the issue of the curse of dimensionality, many methods have been proposed. It is possible to change the kind of distance, switch to density based approaches, use deep neural networks or add a topology. Topology is very useful as it provides a natural way to split the space into sub-regions that can allow finding anomalies. In the context of this thesis, the topology that we use is graphs. Indeed, graphs and tabular data can be mixed up to create attributed graphs. Moreover, as seen before, deep neural networks can be used to tackle the issue of attributed graphs. They can also be used in the context of anomaly detection in regular vectorial datasets. Consequently, we are going to use them to detect anomalies in attributed graphs.

We also previously discussed the notion of graph embedding that allows transforming a graph into a usual vectorial dataset. Thus, a second way of detecting anomalies in attributed graphs is to, first, embed the nodes of the graph into a vectorial space such that it is possible to, then, detect anomalies in this new space. A low-dimensional space is preferable since it allows faster computation and may avoid the curse of dimensionality issue. However, deep learning models are not human-understandable due to their complexity. Thus, new methods have to be developed to understand the decision process of a complex model.

1.4 Explainable Artificial Intelligence

Machine learning is usually a tool to help humans make decisions. A lot of methods have been developed for decision aid. From linear models to deep neural networks, there is a wide range of models in machine learning. Deep learning improved a lot the ability of models to predict accurately. The predictions of any model must be understandable for legal purposes, to verify the correctness of the results, or for fairness for example. But, the main issue of deep neural networks is that they are too complex to be humanly understood. The series of Generative Pre-trained Transformer, GPT, models by OpenAI is a good example of how huge these models can grow. The first GPT model, GPT-1 [Radford et al., 2018], had 117 million parameters. The second version, GPT-2 [Radford et al., 2019], had 1.5 billion parameters. Finally, the most recent version, GPT-3 [Brown et al., 2020], has roughly 175 billion parameters. Clearly, these models are better each time and can deal with natural language processing very efficiently. But, in the meantime, the very complex structure of these networks made them not human-understandable. Indeed, it is not possible to interpret a few hundred of billions of parameters of a deep neural network. Moreover, explainable artificial intelligence has become primordial with the European General Data Protection Regulation, GDPR. The article 13 states that "the controller shall, at the time when personal data are obtained, provide the data subject with the following further information necessary to ensure fair and transparent processing: the existence of automated decision-making, including profiling and, at least in those cases, meaningful information about the logic involved". For a controller to provide such data, it is necessary to understand the functioning of his algorithm. This is exactly the aim of explainable artificial intelligence. Thus, the main challenge of explainable artificial intelligence, or XAI, is to make sense of the process that led a model to a certain decision.

For instance, consider one of the first methods that have been used: the linear model.

Definition 1.4.1 (Linear model). *A linear model is a statistical model that consists of learning the parameters W , or the weights, to find a relationship between some observations \mathbf{y} , and some explanatory variables X .*

The linear relation is simply

$$\mathbf{y} = X \times \mathbf{w} \quad (1.37)$$

It expresses the links between an observation y_i and potential explanations x_{ij} . The weights w_j indicates the relative importance of each explanatory variables. For a particular instance i , we have

$$y_i = w_1 * x_{i1} + w_2 * x_{i2} + \dots + w_j * x_{ij} \quad (1.38)$$

Thus, the higher a weight w_k , the more the related variable x_{ik} is important to explain the observed variable. On the contrary, a low weight shows that this variable is not relevant when explaining an observation. These linear regression models are easy to understand as their output is a linear combination of the input [Efron et al., 2004b]. The weight given to each explanatory

variable is clearly human-understandable and indicates the importance of that variable in the decision process.

While linear models are at the basis of many recent methods, another type of model is the set of trees [Breiman et al., 1984]. A tree is a tool for decision aid that consists of splitting a set of observations y_i into homogeneous groups corresponding to their label. A tree is made of a root, in which the observations stand, some nodes, in which there are subparts of the initial observations, and links between them, also called splits, that indicate how to separate the observations. Assume a population with two groups, the aim of a split is to reduce the number of elements of one group while retaining the maximum of individuals of the second group by examining some explanatory variables x_{ij} . A binary tree will split any node into two children nodes.

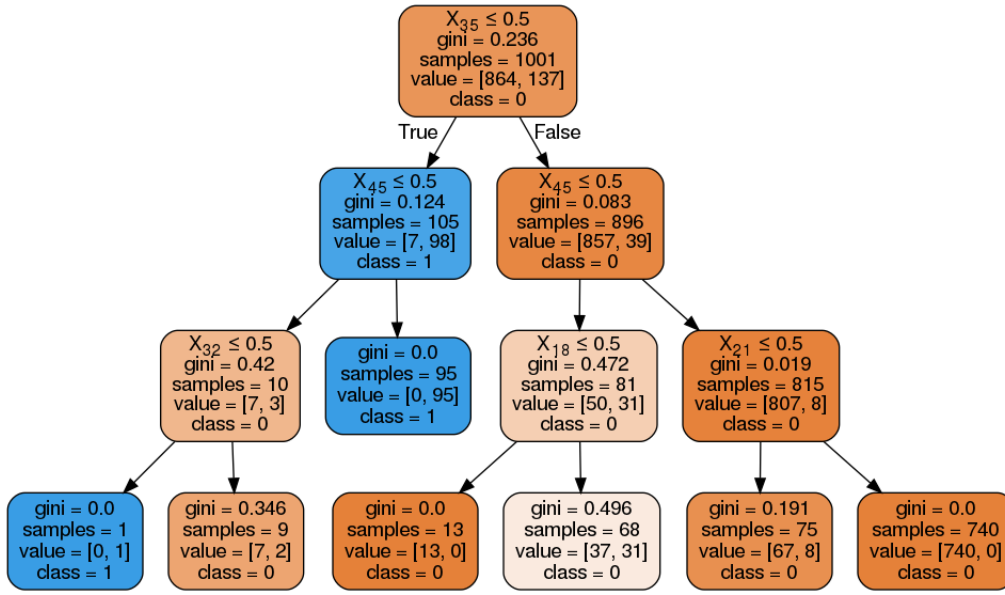


Figure 1.15: An example of a binary decision tree.

On the example tree presented in Figure 1.15, the root is made of 1001 samples, or observations, distributed among two groups. The group, or class, 0 has 864 elements and the group 1 has 137. The Gini value is a statistical measure of the distribution of a variable into a population. Low values of Gini indicate that a node is nearly pure, *i.e.* the elements of this node nearly all belong to the same class. On the contrary, a high value of Gini indicates that the elements of a node are mixed up between different groups. Finally, a split is made according to a specific explanatory variable x_{ij} and a threshold. For example, the first split is made with regards to the 35-th variable X_{35} with a threshold 0.5. Thus, every element of the root whose variable X_{35} is lower than 0.5 will be assigned to the left child and the others to the right child. Each node of the tree is then split again into two children until a maximum depth, *i.e.* a maximum number of splits, is reached. In the end, we have leaves, or final nodes, that are ideally pure or, at least, with many more elements of one group. The series of splits that leads a specific observation into a specific leaf is the decision process. For example, the leaf on the

far right is made of observations which are described by $X_{35} > 0.5$, $X_{45} > 0.5$ and $X_{21} > 0.5$. Binary decision trees are simple models that allow classifying elements of a population into several leaves. The path to go from the root to a leaf is the decision process. This path is human-understandable.

The classification task is one of the most studied tasks in machine learning. It consists of predicting a class for each instance of the dataset. Both linear models and binary decision trees can be trained with part of the dataset, the training set, to learn the best parameters and then to classify an unseen instance. Although they are easier to understand than deep neural networks, linear models and decision trees lack predictive power. To solve this, trees can be aggregated together to make a forest. Random forests consist of training many different trees with different parameters. Then, the classification is made by a majority vote of the trees. While random forests are a first step towards the improvement of the prediction power, they already lost a lot of understandability. Indeed, it is harder to understand the process of classification as many trees may not predict the same thing. For their part, linear models were a first step towards deep neural networks. As discussed previously, deep neural networks tend to be always bigger. It is now impossible to understand the decision process of a neural network. On the other hand, these deep networks are, at the moment, the best tool for classification purposes and many more different tasks.

As a result, it is necessary to bring back some understandability in the decision process. But the key element is to keep a high enough precision of the decision while making the decision process understandable. Two main ways have been proposed. The use of self-explanatory models is a first step to explain the decision of a machine learning model. It is either made by the use of linear models or binary decision trees but also by adding understandability to a more complicated model. Another way to explain the decision process is to learn a simpler and interpretable new model that is can explain the decision of the predictive model. That is the approach we retained in our work as we will see in Chapter 2

Chapter 2

Graph contextual anomalies detection

Anomaly detection has been a field of intense research for the last decades, both for graph data [Akoglu et al., 2015] [Ranshous et al., 2015] [Ma et al., 2021] and for vector data [Mehrotra et al., 2017] [Aggarwal, 2016] [Su and Tsai, 2011] [Hodge and Austin, 2004]. According to these last ones, anomalies are substantial variations from the norm. This introduces two main questions. The first one is how to describe the norm while the second is how to quantify the deviation to this norm. There are a lot of possible choices for both norm and deviation. For example, in regular vectorial data, the norm is usually made of instances of the dataset that are grouped together and the deviating instances are the ones that lie further in the space. As in regular vectorial data, there can be many different types of anomalies in relational data. They can be particular instances of nodes, edges, subgraphs, or even full graphs such as bridges between communities, irregular subgraphs, or out-of-power-laws nodes that are a few of the many possible definitions of anomalies in networks. A node can be an anomaly because of its neighborhood, its attributes, or a combination of both. For instance, in a graph with a community structure (i.e. containing sets of highly connected nodes with few connections to the rest of the graph), an anomaly can correspond to a node that does not really belong to any community either because it is isolated or because it forms a bridge between two groups. Assortativity is a measure of how much does a pair of connected nodes look like each other. If a network is assortative, or assortatively mixed, then nodes that are linked together tend to have similar attribute [Newman, 2002] [Newman, 2003] [Anagnostopoulos et al., 2008]. On the other hand, if a network has the opposite property, it is called disassortativity. In an attributed graph with assortativity, an anomaly can be a node whose attributes are significantly different from those of its neighbors.

In this thesis, we introduce and study a new particular case of node anomaly in attributed networks: context-based anomaly or contextual anomalies. Context-based anomalies correspond to nodes of a graph whose local neighborhood present a specific arrangement. This kind of anomaly is relatively frequent in practice. For instance, in a bibliographic network where the nodes correspond to papers assigned to thematic categories and there is a link from a document node to another if the first one cites the second, a contextual anomaly can correspond to a

node belonging to the category ‘Archeology’ (because it contains the word ‘excavation’) which is cited by documents belonging to the category ‘computer science’. Context-based anomalies are also often associated with fraud or corruption. In those situations, experts try to use "patterns" or contexts to find these frauds or corruption cases. But, of course, these contexts are not known by the experts. For instance, a company has a CEO who has an account in a tax haven then this company is more likely to be fraudulent. To clearly define this kind of anomaly, we consider that there exists an unknown small subgraph (a context), and a distinguished node in this subgraph, such that each time this subgraph occurs in the data, then the node corresponding to the distinguished node is an anomaly with a high probability p . In this thesis, for simplicity, we consider the case $p = 1$ but our work can be extended to a situation where this probability is lower. This unknown subgraph is called a context and the corresponding anomaly is "a context-based anomaly". A few examples are given in Figure 2.1. C_1 and C_2 are contexts defined on the neighborhood of the distinguished node and C_3 is defined on both the node itself and its neighborhood.

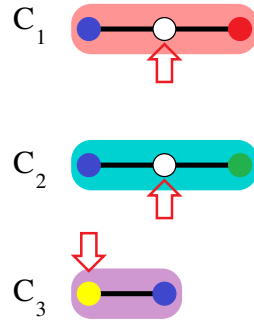


Figure 2.1: Three different types of contextual anomalies and their context. C_1 describes a node with no specific feature that has at least one blue neighbor and at least one red neighbor. C_2 is the same with blue and green. C_3 depicts a node that is yellow and has at least one blue neighbor. Contexts can be defined on the nodes themselves and in their vicinity.

We argue that these context-based anomalies are interesting and, as the experiments show, not always well detected by current approaches.

The detection of anomalies in networks can be tackled in a supervised, semi-supervised, or unsupervised way [Su and Tsai, 2011] [Hodge and Austin, 2004]. A supervised method requires some labeled data, anomalous nodes, and normal nodes, to train a model that will be used to detect anomalies among the other unlabeled nodes of the graph. For the semi-supervised setting, the model is trained with labeled data too but also with unlabeled data. That is, information of the nodes is aggregated onto labeled nodes to learn a model that can be used to detect anomalies. Finally, in a fully unsupervised manner, a model has to be constructed without labeled data to detect anomalies. While an unsupervised model does not require labeled data to be made, it performs usually, not always, worse than a supervised or semi-supervised one. In the following, we use a semi-supervised approach. More precisely, we have a transductive setting as in Figure 2.2: the data consist of a single graph for which a proportion of the nodes is

labeled (either "anomaly" or "normal") and the other nodes have no label. The goal is to find the labels of those unlabeled nodes belonging to the same graph. The difference with a fully supervised setting is that the learning algorithm can use the unlabeled nodes and their features, or attributes, even if it has no access to their labels which is helpful since a single graph has a singular topology. Thus, information of every node is required to be able to process it correctly. Removing unlabeled nodes from the learning would change the connectivity of the graph and thus disrupt it.

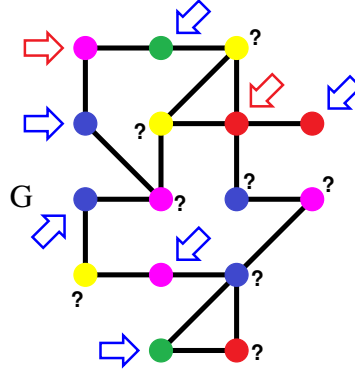


Figure 2.2: In a semi-supervised setting, some nodes are tagged as anomalies (red arrows), some nodes are tagged as normal (blue arrows) and the remaining nodes (with question marks) have to be classified into the normal class or the anomaly class.

Node anomaly detection can be addressed in many ways: finding general laws, using custom criteria, or as byproducts with community detection algorithms [Akoglu et al., 2010] [Chakrabarti, 2004]. But recent years have seen the rise of representation learning on graphs that consists of finding a representation of the nodes in a feature space and then identifying anomalies in this space. The features can be hand-made (not learned) or automatically learned, for instance by a Graph Neural Network (GNN) which is the state-of-the-art for node classification and, more generally, for solving many supervised or unsupervised problems on graphs [Veličković et al., 2018]. Through this thesis, we propose to use this approach to automatically and simultaneously learn a suited representation of the nodes and detect the anomalies. More precisely, we propose CoBaGAD, for **C**ontext-**B**ased **G**raph **A**nomaly **D**etector, a semi-supervised algorithm to detect contextual anomalies. CoBaGAD is a variation of Graph Attention Networks (GAT) where the attention mechanism has been changed by a custom one allowing better feature selection.

To carry out an experimental evaluation of CoBaGAD, we need datasets. However, to our knowledge, real data for this kind of anomaly is not publicly available. Thus, we created a generator of contextual anomalies to add anomalies to a graph.

Our contributions are:

- A new kind of node anomaly in a graph: contextual anomaly.
- A generator of contextual anomalies.

- A GNN architecture to detect it.
- Experiments to validate our model on several kinds of graph with different types of contextual anomalies and compare it with Graph Attention Networks [Veličković et al., 2018], Graph Convolution Networks [Kipf and Welling, 2017], GraphSage [Hamilton et al., 2017] and Node2vec [Grover and Leskovec, 2016] + LOF [Breunig et al., 2000].

This chapter is organized as follows. First, we review related works of anomaly detection in both vectorial and graph data. We mathematically define the problem in Section 2.2 and present our method to detect context-based anomaly in attributed graphs in Section 2.3. Then, we describe our datasets in Section 2.4, our evaluation protocol, and the experiments carried out to evaluate the ability of our method to detect anomalies in Section 2.5.2. Finally, we discuss the obtained results which are generally better than those provided by state-of-the-art methods.

2.1 Related work

Many different approaches have already been proposed to tackle anomaly detection for both vector data and relational data. Through this section, we discuss previously published methods that were created to deal with anomaly detection. Please note that the words "outliers" and "anomalies" will be used interchangeably. According to [Aggarwal, 2016], "outliers are also referred to as [...] anomalies in the data mining and statistics literature". Obviously, graph anomaly detection techniques are very interesting since they directly deal with the type of data we are interested in. On the other hand, we also review anomaly detection techniques for usual vector data since, as seen in Section 1.2.3, the use of graph embedding methods provides a low-dimensional representation of a graph and can, thus, be fed to such an anomaly detector. In the following, we first discuss the case of outlier mining in the context of vector data. Then, we review state-of-the-art models that detect anomalies in graph data.

2.1.1 Anomaly detection with vectorial data

We can distinguish two main fields of research in anomaly detection according to the type of data that we investigate. First, anomaly detection with vectorial data aims at finding vectors in the space that are "far" from the others. Different criteria have been proposed to quantify the notion of "far" from the others. On the other hand, graph anomaly detection is a related field of research that focuses on relational data. While they do not use the same type of data, they share common ideas.

In regular data, where instances lie in \mathbb{R}^f , many types of anomalies can be defined depending on their proximity to the rest of the instances. In Figure 2.3, we can see three types of outliers. First, global outliers are instances of the data that are far from all the other points. The notion of distance can be defined in many ways and will be discussed later but influences a lot the

detection of outliers. Second, local outliers are instances that are far from a cluster, or group, of data points but are not necessarily far from all the points of the data set. Lastly, collective outliers are small groups of instances that are split from the rest of the data but do not contain enough points to be considered as a cluster. These different notions of outliers imply a lot of different ways to detect them. Many distances can be introduced in \mathbb{R}^f . This can lead to the use of densities or sub-space discovery to isolate outliers.

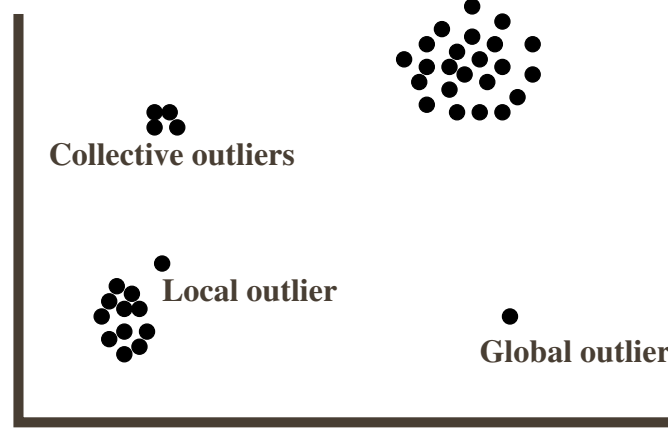


Figure 2.3: Different types of anomalies in vectorial data.

Following the definitions of [Ting et al., 2018], anomalies can be related to different methods based on the notion of distance, neighborhood, or density:

- Density-based methods: anomalies are instances of the data in regions of low density.
- Isolation-based methods: anomalies are instances of the data which are most susceptible to isolation.
- k-th nearest neighbor methods: anomalies are instances of the data that are the furthest to their k-th nearest neighbor.

To that list, we also add some specific methods:

- Kernel-based methods: kernels measure the similarity between two points. Anomalies are instances that are dissimilar to all the other points.
- High dimension methods: due to the curse of dimensionality, finding anomalies in high dimensions is challenging and some methods have been developed to tackle this issue.

In the sequel, we will present examples of methods belonging to these different families: density-based, tree-based, k-nearest neighbors, kernel-based or high-dimension-based techniques.

2.1.1.1 Density-based methods

Density methods focus on instances of the data that are "alone", *i.e.* there are very few other instances in the neighborhood of this specific instance. Mathematically speaking, density is

the number of instances in a specific area of the space divided by the volume of this subspace. While this notion can be used in the context of subspace anomaly detection, the notion of local density is rather used in this case.

Among the popular methods, Local Outlier Factor (LOF) [Breunig et al., 2000] is a density-based measure to detect points in the space that are far from the others by comparing their local density to that of their neighbors. More precisely, the authors define the local reachability density, lrd . For a data point $p \in \mathbb{R}^f$, the local reachability density is the inverse of the mean of the distance to its neighbors.

$$lrd(p)^{-1} = \frac{1}{|\mathcal{N}_k(p)|} \times \sum_{q \in \mathcal{N}_k(p)} D(p, q) \quad (2.1)$$

where $\mathcal{N}_k(p)$ is the set of its k nearest neighbors and D a distance of \mathbb{R}^f . Then LOF is defined as the average of the neighbors' local reachability density normalized by that of the instance we are studying.

$$LOF(p) = \frac{1}{|\mathcal{N}_k(p)|} \times \sum_{q \in \mathcal{N}_k(p)} \frac{lrd(q)}{lrd(p)} \quad (2.2)$$

LOF compares the local density of the point p with the local density of its neighbors. Thus, an outlier can be detected as its local density is low compared to that of its nearest neighbors.

For their part, [Ester et al., 1996] introduced Density-Based Spatial Clustering, or Applications with Noise, or DBSCAN, a clustering algorithm. Clustering is, usually, an unsupervised technique that consists of grouping similar data instances into clusters in such a way that similar instances are in the same cluster. DBSCAN does not force every data point to belong to a cluster. Thus, by assuming that normal instances are those in a cluster and anomalies are those that do not belong to a cluster, DBSCAN can detect anomalies. In this case, anomalies are byproducts of the clustering process.

DBSCAN uses two parameters: a distance ϵ and a minimum number of points $MinPts$ that must stand in a radius of ϵ to be considered a cluster. The main idea of the method is to look at the ϵ -neighborhood V_ϵ of an instance p . The ϵ -neighborhood is defined as the set of data points that are in the sphere of radius ϵ and center p .

$$V_\epsilon = \{q | D(p, q) < \epsilon\} \quad (2.3)$$

The next step is to check whether there are $MinPts$ instances close to it, *i.e.* the number of elements of V_ϵ is higher than $MinPts$. If so, the data point belongs to a cluster and the instance must be added to the right cluster. On the contrary, if an instance does not have enough points close to it, then it is regarded as the noise of the data and considered as anomalous. Clustering methods are very interesting since they can operate in an unsupervised setting and their testing is fast since there are a lot less groups than instances of the data. But, on the other hand, such methods are generally not explicitly made to detect anomalies and can be less precise than some others.

Another way to find anomalies is to rank them according to some measures and focus on the most important ones. That is what [He et al., 2003] did. They proposed CBLOF for Cluster-Based Local Outlier Factor as an extension of LOF. CBLOF is a measure based on the size of the cluster an instance belongs to and its distance to the closest cluster. The first step of the method consists of clustering the data set into several clusters. The authors argue that any proper clustering algorithm can be used. Then, clusters are divided into two sets. Large clusters and small clusters are defined thanks to two hyper-parameters. The first parameter controls how many instances must lie in the set LC of the large clusters or in the set SC of small clusters. For example, it specifies that 90% of the data points must be in the large clusters. The second parameter sets the relative size difference between the smallest large cluster and the biggest small cluster. For example, the large clusters must all be at least ten times bigger than any small cluster. Then, for an instance p in a small cluster C_i , the cluster-based local outlier factor is defined as:

$$CBLOF(p) = |C_i| \times \min_{C_j \in LC} D(p, C_j) \quad (2.4)$$

where $p \in C_i$ and $C_i \in SC$. Thus, for an instance in a small cluster, the CBLOF is determined by the size of its cluster and the distance D to its closest large cluster relative to each instance in this cluster.

For an instance p in a large cluster C_i , the CBLOF is defined as:

$$CBLOF(p) = |C_i| \times D(p, C_i) \quad (2.5)$$

where $p \in C_i$ and $C_i \in LC$. In this case, the CBLOF is given by the size of the cluster the instance belongs to multiplied by the distance to its cluster. The distance between an instance and a cluster can be the one given by the chosen clustering algorithm. Finally, based on the CBLOF, the different instances of the data can be ranked. According to this ranking, data points can be tagged as anomalies.

When data is not distributed along linear manifolds, [Sathe and Aggarwal, 2016] come in. They proposed LODES, a local density Spectral Outlier Detector to tackle the issue of outliers embedded in arbitrary manifolds. LODES uses spectral clustering which consists of a spectral decomposition of a specific matrix. This matrix is based on pairs (i, j) of instances that are mutually in the set of k -nearest neighbors. For each such pair, $i \in \mathcal{N}_k(j)$ and $j \in \mathcal{N}_k(i)$, a similarity w_{ij} can be computed using a kernel. These pairs and weights define a weighted undirected graph. Thus, the degree d_i of each instance can be computed and the normalized weight matrix can be defined as:

$$w'_{ij} = \frac{Linkage_Density}{Symmetric_Density_Difference} = \frac{w_{ij}}{(d_i - d_j)^2} \quad (2.6)$$

The next step consists of computing the corresponding degree matrix D' and, then, the Laplacian as the difference between the new degree matrix D' and the normalized weight matrix

W' . The final step of the spectral embedding is the computation of the first eigenvectors of the Laplacian which give the embedding. This new representation of the data can be used to detect anomalies. A first idea is to apply a k -nearest neighbor approach but the authors argue that there is a better method. They use an iterative approach based on the decomposition of the Laplacian to adapt the weights w_{ij} and compute an anomaly score as the average of the distance to the neighbors of the neighborhood graph.

2.1.1.2 Tree-based methods

As discussed in Chapter 1.4, trees can be used to split the data into several groups. The root of the tree contains all the instances of the data. Every node of the tree can be split. A split is a simple, often binary, rule to discriminate the population of a node of the tree into two, or more, children. The fewer rules to reach a specific instance, the easier it is to describe it with some simple rules. This is the point of view of anomaly detection methods that use trees.

Based on this underlying idea, anomalies can be found thanks to an isolation forest. iForest [Liu et al., 2008] is an outlier detection method based on isolation trees, iTrees. Such trees are binary decision trees that aim at cutting the space into sub-spaces by splitting it along specific coordinates. The process to grow a tree is iterative. At each step, a feature, or coordinate, c and a value v are randomly selected. A sub-sample of the data set, a node of the tree, is split into a left child containing instances i where $c_i < v$, and a right child containing instances i where $c_i \geq v$. An isolation tree step is a random cut along a specific axis of the space. The process is repeated until the tree is fully grown *i.e.* each instance of the data is isolated in a leaf. Once built, it is possible to look for a specific instance in the tree. The quantity $h_l(p)$ is the path length to reach the instance p in the l -th tree. If $h_l(p)$ is low, it means that it is easy to isolate this instance of the data. On the contrary, an instance that is hard to isolate will have a high $h_l(p)$. Finally, an anomaly score is computed for every data point p as:

$$score(p) = \frac{1}{T} \sum_{l=1}^T h_l(p) \quad (2.7)$$

where T is the total number of iTrees. The score is the average of all the path lengths among the different trees of the forest. Thus, the highest the score, the lowest the probability of being an anomaly. It is, thus, possible to focus on the instances of the data that are outliers with the highest probability. While this method is very efficient and provides a lot of insights when looking for anomalies, it has also some drawbacks. Due to the non-deterministic cuts along some specific axis, the method can miss some anomalies.

An improvement of iForest is iNNE [Bandaragoda et al., 2018]. The authors proposed isolation using Nearest Neighbors Ensemble, iNNE, which is an alternative of iForest. It overcomes its main weaknesses: inability to detect local anomalies due to the use of a global score, anomalies that are masked by axis-parallel clusters, anomalies with a high percentage of irrelevant

attributes and anomalies in multi-modal data sets. iNNE provides a local distribution adaptation for each region of the space. It also uses all the coordinates of the space instead of a sample of them. Thirdly, it can detect local anomalies through the use of a local score. First, iNNE defines the maximum sphere $\mathcal{B}(p)$ containing p and no other data point:

$$\mathcal{B}(p) = \{\mathbf{x} \in \mathbb{R}^f / \|\mathbf{x} - p\|_2 < \tau(p)\} \quad (2.8)$$

where $\tau(p) = \|p - NN(p)\|_2$ is the radius of the sphere and $NN(c)$ is the nearest neighbor of c . Then, a subsample \mathcal{S}_i is drawn from the initial dataset without replacement. The quantity $cnn_i(\mathbf{x})$ is defined as:

$$cnn_i(\mathbf{x}) = \underset{c \in \mathcal{S}_i}{\operatorname{argmin}} \{\tau(c) | \mathbf{x} \in \mathcal{B}(c)\} \quad (2.9)$$

This quantity allows the authors to derive a local anomaly score. First, the isolation score $score_i$ of \mathbf{x} with respect to the subset \mathcal{S}_i is given by:

$$score_i(\mathbf{x}) = 1 - \frac{\tau(NN(cnn_i(\mathbf{x})))}{\tau(cnn_i(\mathbf{x}))} \quad (2.10)$$

if $\mathbf{x} \in \cup_{c \in \mathcal{S}_i} \mathcal{B}(c)$ and $score_i(\mathbf{x}) = 1$ otherwise. This isolation score is maximum when the instance p is located far from all points in \mathcal{S}_i . Finally, the local anomaly score for a data point \mathbf{x} is defined as the average of all the isolation score for the different subsets.

$$score(\mathbf{x}) = \frac{1}{T} \sum_{i=1}^T score_i(\mathbf{x}) \quad (2.11)$$

where T is the total number of subsampling. In the end, instances can be ranked according to their potential of being an outlier.

2.1.1.3 k-nearest neighbors methods

One of the most straightforward ideas to detect anomalies that are far from the rest of the data is to look at the set of the nearest neighbors. If the closest instances to a specific data point are far, then it is likely to be an anomaly. Many methods have been proposed based on these nearest neighbors.

For [Ramaswamy et al., 2000], "a point p in a dataset is an outlier with respect to parameters k and d if no more than k points in the dataset are at a distance of d or less from p ". They propose three algorithms that rank the data points p according to their distance $D^k(p)$ to their k -th neighbor and they take the top ranks as outliers. For a data point p , the distance is defined as:

$$D^k(p) = \max_{q \in \mathcal{N}_k(p)} D(q, p) \quad (2.12)$$

where $\mathcal{N}_k(p)$ is the set of the k nearest neighbors of p . First, they propose two relatively straightforward algorithms which are a nested-loop join algorithm and an index join algorithm. The main issue with these methods is that they need to compute the distance $D^k(p)$ for every instance in the data set. To tackle the issue of being computationally expensive, the authors showcase a partition-based algorithm. Its four steps are as follows:

- partition the dataset using a clustering algorithm (BIRCH [Zhang et al., 1996]).
- For each cluster, compute lower and upper bound for $D^k(p)$ with p in the cluster.
- Find candidate partitions in which there are potential outliers.
- Find outliers in those partitions

This algorithm is computationally much more efficient than the two other described.

For their part, [Angiulli and Pizzuti, 2002] propose a new definition for distance-based outliers that takes the sum of the distances from its k nearest neighbors into account. Then, outliers are instances of the data which have the largest distance. To find the k nearest neighbors, they linearize the search space with Hilbert space-filling curve iteratively with $f + 1$ shifts of the data set where f is the number of features or dimensions. Hilbert space-filling curve is mapping between values in a 1-D interval I and the coordinates of f -dimensional points in \mathcal{S} such that if two points are close in the initial space I , they are also close in \mathcal{S} but the converse is not necessarily true. This first step reduces a lot of the potential nearest neighbors. Then, a final step can find exactly the set $\mathcal{N}_k(p)$ of the k nearest neighbors of a data point p . Once the set of nearest neighbors is known, it is easy to rank the different data points with the distance $D_k(p)$ defined as:

$$D_k(p) = \sum_{q \in \mathcal{N}_k(p)} D(q, p) \quad (2.13)$$

This sum could be normalized by the number of elements of $\mathcal{N}_k(p)$ but it is not necessary as every point has, by definition, the same number of k -nearest neighbors. Finally, the different instances of the data are ranked according to their distance $D_k(p)$. The data points with the highest distance $D_k(p)$ are those which are flagged as outliers.

2.1.1.4 Kernel-based methods

Kernel-based methods [Ting et al., 2016], [Ting et al., 2018] are recent methods that avoid distance-based neighborhood problems, like the curse of dimensionality notably. The authors introduce a data-dependent dissimilarity, the mass-based dissimilarity. From this dissimilarity, they introduce a new function that can recover the neighborhood. Finally, they show that their approach outperforms previously state-of-the-art models. The kernel introduced in [Ting et al., 2016] is defined as follows. Let \mathcal{S} be a data sample drawn from a probability distribution function \mathcal{F} . Let $H_0 \in H(\mathcal{S})$ be a hierarchical partitioning of the space into non-overlapping and non-empty regions, where $H(\mathcal{S})$ is the set of all possible hierarchical partitions. For two points p and q , the set of partitions that they simultaneously belong to is:

$$J_{p,q} = \{r \subset H_0 | (p, q) \in r\} \quad (2.14)$$

Then, they define the smallest local region covering p and q with respect to H_0 and \mathcal{S} as:

$$R(p, q|H_0; \mathcal{S}) = \underset{r \subset H_0 \text{ s.t. } \{p, q\} \in r}{\operatorname{argmin}} \sum_{z \in \mathcal{S}} \mathbb{1}(z \in r) \quad (2.15)$$

where $\mathbb{1}$ function is the indicative function. Then, the mass-based dissimilarity of p and q with respect to \mathcal{S} and \mathcal{F} is defined by:

$$m(p, q|\mathcal{S}, \mathcal{F}) = \mathbb{E}_{H(\mathcal{S})}[P_{\mathcal{F}}(R(p, q|H; \mathcal{S}))] \quad (2.16)$$

where \mathbb{E} is the expectation taken over all models in $H(\mathcal{S})$ and $P_{\mathcal{F}}$ is the probability with respect to \mathcal{F} . In practice, the dissimilarity is estimated from a finite number of models and the expectation becomes an average. Afterward, the data set is partitioned into iTrees, then, the data points "are traversed through each tree to record the mass of the nodes". The sum of mass of the lowest nodes containing both p and q is $\sum_i |R(p, q|H_i)|$ and the mean over T iTrees of these mass is given by:

$$m_e(p, q) = \frac{1}{T} \sum_{i=1}^T \frac{|R(p, q|H_i)|}{|\mathcal{S}|} \quad (2.17)$$

Finally, μ -neighborhood mass is defined as $M_{\mu}(p) = \{q \in \mathcal{S} | m_e(p, q) < \mu\}$. In the end, anomalies are defined as those instances which have the highest probability mass to their k -th lowest probability mass neighbors in a given data set.

2.1.1.5 High dimensional anomaly detection

A key issue in anomaly detection is the curse of dimensionality. We already discussed it previously in Section 1.3 and showed why this is a problem. Thus, some methods have been proposed to tackle to the case of high dimensions.

Instead of using a regular distance, like the euclidean distance, it is possible to use the angle between two vectors. [Kriegel et al., 2008] introduced **ABOD** for Angle-Based Outlier Detection. This is a non-parametric approach for outlier detection in high dimensional data based on the use of the angle between two vectors instead of the euclidean distance between any pair of data points. They define the angle-based outlier factor. It corresponds to the variance over the angles between the difference vectors of p to all pairs of points weighted by the distance of the points. For a data point p , we have

$$ABOF(p) = Var_{q,r} \left(\frac{\langle q - p, r - p \rangle}{\|q - p\|^2 \times \|r - p\|^2} \right) \quad (2.18)$$

where $\langle p, q \rangle$ is the scalar product between p and q and Var is the variance. They use a speed-up version to lower the complexity. The variance is computed on a subset of the dataset which is the set of nearest neighbors of p . To speed up even more, a lower bound of $ABOF$ can be computed. In the end, the algorithm first looks for nearest neighbors, then, computes the lower bound of $ABOF$ for every data point. Then, it sorts the data points by the value of lower bound and finally, computes the real $ABOF$ of the potential outlier and check whether

they really are outliers.

For their part, [Vries et al., 2010] proposed **PINN** which stands for Projection Indexed Nearest Neighbors. It assesses the problem of finding outliers in very high dimensional spaces. The authors use LOF as an outlier score but instead of finding the nearest neighbors in the original space they first project the data into a lower dimension space with a random projection that preserve the k nearest neighbor distance. Then, they find the k nearest neighbors in the new space. Finally, they can compute $LOF(p)$ and discriminate data points according to their score. In the end, the main trick is to change the space of the dataset to be able to compute the usual Local Outlier Factor.

Recently, machine learning tools have been brought to deal with anomaly detection. Among many fields of research, representation learning has been very important. Representation learning consists of learning a mapping of the data into a low-dimensional space such that each instance of the input dataset is represented by a vector in low-dimension. This dimension reduction can be very useful when dealing with outlier detection. Indeed, the curse of dimensionality does not operate in low-dimension. Thus, [Pang et al., 2018] introduced RAMODO, a ranking model-based framework. It learns a low-dimensional representation of the data with a neural network to infer outliers. It takes, as input, a set of potential outliers and a set of candidate inliers. The model assumes a distance-based anomaly detector ϕ such as LOF. The neural network learns a mapping function f such that $\phi(f(p)) > \phi(f(q))$ if p is an outlier and q is an inlier. It is made of a single fully connected layer with reduced dimension and the activation function is a usual *ReLU* function. RAMODO optimizes an outlier score-based ranking loss to detect outliers. The candidate outliers are found using Sp [Sugiyama and Borgwardt, 2013], which is a fairly good but unstable outlier ranking, and are used as negative examples in the training process. Positive examples are sampled based on the inverse of their outlier score given by Sp. Once the network has been trained, it is possible to compute the new representation of the dataset. The new dataset is a set of low-dimensional vectors in which it is possible to use any distance-based anomaly detector previously discussed.

To sum up, there is a wide range of methods to detect anomalies in vector data. The type of anomalies detected often depends on the type of methods that are used to detect them. There are density-based, isolation-based, k -th nearest neighbor, kernel-based, and high dimension methods. While some methods are better suited for low-dimensional data, most of the recent works focus on detecting anomalies in high-dimensional data. Another challenge for anomaly detection is graph anomaly detection. All previous methods cannot be directly applied in graphs. For this reason, some methods have been developed specifically for graphs.

2.1.2 Anomaly detection in relational data

There are a lot of methods that aim at finding anomalies in graphs and a lot of different kinds of anomalies that those methods try to detect. Due to the complex topology of graphs, a variety of graph anomalies exists. Contrary to the case of regular vectorial datasets, it is not possible to define anomalies as an instance, or a small group, that is far from the rest of the data. The notion of distance in graphs is not the same. Especially in unweighted graphs, the distance to the nearest neighbors is the same for every node since the distance between a node and its neighbor is always one. On the other hand, the graph topology avoids computing the k nearest neighbors search that is, often, computationally expensive. Indeed, it is not necessary to look for the neighbors of every instance in the data since it is given by the graph itself. In the end, graphs are more complex than usual vectorial datasets but they provide their topology which is very useful to avoid some computations. On the other hand, this complexity implies the need for new methods to be able to deal with the detection of new types of anomalies. In the following, we review state-of-the-art models that deal with anomaly detection in different types of static graphs.

Methods can be cataloged according to their graph anomaly detection approaches. According to [Pourhabibi et al., 2020], there are five main categories:

- Community-based: find dense groups of nodes
- Probabilistic-based: find some statistics in the data and anomalies are deviations of the statistics
- Structural-based: exploit graph topological structure to find anomalies
- Compression-based: used to find suspicious activities in a dynamic network. These methods are out of the scope of this thesis and will not be discussed there.
- Decomposition-based: use the minimum description length to find elements that inhibit compressibility

While this typology is useful to categorize the methods, in this thesis, we will take another point of view and catalog the methods based on the kind of graph they take as input. We distinguish two cases: either a method needs a plain graph as input or an attributed graph.

2.1.2.1 Plain graphs

Plain graphs are unweighted unsigned un-attributed non-oriented static graphs. This means that edges have no weight, nodes and edges have no features describing them, edges are not oriented *i.e.* the interaction between nodes is reflexive, and the graph does not evolve over time. In this particular, yet the simplest, form of a graph, there are already a lot of potential definitions for a graph anomaly such as bridges between communities or unexpected edges. In the sequel, we describe methods dedicated to anomaly detection in graphs.

Autopart [Chakrabarti, 2004] is a parameter-free graph partitioning and edge outlier detection algorithm that belongs to the category of decomposition-based methods. It mines a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ to discover clusters, or groups of similar nodes, and outliers that are defined as edges deviating from the overall structure of the graph. Moreover, the algorithm must be automatic, *i.e.* without parameters, scalable and incremental in the sense that it can recompute the results to adapt to new data. The method relies on the Minimum Description Length [Rissanen, 1978] which is the minimum number of bits needed to encode a piece of data. It modifies the adjacency matrix by reordering the rows and columns such that blocks of high (or low) density are created. This way, nodes in the same block belong to the same group. Thus, nodes are indexed into k disjoint groups with an index function $I : \mathcal{V} \rightarrow \{1, \dots, k\}$. Moreover, there is a tradeoff between the homogeneity of the blocks and the number of blocks desired. To do so, the authors use the Minimum Description Length to count the number of bits needed to re-arrange the information in the adjacency matrix. The total encoding cost $T(A, k, I)$ of the adjacency matrix A for a given number of groups k and the index of groups I is:

$$T(A, k, I) = \log^*(k) + \sum_{i=1}^{k-1} \lceil \log(b_i) \rceil + \sum_{i=1}^k \sum_{j=1}^k \lceil \log(b_i b_j + 1) \rceil + C(A_{ij}) \quad (2.19)$$

where $\log^*(x) = \log_2(x) + \log_2 \log_2(x) + \dots$, b_i is the number of nodes in the i -th group and $C(A_{ij})$ is the code cost of the submatrix restricted to the links from group i to j . This cost has to be minimized. Their algorithm has two steps that are called recursively. First, for a specific number of blocks, the inner loop finds the best arrangement of the matrix. Then, the outer loops increases the number of groups and checks if the compression cost has decreased or not. Figure 2.4 shows this two-step process.

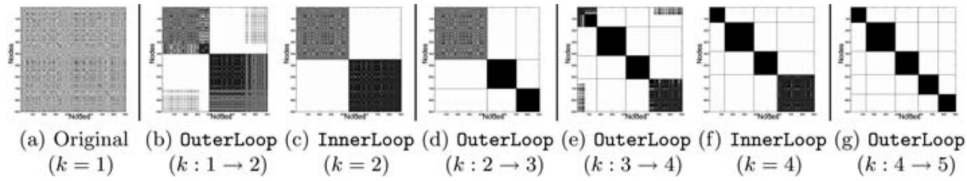


Figure 2.4: Two step process of the Autopart method [Chakrabarti, 2004]. First, arrange the matrix to have a specific number of groups. Then, increase the number of groups if necessary.

Finally, outliers are defined as edges that, if removed, reduce a lot the compression cost. It is possible to rank an edge (u, v) according to its outlierness which is defined as:

$$\text{outlierness}(u, v) = T(A', k, I) - T(A, k, I) \quad (2.20)$$

where $A' = A$ except that the edge (u, v) is removed. In the end, the authors can find the underlying structure in the graph. They introduce a lossless compression scheme for finding node groups and outlier edges. Their proposed algorithm is automatic and parameter-free.

Another decomposition-based method is **SUBDUE** proposed by [Noble and Cook, 2003]. It takes as input a plain graph and defines anomalies as "a surprising or unusual occurrence". The description length of a graph $DL(\mathcal{G})$ is defined as the minimum number of bits to encode this structure. The best substructure S of the graph is the one that minimizes the following value:

$$F(S, \mathcal{G}) = DL(\mathcal{G}|S) + DL(S) \quad (2.21)$$

where $DL(\mathcal{G}|S)$ is the description length of \mathcal{G} using S and $DL(S)$ is the description length of S . Substructures that reduce the description length a lot may appear frequently and anomalies appear, by definition, rarely. Based on this value $F(S, \mathcal{G})$, it is possible to rank substructures according to their degree of abnormality. Afterward, the authors provide experimental results of their method using real-world intrusion networks and artificially created data.

The main issue with these two methods that use the minimum description length principle is that they are computationally very expensive. Their complexity is exponential in the number of nodes in the worst case and can be reduced to polynomial. Such methods are not suitable for dealing with graphs of millions of nodes.

For their part, [Jimeng Sun et al., 2005] proposed an algorithm to detect bridges between communities in a bipartite graph. First, for every node, they identify similar nodes (their community) and then, they rank nodes according to their "normality" to identify the bridges between communities. The method takes as input a bipartite graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ such that $\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2$, $\mathcal{V}_1 \cap \mathcal{V}_2 = \emptyset$ and A is its adjacency matrix. For every node $v_a \in \mathcal{V}_1$, it computes a relevance score vector \mathbf{u}_a which is the personalized page rank (PPR) of its neighbors. This is done by finding the steady-state probability vector of:

$$\mathbf{u}_a = (1 - c)P_A\mathbf{u}_a + c\mathbf{q}_a \quad (2.22)$$

where P_A is the column normalized version of M_A which is the matrix 2×2 block matrix made of 0 on the diagonal, A on the top right and A^T on the bottom left. c is a parameter and \mathbf{q}_a is the one-hot vector corresponding to v_a . The steady-state is reached by simply iteratively repeating the Equation 2.22. Based on this score, the authors are able to compute a normality score. For a node $v_b \in \mathcal{V}_2$, let t be its number of neighbors. A $t \times t$ similarity matrix RS_t is computed where each element (a, a') is the similarity between \mathbf{u}_a and $\mathbf{u}_{a'}$. Finally, the normality score for the node $v_b \in \mathcal{V}_2$ is the mean of all elements of RS_t except the diagonal. Nodes with lowest normality are spotted as anomalies.

OddBall [Akoglu et al., 2010] is an approach that aims at detecting outliers as nodes of weighted graphs. This method belongs to the class of probabilistic-based models. The normal data is defined thanks to power-laws. Outliers are those nodes that deviate from these power laws. More specifically, for every node v_i , some features are extracted: its degree d_i , the number of edges in the ego network e_i , the total weight of ego network w_i , and the principal eigenvalue of the adjacency matrix of the ego network λ_i . Then, it could be possible to use a standard

anomaly detector in the space of extracted features but the authors promote the use of another technique. They group features into carefully chosen pairs. For example, they consider the number of edges of the ego network e_i of the nodes versus the total weight of the ego network w_i and they find that these two variables follow a power law. This is useful to detect heavy vicinities. The same is applied for e_i and d_i to detect near-cliques and stars and for λ_i and w_i to detect single dominating heavy edges. These outliers are defined as nodes whose features deviate from these power laws. An example of the laws found by OddBall is presented in Figure 2.5.

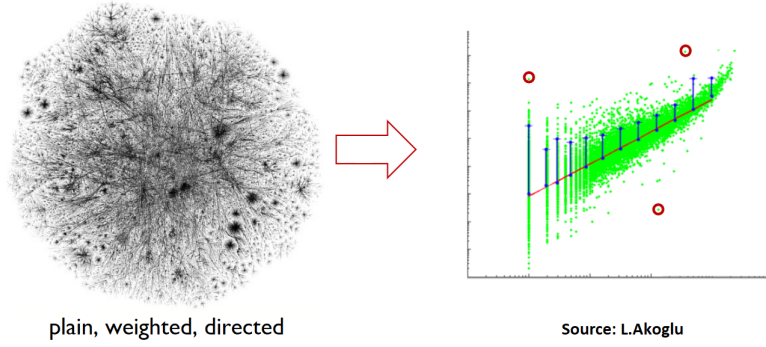


Figure 2.5: From a graph, some features are extracted to compute power laws. Outliers are those nodes whose features deviate from the laws, in red on the right figure. Source [Akoglu et al., 2010]

2.1.2.2 Attributed graphs

Sometimes, graphs have additional information on their nodes or edges. In this section, we will discuss the case of attributed graphs $\mathcal{G}(\mathcal{V}, \mathcal{E}, X)$ where the additional information is given by the feature vector $\mathbf{x}_i \in \mathbb{R}^f$ for the node v_i .

Goutrank [Müller et al., 2013] is a graph outlier ranking method for attributed graphs that is part of the community-based methods. It promotes the use of ranking techniques to focus on the most important outliers and argue that it is more suitable in some situations. Goutrank is one of the first anomaly detectors that can deal with attributed graphs and not only with a plain graph or with vector data. To find outliers, nodes are ranked according to their degree of deviation in both graph and attribute properties. The method has two key steps. It, first, clusters the feature space into subspaces and the graph into subgraphs. Then, Goutrank gives a score $score(v_i)$ to each node v_i depending on the cluster partitioning: the more the node belongs to subspaces, the less likely it is to be an anomaly. To do so, they define a Subspace Clustering Result as a set of subspace clusters $\{(C_1, S_1), (C_2, S_2), \dots\}$ where $C_j \subset \mathcal{V}$ is a densely connected subgraph with high attribute similarity in the feature subspace $S_j \subset \mathbb{R}^f$. In a dense subgraph, there must be a correlation between the graph structure and some attribute values.

This step is done with the use of the methods proposed by [Moser et al., 2009] and [Günemann et al., 2010]. The next step is the ranking of the nodes with regards to the found Subspace Clustering Result. The authors propose a score for each node v_i with node degree importance:

$$score(v_i) = \frac{1}{3} \sum_{\{(C_j, S_j) | v_i \in C_j\}} \frac{|C_j|}{c_{max}} + \frac{|S_j|}{s_{max}} + \frac{d_i}{d_{max}} \quad (2.23)$$

where $|C_j|$ is the number of elements in C_j , $|S_j|$ is the number of attributes of S_j , c_{max} is the size of the biggest cluster, s_{max} is the maximal dimensionality of the subspaces and d_{max} is the maximum degree of the nodes. Thanks to this score, the nodes can be ranked and a user can focus on the nodes with the lowest scores.

FocusGo [Perozzi et al., 2014b] is an outlier detection method for an attributed graph which focuses on user-specific communities to study the more interesting parts of the graph. This method belongs to the community-based methods. An anomaly is a node of a community that deviates from the focus attribute. Through a set of user-provided nodes, the aim of the method is to infer these focus attributes. Then, communities and outliers are simultaneously mined. Thanks to local clustering, the algorithm can efficiently scale to large graphs. The power of focus attributes is illustrated in Figure 2.6.

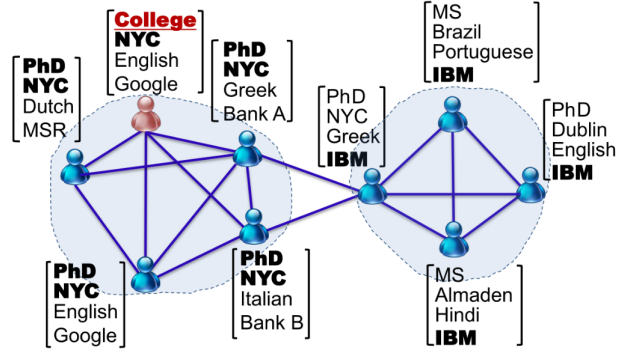


Figure 2.6: Example of a graph. Two communities focus each on a specific attribute (NYC and IBM). The outlier is in the NYC community and does not have a PhD but did College instead. Source [Perozzi et al., 2014b]

The input of the method is an attributed graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, X)$ and a set of user-specific focus nodes C_{ex} which are considered to be similar to the type of nodes the communities of interest should contain. The first step is to find the focus attributes which are the few features that unite the nodes in a community. They use the Mahalanobis distance $(\mathbf{x}_i - \mathbf{x}_j)^T W (\mathbf{x}_i - \mathbf{x}_j)$ to capture the importance of each feature where \mathbf{x}_i is the feature vector of v_i and W is a diagonal matrix to learn with a distance metric learning problem. Thus, each element W_{ii} is the importance of the i -th feature. Then, the algorithms extract communities that are structurally dense, well separated from the rest of the graph, and consistent with the focus attributes. Cores of the communities are nodes that have high weighted similarity to their neighbor. Afterward,

communities are expanded by adding the node that increases the quality of the community the most if it exists. This quality is measured by conductance [Andersen et al., 2006] but could be measured with modularity [Newman, 2006] too. At the same time, the method detects outliers as nodes that structurally belong to a community but deviate in some focus attributes. Best structural nodes are identified during the expansion of the community as nodes that would be included to the community disregarding the attributes. Finally, outliers are the best structural nodes that are not in the final community: they should be in the community if attributes did not matter but they are not in the end. The authors conducted experiments to control the quality of their community structure, evaluate their outlier detection technique and its scalability.

An extension of Autopart [Chakrabarti, 2004] to the case of attributed graphs has been proposed by [Akoglu et al., 2012]. But, there is a much more performing method that deals with the same kind of graph and anomalies. Recently, [Perozzi and Akoglu, 2018] proposed a method for detecting both communities of an attributed graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, X)$ and anomalies as abnormal communities. This method is part of the community-based methods. It can uncover communities of the graph and extract local information for each community. The authors introduce a new measure of subgraph quality for attributed communities called normality and a community extraction algorithm that maximizes this normality. From the modularity, one can derive the internal consistency I of a cluster C as:

$$I = \sum_{v_i \in C, v_j \in C, i \neq j} \left(A_{ij} - \frac{d_i d_j}{2m} \right) \text{sim}_w(\mathbf{x}_i, \mathbf{x}_j | w) \quad (2.24)$$

where $\text{sim}_w(\mathbf{x}_i, \mathbf{x}_j | w)$ is a weighted node similarity computed thanks to \mathbf{w} a non-negative weight vector, d_i is the degree of v_i , m is the total number of edges and A is the adjacency matrix. The internal consistency is a measure of the quality of the community with regards to its features. For a community to be well-defined, it must be also well-separated from the rest of the graph. The external separability ES of a community C is the measure that controls it.

$$ES = - \sum_{v_i \in C, v_b \in \mathcal{B}, (v_i, v_b) \in \mathcal{E}} \left(1 - \min \left(1, \frac{d_i d_b}{2m} \right) \right) \text{sim}_w(\mathbf{x}_i, \mathbf{x}_j | w) \leq 0 \quad (2.25)$$

where \mathcal{B} is the set of nodes at the boundary of C . Finally, the normality of a community is defined as the sum of the internal consistency and the external separability.

$$N = I + ES \quad (2.26)$$

Then, the method infers the attribute weight vector w_c for a specific community C by maximizing the normality score such that nodes in the community are very similar and nodes at the boundary are very dissimilar. Finally, it is possible to detect outliers simply defined as communities whose normality is low. The authors compare their method with some other standard measures, like conductance [Andersen et al., 2006], Oddball [Akoglu et al., 2010] or

cut ratio [Yang and Leskovec, 2015], and show that their method consistently outperforms all other methods by roughly 20% in terms of average precision.

2.1.2.3 Deep learning methods for anomaly detection.

With the rise of deep learning methods and graph analysis, some methods belonging to this family have been proposed to tackle the issue of graph anomaly detection.

NetWalk [Yu et al., 2018] is a deep embedding approach that deals with dynamic graphs. It belongs to the family of probabilistic-based models. This method aims at finding anomalous nodes by learning a representation of the nodes at each time step and clustering them into groups such that nodes that do not belong to a cluster are spotted as anomalies. The representation is learned by clique embedding. First, a collection Ω of unbiased random-walks of size l is computed that is used as input of their novel embedding method. They propose a deep autoencoder neural network architecture to learn the vector representation while minimizing the pairwise distance among the vertices of a walk. The autoencoder ψ is a multi-layer perceptron of n_l layers that takes as input a node v_i encoded as a one-hot vector $\mathbb{1}_i$. The first half of the layers make the encoder $\psi^{(n_l/2)}$ while the remaining half makes the decoder $\psi^{(n_l)}$. The loss function L has three parts: a reconstruction error, the minimization of the pairwise distances, and a regularization term.

$$L = \frac{\gamma}{2} \sum_{k=1}^{|\Omega|} \sum_{i=1}^l \|\psi^{(n_l)}(v_i) - \mathbb{1}_i\|_2^2 + \sum_{k=1}^{|\Omega|} \sum_{1 \leq i, j \leq l} \|\psi^{(n_l/2)}(v_i) - \psi^{(n_l/2)}(v_j)\|_2^2 + \frac{\lambda}{2} \sum_{c=1}^{n_l} \|W^c\|_F^2 \quad (2.27)$$

where W^c are the weights learned at layer c . Thus, a representation of the nodes can be learned. It is then possible to cluster these vectors into groups. Finally, newly arriving nodes can be detected as anomalous if their representation does not fall into a cluster. It is also possible to rank the nodes with the distance to their closest centroid of cluster.

For their part, [Wang et al., 2021] proposed a One-Class graph neural network to detect anomalies in attributed graphs. This method also belongs to the family of probabilistic-based models. The main idea of one-class classification is to cluster the regular data into a single class or group and the remaining data is considered as outliers. The framework OCGNN, for One-Class Graph Neural Network, relies on the use of Graph Neural Network that will be extensively discussed next. The framework is described in Figure 2.7.

The algorithm takes as input an attributed graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, X)$ with several nodes labeled as normal nodes. The overall functioning of a graph neural layer can be summed up as:

$$H^{c+1} = f(H^c, A, W^c), H^0 = X \quad (2.28)$$

where $c \in \{1, \dots, n_l\}$ is the index of the layer, H^c is the output of this layer, A is the adjacency matrix and W^c are the weights learned. The full network is composed of several graph neural

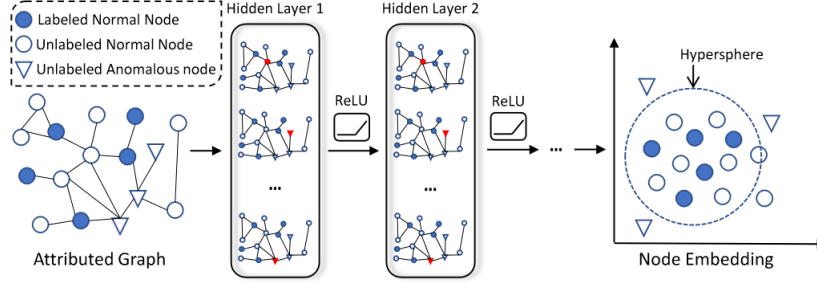


Figure 2.7: General workflow of the OCGNN framework. The method embeds the graph into a low-dimension space. Then it learns an hypersphere to cluster the regular data [Wang et al., 2021].

layers. Then, to learn the hypersphere, the authors propose a loss function that simultaneously learns the parameters of the network and minimizes the volume of the data description hypersphere which is characterized by a radius $\mathbf{r} \in R^+$ and a center $\mathbf{c} \in R^f$:

$$L(\mathbf{r}, W) = \frac{1}{\beta |\mathcal{V}_{training}|} \sum_{v_i \in \mathcal{V}_{training}} (||\mathbf{h}_i^{n_l} - \mathbf{c}||_2^2 - \mathbf{r}^2) + \mathbf{r}^2 + \frac{\lambda}{2} \sum_{c=1}^{n_l} ||W^c||^2 \quad (2.29)$$

where $\mathcal{V}_{training}$ is the set of training nodes, n_l is the total number of layers and β and λ are hyper-parameters. The first term is a penalty for node that stands out of the hypersphere. At this step, the embedding of a node v_i is the output of the final layer $H_i^{n_l}$, and we know the hypersphere. Thus, it is possible to detect anomalous nodes either as those that are very far from the center of the hypersphere or simply those nodes that are outside the sphere.

To conclude, a lot of methods have been proposed to tackle the issue of graph anomaly detection. While it is possible to classify the different methods according to the kind of graphs they require as input, it is harder to classify the kinds of anomalies. Often, the type of anomalies detected depends on the model itself [Yu et al., 2018] [Wang et al., 2021] [Chakrabarti, 2004] [Perozzi and Akoglu, 2018]. Clearly defined anomalies, such as bridges between communities or contextual anomalies, are rare. In the next section, we will discuss the detection of contextual anomalies.

2.2 Problem definition

[Chandola et al., 2009] defines contextual anomalies for vector data as: "if a data instance is anomalous in a specific context (but not otherwise), then it is termed as contextual anomaly". In their case, the contextual attributes are "used to determine the context". We introduce the same concept of anomalies but for relational data. The context is not only some specific features but an entire attributed subgraph around a focus node.

More formally, let $\mathcal{G}(\mathcal{V}, \mathcal{E}, X)$ be a graph on a set of n nodes $\mathcal{V} = \{v_i\}_{i=0}^{n-1}$, a set of edges $\mathcal{E} = \{e_{ij}\}$ and a feature matrix $X \in \mathbf{R}^{n \times f}$ such that each row \mathbf{x}_i of the matrix X is the feature vector of the node v_i .

Hypothesis 1 (Contextual anomaly). *We consider that there exists an unknown small subgraph (a context), and a distinguished node in this subgraph, such that each time this subgraph occurs in the data, then the node corresponding to the distinguished node is an anomaly.*

Example 2.2.1. *Some examples of contextual anomalies and their contexts are given in Figure 2.8. In these examples, the features of the nodes are colors. The context C_1 describes a node with no specific color that has at least one neighbor that is blue and one neighbor that is red. The context C_2 is the same except that the second neighbor should be green. Finally, the context C_3 describes a node that is yellow and that has at least one blue neighbor.*

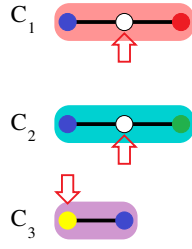


Figure 2.8: Three different types of contextual anomalies and their context. C_1 describes a node with no specific feature that has at least one blue neighbor and at least one red neighbor. C_2 is the same with blue and green. C_3 depicts a node that is yellow and has at least one blue neighbor. Contexts can be defined on the nodes themselves and in their vicinity.

We deal with the task of anomaly detection as a node classification task in a transductive setting as illustrated in Figure 2.9: the data consists of a single graph for which a proportion of the nodes is labeled (either "anomaly" or "normal") and the other nodes have no labels. The goal is to find the labels of the unlabeled nodes.

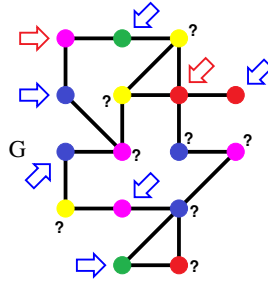


Figure 2.9: In a semi-supervised setting, some nodes are tagged as anomalies (red arrows), some nodes are tagged as normal (blue arrows) and the remaining nodes (with question marks) have to be classified into the normal class or the anomaly class.

However and most importantly, the conditions (i.e., the context) which make a node anomalous are **not known** during the training of the model.

It may be noted that our approach is related to the subgraph matching problem of [Scarselli et al., 2009] that can be defined as follows. Given a pattern graph P , all nodes belonging to a subgraph isomorphic to P in the graph database are labeled 1, the others are labeled -1 . In

our case, only one particular node of the pattern is an anomaly. This formalization seems more natural given the supervised information that is available: it is possible to know if a company or individual has been convicted/found guilty of corruption for instance, but it would be difficult to have the ground truth about the suspicious subgraph/context. We will see in Chapter 3 how to explain the reasons for which an element has been classified as anomalous. But, first, in the next section, we present the method that we designed to detect these anomalous nodes.

2.3 Our method: CoBaGAD

The main idea of our method is to learn simultaneously two two-classes classifiers with attention mechanisms. Then, local information is aggregated to determine whether a node is normal or not. Parameters of the network are learned with a standard classification loss in a semi-supervised fashion. For this, we propose to improve Graph Attention Networks (GAT)[Veličković et al., 2018] and propose CoBaGAD, for **C**ontext-**B**ased **G**raph **A**nomaly **D**etector, which has four steps detailed below.

The input of the model is a graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, X)$ where X is the feature matrix such that \mathbf{x}_i is the feature vector describing node v_i .

Global affine transformation: The first step is an affine transformation followed by a non linear function σ . This function σ is applied elementwise. The parameters are the matrix $W \in \mathbb{R}^{f \times f'}$ and a row vector $\mathbf{b} \in \mathbb{R}^{1 \times f'}$. The matrix with only 1 is denoted $\mathbb{1}$.

$$\Lambda = \sigma(XW + \mathbb{1}\mathbf{b}) \quad (2.30)$$

This step transforms the initial features X independently of the graph structure. The i -th row of the matrix Λ is the new representation for node v_i , and notice that since the matrix W is not necessarily square, this new representation can have more or fewer features than \mathbf{x}_i . Basically, this step embeds the features into a low-dimension space where $f' < f$. This allows a faster computation for the next steps and improves the performance of the classification.

Attention layer: It consists of k attention heads where k is a hyper-parameter. For each attention head $c \in \{0, \dots, k-1\}$, we perform a local linear transformation followed by a weighted aggregate. First, a local linear transformation $W_c \in \mathbb{R}^{f' \times f'}$ is applied on the features:

$$\Lambda_c = \Lambda W_c \quad (2.31)$$

Then, for each edge $(i, j) \in \mathcal{E}$, the value $e_{i,j,c}$ is computed:

$$e_{i,j,c} = \text{LeakyReLU}((\lambda_{i,c} \odot \lambda_{j,c}) \cdot \mathbf{u}_c) \quad (2.32)$$

where \odot is the Hadamard product, $\lambda_{i,c}$ and $\lambda_{j,c}$ are respectively the i th and j th rows of Λ_c , and $\mathbf{u}_c \in \mathbb{R}^{f'}$ is a learned column vector. The Hadamard product $\lambda_{i,c} \odot \lambda_{j,c}$ computes a similarity

vector between $\lambda_{i,c}$ and $\lambda_{j,c}$. When it is multiplied by \mathbf{u}_c , it corresponds to a weighted dot-product similarity where each element of \mathbf{u}_c is the relative importance of each corresponding element of the Hadamard product.

Then, the attention weights $\alpha_{i,j,c}$ are defined as a normalized version of $e_{i,j,c}$ such that for each node v_i and each head c , they are positive and sum to 1:

$$\alpha_{i,j,c} = \frac{\exp(e_{i,j,c})}{\sum_{k \in \mathcal{N}(v_i)} \exp(e_{i,k,c})} \quad (2.33)$$

where $\mathcal{N}(v_i)$ is the set of the neighbors of node v_i . The set $\mathcal{N}(v_i)$ may contain the node v_i itself. If so, we say that we add *self-loops* to the graph. Adding self-loops provides the information of the node v_i in its future representation.

The next step consists of computing, for each node v_i , a convex combination of the $\lambda_{j,c}$ for all neighbors v_j of v_i using the attention weights. These weights can be seen as the amount of information that flows between nodes or the importance of a message passing from v_j to v_i . Finally, the new representation \mathbf{h}_i of the node v_i is given by the concatenation of all the representations given by the k attention heads. i.e., each node v_i is represented by a row vector of $f' \times k$ features.

$$\mathbf{h}_{i,c} = \sum_{j \in \mathcal{N}(v_i)} \alpha_{i,j,c} \lambda_{j,c} \text{ and } \mathbf{h}_i = \sigma' \left(\parallel_{c=0}^{k-1} \mathbf{h}_{i,c} \right) \quad (2.34)$$

where σ' is an activation function, *softmax* in the case of the last layer or *ReLU* for the other layers, and \parallel is vector concatenation.

The network can be a stack of several such attention layers, the output of each layer being the input of the next layer.

The parameters that must be learned are W , \mathbf{b} , and for each of the k attention heads in each attention layer: the matrix W_c and the vector u_c . The hyper-parameters are f' , the number of attention layers, and the number k of heads in each attention layer. The activation functions σ and σ' can also be chosen by the user, basically they are *ReLU* or *softmax* functions.

CoBaGAD differs from the original GAT by two major points. First, we added a global affine transformation (Eq.2.30) to embed the original features in a new space. This operation improves the ability to detect anomalies and allows reducing the dimension of the problem. Then, we use a custom attention mechanism in Eq.2.32. Rather than concatenating the new representations of a pair of nodes, we compute a similarity between them with the Hadamard product. The attribute weight vector \mathbf{u}_c focuses on the most important part of this product for later inference.

Two-class classification: In the case of anomaly detection, we classify the nodes into two classes: anomalies and normal nodes. We design our neural network with one layer described above and $k = 2$ attention heads. We choose two attention heads since we want that the first one focuses on anomalies and the second one focuses on normal nodes. To do so, we set the hidden dimension $f' = 2$. Thus, the output of the network is a vector of dimension

4 after concatenation. We force the first head to recognize anomalies and the second head to recognize normal nodes by choosing carefully the design of the ground-truth vectors. Anomalies are represented by the vector $(1, 0, 0, 0)$ while the normal nodes are represented by $(0, 0, 0, 1)$. Thus, a high value on the first coordinate of the first attention head will be the sign of an anomaly and a high value on the second coordinate of the second head indicates that the node should be normal. Reciprocally, a high value on the second coordinate of the first head indicates that the node may be normal while a high value on the first coordinate of the second head indicates an anomaly. To sum up, coordinate 0 and 2 are indicative of an anomaly while 1 and 3 indicate a normal node. In the end, the parity of the maximum coordinate of the output of the network indicates whether a node should be classified as an anomaly or a normal node.

To showcase the detection power of our method, we conducted extensive experiments on many different datasets. The datasets depend on the kind of graph that is used and the type of anomaly defined by a context. A key issue is that such datasets are most of the time not publicly available. Thus, we proposed a generator to create some datasets with contextual anomalies.

2.4 Datasets generation

2.4.1 Generation of a graph with contextual anomalies

As benchmarks corresponding to the kind of anomaly considered in this chapter are not publicly available, to experimentally evaluate our model and compare it with the state of the art, we have designed a generator to automatically introduce context-based anomalies in a graph.

Starting from a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ and a context, our **Contextual Anomaly Generator**, ConA-Gen, transforms this graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ into an attributed graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, X)$ with contextual anomalies. First, we choose a graph without any feature on the nodes. Then, we add features to each node. There are many ways to create features for the nodes. The first step consists of choosing between categorical and numerical attributes. Attributes correspond to discrete categories like, for example, the city someone lives in. Numerical attributes are attributes whose values are numbers in a continuous space. For example, the age of a person is a numerical attribute, and "being young" is a categorical attribute. Thus, categorical attributes can be drawn from a finite set while numerical attributes are usually drawn from a continuous distribution. Each node is given a feature vector. Therefore, it is possible to mix the different attributes for each coordinate of the vectors. The easiest way to generate feature vectors is the 1 dimensional categorical case where each node is given a category from a small finite set of categories. For example, each node is given one color at random among five possible colors. Much more complicated feature vectors can be designed with multi-dimensional numerical attributes drawn from complex distributions for instance. Finally, nodes are tagged as anomalies or normal nodes depending on

their local context.

The generating process can be summed up as follows:

- Choose a graph and a context for anomalies
- Draw features from a distribution
- Tag nodes as anomalies if their neighborhood exhibits the context

Of course, during the generation of a dataset with contextual anomalies, the context that makes a node an anomaly is known. This is necessary to label the nodes as anomalous or normal. These labels may be used by an anomaly detector afterward.

2.4.2 An illustrative example

Assume a simple example where the input of ConAGen is the small graph drawn on Figure 2.10. The features of the nodes will be randomly drawn in a finite set of categorical attributes $\{\text{blue (B), green (G), red (R), yellow (Y), purple (P)}\}$ where each node can only have one color. The context to define an anomaly is simple too. It is the fact that a node is yellow. In the following, it will be denoted as the anomaly type A_3 .

Next, we are going to detail the process of generating the dataset in the case of a simple graph with context A_3 . The first step of the dataset generation is to choose a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ without any feature on the nodes, for example, the graph of Figure 2.10.

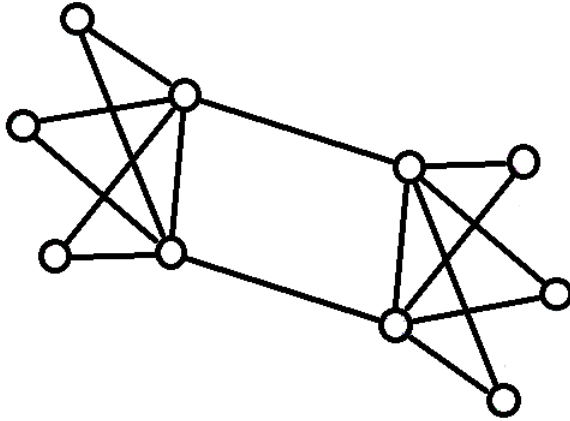


Figure 2.10: The first step of the dataset generation process: choose a graph with no features.

The second step consists of attributing a feature vector to each node of the graph. To do so, we choose one-hot vectors of size f describing the f different colors. For instance, the color blue (B) is equivalent to $(1, 0, 0, 0, 0)$ and yellow (Y) is equivalent to $(0, 0, 0, 1, 0)$. In this example, the features are categorical attributes that we interpret as colors but could be interpreted as

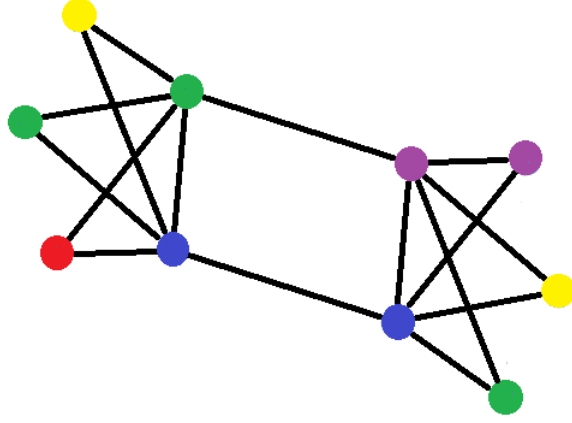


Figure 2.11: The second step of the dataset generation process: randomly draw feature vectors for each node of the graph.

any discrete categorical features. In the end, each node is attributed a one-hot feature vector randomly with uniform probability among the different colors as in Figure 2.11.

The final step is the labelization of the nodes. Depending on the chosen context, nodes are tagged as anomalies or normal nodes as in Figure 2.12.

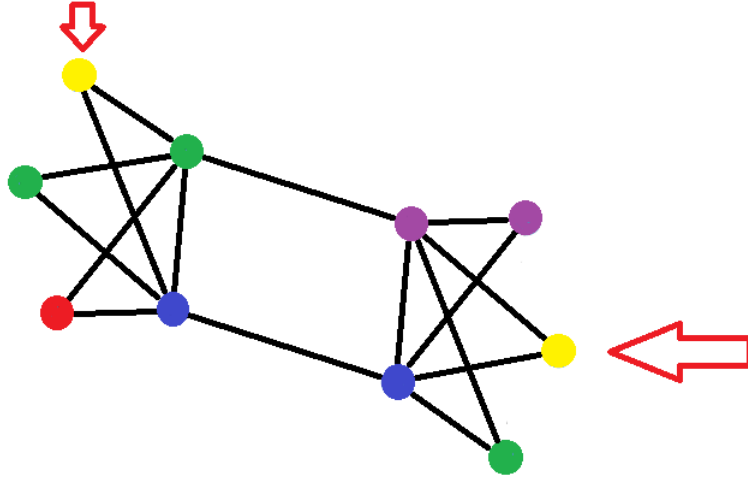


Figure 2.12: The final step of the dataset generation process: flag nodes that are anomalies.

2.5 Experimental evaluation of CoBaGAD

To experimentally evaluate the ability of CoBaGAD to identify anomalies, we use ConAGen to generate different datasets with different contexts, and then, we compare its performances with other state-of-the-art methods.

2.5.1 Datasets

In our experiments, we choose six graphs with very different structures and seven types of contexts. Table 2.1 gives the name, the number of nodes, and the number of edges of these graphs. G_0 is a random Erdos-Renyi graph. G_1 and G_4 are also generated, respectively with Dancer [Larger et al., 2017] and LFR [Lancichinetti et al., 2008], which mimic real-world graph behavior. Moreover, we chose real-world graphs that are common in the literature: Polblogs ¹, Cora ² and Facebook ³.








Graph	Name	Nodes	Edges	Anomalies	Definition	Context subgraphs
G_0	Erdos-Renyi	10000	24907	A_0	$B \wedge G$	
G_1	Dancer	10000	189886	A_1	$(B \wedge G) \vee (B \wedge R)$	
G_2	Facebook	4039	88234	A_2	$(B \wedge G) \vee (Y \wedge R)$	
G_3	Polblogs	1224	16715	A_3	\bar{Y}	
G_4	LFR	1000	5622	A_4	$\bar{Y} \wedge B$	
G_5	Cora	1433	5429	A_5	$Y \wedge \bar{Y} \wedge B$	
				A_6	$\bar{Y} \vee (Y \wedge B)$	

Table 2.1: Left table: Datasets characteristics: name of the graphs, number of nodes and edges. Right table: Anomalies characteristics. B = blue, G = green, R = red, Y = yellow, P = purple. \bar{Y} means that anomalies are nodes that have color Y . B means that anomalies have at least one neighbor whose color is B .

Then, the feature vector \mathbf{x}_i of node v_i is defined as a one-hot vector of dimension f . In our illustrative example related to fraud detection, such features can be interpreted as the role in a company (*e.g.* (0, 0, 1, 0, 0) represents the CEO and (0, 1, 0, 0, 0) represents an employee). In the following, to simplify the explanation, the one-hot vectors are equivalent to colors respectively: blue (B), green (G), red (R), yellow (Y), and purple (P). Indeed, every node of the graph is attributed a one-hot vector that can be interpreted as to its color or any categorical attribute.

Finally, we choose a context that will describe the anomalies. Among the nodes of the graph, some of them are flagged as anomalies if they follow a simple rule described by this context. In our experiments, we studied the rules presented in Table 2.1 but other rules can be considered. In this table, \bar{Y} means that anomalies are nodes that have color *Yellow* whereas B means that anomalies have at least one neighbor whose color is *Blue*. For example, $A_6 = \bar{Y} \vee (Y \wedge B)$ represents nodes that have the color yellow (Y) or have at least one neighbor with color blue (B) and at least one neighbor with color yellow (Y). By doing so, there are a lot of anomalies in the graph. It is not an issue in its current form but we want to address the problem of anomaly detection in the framework of imbalanced data since anomalies are, by definition, rare.

¹<http://konect.cc/networks/dimacs10-polblogs/>

²<https://relational.fit.cvut.cz/dataset/CORA>

³<https://snap.stanford.edu/data/egonets-Facebook.html>

To reduce their number and to be in an imbalanced setup that is closer to real-world cases, we change the color of some nodes. The color purple (P) which corresponds to the feature vector $(0, 0, 0, 0, 1)$ will never be used in the definition of the local contexts (see Table 2.1). Thus, we use it to replace the features of other nodes to have only between 4% and 6% of anomalies in the graphs. We could still reduce this number to control the level of imbalance but lower numbers of anomalies result in fewer instances to learn a machine learning model. Some of the graphs we use in our experiments only have 1000 nodes which gives roughly 50 anomalies. As explained later, only half of them are used to train the model, that is 25 examples. In the end, we cannot reduce this number as there would not be enough data to train. On the other hand, it would be possible to study cases where there are fewer anomalies (as a percentage of the total nodes) if we were to use bigger graphs.

Note that our algorithm CoBaGAD does not know how the anomalies have been created. Indeed, in a real-world case, the expert would flag some nodes as anomalies but he does not necessarily know the conditions which make a node anomalous. Thus the goal of our algorithm is to recognize these anomalies without this contextual knowledge.

2.5.2 Experimental setup

Thanks to our generator, ConAGen, all the nodes of the graphs belong to either the set of anomalies or the set of normal nodes. In a transductive setup, nodes are split into train, validation, and test sets. The train set is made of 50% of the total anomalies. Then, we add as many normal nodes as there are anomalies. The same applies to the validation set with 25% of anomalies. The test set is composed of the remaining 25% of anomalies and 25% of normal nodes of the graph. Balancing train and validation sets to have as many normal nodes as anomalies improved a lot the results. Thus a part of negative examples (normal nodes) is ignored during training. To ensure the reproducibility of our results, the code and the datasets are available in our GitHub ⁴.

We compare our algorithm, CoBaGAD, with state-of-the-art methods in node classification: Graph Convolution Networks [Kipf and Welling, 2017] (GCN), Graph Attention Networks [Veličković et al., 2018] (GAT), GraphSAGE [Hamilton et al., 2017] with mean aggregator and an unsupervised anomaly detection approach based on Node2vec [Grover and Leskovec, 2016] and LOF [Breunig et al., 2000].

For every deep learning method, we learn a single layer: we experimentally show that it is the best parameter. Given the kind of studied pattern, adding more layers seems not relevant. For CoBaGAD, we use GELU [Hendrycks and Gimpel, 2016] as activation function σ and softmax as activation function σ' , f' and $k = 2$ are set to 2 as we learn two 2-classes classifiers for classifying both anomalies and normal nodes. For GAT and GraphSAGE, we

⁴<https://github.com/vaudaine/Detection-of-contextual-anomalies-in-attributed-graphs>

use the same parameters. For GCN, we use *localpool* filter, softmax as activation function and output of dimension 2. For every algorithm, we tried two versions: without self-loop and with self-loops by adding the identity matrix to the adjacency matrix and we present the best results. We train for 1000 epochs with Adam optimizer [Kingma and Ba, 2015] and a learning rate of $5e - 3$ on the train set and validate it at each step. Thanks to early stopping, the weights of the networks are kept when the accuracy on the validation set is the highest. We use the standard categorical cross-entropy loss for k classes and n instances of the data: $L(Y^{true}, Y^{pred}) = - \sum_{j=1}^k \sum_{i=0}^{n-1} (y_{ij}^{true} \times \log(y_{ij}^{pred}))$.

Concerning the unsupervised approach, it is an association between Node2vec embedding and LOF anomaly detection. First, we compute an embedding of the graph using Node2vec with $p = 1, q = 1$, and dimension 128. It outputs a new representation for every node v_i . This representation is concatenated with its feature vector \vec{x}_i . Finally, these vectors are fed to a LOF classifier to detect anomalies.

2.5.3 Results

For each model, each dataset, and each type of anomaly, experiments are conducted 12 times by changing the train/validation/test split. We choose the 3 best results on the validation and report the mean and standard deviation of the precision or the recall obtained on the test set. The experiments have been conducted on the six different graphs from G_0 to G_5 and on the seven anomaly types from A_0 to A_6 for a total of 42 experiments. Only the precision of the anomaly class is reported. The recall of both anomalies and normal nodes and the precision of normal nodes are not discriminating between the methods. They are given in Annexes 3.7. Among these 42 results, CoBaGAD outperforms every other state-of-the-art method 33 times in terms of precision of the detection of the anomaly class. These results can be found in the following tables from 2.2 to 2.8 for each anomaly type.

A_0 / Graph	G_0	G_1	G_2	G_3	G_4	G_5
CoBaGAD	0.98 ± 0.03	0.96 ± 0.03	0.87 ± 0.09	0.56 ± 0.16	0.85 ± 0.13	0.92 ± 0.12
CoBaGAD loops	0.76 ± 0.05	0.89 ± 0.14	0.95 ± 0.03	0.87 ± 0.09	0.22 ± 0.1	0.69 ± 0.07
GAT	0.96 ± 0.04	0.5 ± 0.24	0.29 ± 0.02	0.55 ± 0.08	0.59 ± 0.13	0.93 ± 0.1
GAT loops	0.68 ± 0.04	0.83 ± 0.02	0.33 ± 0.08	0.43 ± 0.09	0.18 ± 0.04	0.59 ± 0.02
GCN	0.34 ± 0.02	0.11 ± 0.0	0.1 ± 0.0	0.18 ± 0.02	0.23 ± 0.02	0.26 ± 0.02
GCN loops	0.24 ± 0.0	0.12 ± 0.01	0.11 ± 0.0	0.16 ± 0.02	0.31 ± 0.03	0.14 ± 0.01
GraphSAGE	0.51 ± 0.03	0.51 ± 0.03	0.63 ± 0.05	0.44 ± 0.08	0.34 ± 0.05	0.52 ± 0.03
GraphSAGE loops	0.53 ± 0.01	0.56 ± 0.03	0.54 ± 0.12	0.41 ± 0.05	0.36 ± 0.08	0.5 ± 0.05
Naive	0.06 ± 0.01	0.06 ± 0.01	0.06 ± 0.02	0.16 ± 0.09	0.05 ± 0.04	0.03 ± 0.01

Table 2.2: Precision of the anomalies for A_0 on several graphs (G_0 - G_5) in the testing set.

The first group of anomaly types is $A_0 = B \wedge G$, $A_1 = (B \wedge G) \vee (B \wedge R)$ and $A_2 = (B \wedge G) \vee (Y \wedge R)$. For these anomaly types, the context of the node is entirely defined by its neighborhood. The information necessary to know whether a node is an anomaly is located

A_1 / Graph	G_0	G_1	G_2	G_3	G_4	G_5
CoBaGAD	0.96 ± 0.03	0.92 ± 0.11	0.78 ± 0.29	0.69 ± 0.22	0.72 ± 0.15	0.85 ± 0.12
CoBaGAD loops	0.72 ± 0.03	0.98 ± 0.02	0.78 ± 0.1	0.73 ± 0.05	0.44 ± 0.22	0.69 ± 0.02
GAT	0.74 ± 0.3	0.55 ± 0.21	0.51 ± 0.14	0.63 ± 0.07	0.46 ± 0.19	0.8 ± 0.18
GAT loops	0.71 ± 0.01	0.4 ± 0.06	0.33 ± 0.03	0.61 ± 0.12	0.25 ± 0.09	0.54 ± 0.04
GCN	0.34 ± 0.02	0.11 ± 0.01	0.12 ± 0.02	0.19 ± 0.03	0.32 ± 0.02	0.26 ± 0.01
GCN loops	0.24 ± 0.01	0.19 ± 0.03	0.13 ± 0.02	0.18 ± 0.02	0.34 ± 0.04	0.16 ± 0.01
GraphSAGE	0.46 ± 0.04	0.49 ± 0.04	0.51 ± 0.06	0.39 ± 0.06	0.38 ± 0.07	0.47 ± 0.04
GraphSAGE loops	0.41 ± 0.02	0.5 ± 0.02	0.49 ± 0.03	0.38 ± 0.08	0.31 ± 0.07	0.47 ± 0.03
Naive	0.05 ± 0.0	0.05 ± 0.01	0.09 ± 0.01	0.18 ± 0.04	0.0 ± 0.0	0.01 ± 0.01

Table 2.3: Precision of the anomalies for A_1 on several graphs (G_0 - G_5) in the testing set.

A_2 / Graph	G_0	G_1	G_2	G_3	G_4	G_5
CoBaGAD	0.61 ± 0.04	0.61 ± 0.02	0.58 ± 0.2	0.47 ± 0.14	0.64 ± 0.15	0.72 ± 0.06
CoBaGAD loops	0.46 ± 0.01	0.62 ± 0.06	0.34 ± 0.01	0.35 ± 0.15	0.26 ± 0.14	0.48 ± 0.06
GAT	0.52 ± 0.02	0.41 ± 0.04	0.29 ± 0.03	0.32 ± 0.07	0.3 ± 0.03	0.51 ± 0.11
GAT loops	0.43 ± 0.02	0.42 ± 0.1	0.25 ± 0.01	0.35 ± 0.06	0.18 ± 0.03	0.49 ± 0.03
GCN	0.27 ± 0.02	0.12 ± 0.0	0.13 ± 0.01	0.17 ± 0.01	0.25 ± 0.02	0.23 ± 0.03
GCN loops	0.16 ± 0.01	0.11 ± 0.0	0.14 ± 0.01	0.2 ± 0.04	0.19 ± 0.04	0.14 ± 0.0
GraphSAGE	0.33 ± 0.01	0.37 ± 0.01	0.43 ± 0.04	0.44 ± 0.05	0.27 ± 0.05	0.38 ± 0.01
GraphSAGE loops	0.32 ± 0.03	0.38 ± 0.02	0.44 ± 0.03	0.41 ± 0.04	0.33 ± 0.03	0.32 ± 0.04
Naive	0.03 ± 0.01	0.05 ± 0.02	0.08 ± 0.01	0.1 ± 0.01	0.03 ± 0.03	0.01 ± 0.01

Table 2.4: Precision of the anomalies for A_2 on several graphs (G_0 - G_5) in the testing set.

entirely on the set of neighbors of this node. Thus, no information from the node itself is needed. This means that, when aggregating the message from all neighboring nodes in Equation 2.34, it is not mandatory to take into account the node itself. In other words, the additional self-loops in the adjacency matrix should not add useful information.

From Tables 2.2, 2.3 and 2.4, we can see that CoBaGAD consistently outperforms every other state-of-the-art methods for 17 out of 18 experiments with an average precision of 0.78 ± 0.15 among all graphs and those three kinds of anomaly. In comparison, GAT has an average precision of 0.57 ± 0.20 , GCN has 0.22 ± 0.08 , GraphSAGE has 0.44 ± 0.08 and the naive approach has an average precision of 0.06 ± 0.05 . Moreover, we do not necessarily expect the experiments without self-loops to perform better as additional information from the loops could be put aside by the algorithm. The results show that CoBaGAD has better results without self-loops for 12 out of the 18 experiments which confirm that self-loops are not mandatory for the detection of these three types of anomalies even though it sometimes improves the performance of the algorithm.

The second group of anomaly types is $A_3 = \bar{Y}$, $A_4 = \bar{Y} \wedge B$, $A_5 = Y \wedge \bar{Y} \wedge B$ and $A_6 = \bar{Y} \vee (Y \wedge B)$. For these anomaly types, the context of the node is entirely defined by its neighborhood and the node itself. Thus, it is mandatory to have information from the node itself to be able to classify the node correctly. Thus, we expect the methods with self-loops to

A_3 / Graph	G_0	G_1	G_2	G_3	G_4	G_5
CoBaGAD	0.18 ± 0.0	0.27 ± 0.0	0.44 ± 0.05	0.21 ± 0.01	0.33 ± 0.08	0.13 ± 0.0
CoBaGAD loops	0.99 ± 0.01	0.99 ± 0.01	0.95 ± 0.05	0.91 ± 0.13	0.89 ± 0.15	0.93 ± 0.02
GAT	0.17 ± 0.01	0.36 ± 0.16	0.45 ± 0.12	0.22 ± 0.06	0.37 ± 0.23	0.09 ± 0.02
GAT loops	0.82 ± 0.07	0.79 ± 0.03	0.78 ± 0.07	0.55 ± 0.03	0.43 ± 0.01	0.61 ± 0.09
GCN	0.07 ± 0.0	0.06 ± 0.0	0.06 ± 0.0	0.06 ± 0.0	0.07 ± 0.0	0.06 ± 0.0
GCN loops	0.95 ± 0.02	0.97 ± 0.02	0.91 ± 0.07	0.8 ± 0.21	0.94 ± 0.08	0.78 ± 0.04
GraphSAGE	1.0 ± 0.0	0.95 ± 0.04	0.97 ± 0.04	0.71 ± 0.1	0.65 ± 0.29	0.8 ± 0.11
GraphSAGE loops	0.93 ± 0.08	0.99 ± 0.01	0.84 ± 0.09	0.66 ± 0.1	0.68 ± 0.24	0.97 ± 0.03
Naive	0.21 ± 0.01	0.06 ± 0.01	0.17 ± 0.01	0.0 ± 0.0	0.17 ± 0.06	0.04 ± 0.02

Table 2.5: Precision of the anomalies for A_3 on several graphs (G_0 - G_5) in the testing set.

A_4 / Graph	G_0	G_1	G_2	G_3	G_4	G_5
CoBaGAD	0.88 ± 0.09	0.9 ± 0.03	0.97 ± 0.04	0.8 ± 0.07	0.56 ± 0.12	0.57 ± 0.08
CoBaGAD loops	0.9 ± 0.1	0.95 ± 0.04	0.88 ± 0.05	0.61 ± 0.03	0.62 ± 0.1	0.89 ± 0.09
GAT	0.44 ± 0.03	0.72 ± 0.02	0.77 ± 0.03	0.47 ± 0.17	0.51 ± 0.1	0.33 ± 0.04
GAT loops	0.5 ± 0.07	0.77 ± 0.15	0.58 ± 0.12	0.67 ± 0.23	0.45 ± 0.1	0.43 ± 0.05
GCN	0.12 ± 0.0	0.08 ± 0.0	0.07 ± 0.01	0.06 ± 0.02	0.08 ± 0.01	0.12 ± 0.02
GCN loops	0.44 ± 0.01	0.7 ± 0.03	0.71 ± 0.12	0.6 ± 0.12	0.65 ± 0.07	0.35 ± 0.02
GraphSAGE	0.46 ± 0.03	0.73 ± 0.04	0.69 ± 0.03	0.58 ± 0.11	0.61 ± 0.04	0.41 ± 0.03
GraphSAGE loops	0.45 ± 0.02	0.73 ± 0.02	0.72 ± 0.02	0.53 ± 0.11	0.49 ± 0.06	0.36 ± 0.03
Naive	0.08 ± 0.0	0.09 ± 0.01	0.15 ± 0.01	0.0 ± 0.0	0.09 ± 0.01	0.02 ± 0.01

Table 2.6: Precision of the anomalies for A_4 on several graphs (G_0 - G_5) in the testing set.

A_5 / Graph	G_0	G_1	G_2	G_3	G_4	G_5
CoBaGAD	0.84 ± 0.02	0.9 ± 0.03	0.82 ± 0.07	0.61 ± 0.03	0.51 ± 0.23	0.84 ± 0.22
CoBaGAD loops	0.56 ± 0.04	0.8 ± 0.12	0.75 ± 0.07	0.48 ± 0.15	0.49 ± 0.03	0.5 ± 0.01
GAT	0.69 ± 0.11	0.74 ± 0.11	0.54 ± 0.09	0.57 ± 0.07	0.52 ± 0.09	0.46 ± 0.1
GAT loops	0.4 ± 0.06	0.74 ± 0.07	0.71 ± 0.09	0.5 ± 0.16	0.41 ± 0.01	0.38 ± 0.06
GCN	0.11 ± 0.01	0.08 ± 0.0	0.07 ± 0.01	0.07 ± 0.0	0.08 ± 0.01	0.1 ± 0.01
GCN loops	0.35 ± 0.01	0.67 ± 0.04	0.62 ± 0.07	0.6 ± 0.04	0.46 ± 0.13	0.23 ± 0.02
GraphSAGE	0.32 ± 0.02	0.68 ± 0.0	0.68 ± 0.06	0.47 ± 0.02	0.56 ± 0.06	0.26 ± 0.05
GraphSAGE loops	0.34 ± 0.01	0.68 ± 0.03	0.65 ± 0.06	0.51 ± 0.04	0.5 ± 0.04	0.3 ± 0.04
Naive	0.04 ± 0.0	0.06 ± 0.01	0.13 ± 0.02	0.0 ± 0.0	0.05 ± 0.04	0.04 ± 0.03

Table 2.7: Precision of the anomalies for A_5 on several graphs (G_0 - G_5) in the testing set.

perform much better than the ones without self-loops.

From Tables 2.5, 2.6, 2.7 and 2.8, we can see that CoBaGAD still consistently outperforms every other state-of-the-art methods for 17 out of 24 experiments with an average precision of 0.79 ± 0.19 among all graphs and those four kinds of anomaly. In comparison, GAT has an average precision of 0.58 ± 0.15 , GCN has 0.57 ± 0.24 , GraphSAGE has 0.63 ± 0.19 and the naive approach has an average precision of 0.08 ± 0.06 . Moreover, we do expect the experiments with self-loops to perform better as additional information from the loops is mandatory. The

A_6 / Graph	G_0	G_1	G_2	G_3	G_4	G_5
CoBaGAD	0.66 ± 0.3	0.37 ± 0.12	0.5 ± 0.18	0.4 ± 0.03	0.25 ± 0.08	0.27 ± 0.15
CoBaGAD loops	0.9 ± 0.01	0.89 ± 0.05	0.54 ± 0.27	0.62 ± 0.2	0.73 ± 0.08	0.4 ± 0.04
GAT	0.14 ± 0.02	0.34 ± 0.08	0.31 ± 0.02	0.4 ± 0.03	0.22 ± 0.07	0.16 ± 0.07
GAT loops	0.55 ± 0.12	0.59 ± 0.12	0.39 ± 0.05	0.48 ± 0.1	0.27 ± 0.03	0.39 ± 0.03
GCN	0.13 ± 0.02	0.12 ± 0.01	0.13 ± 0.01	0.15 ± 0.03	0.19 ± 0.01	0.11 ± 0.03
GCN loops	0.38 ± 0.05	0.26 ± 0.09	0.2 ± 0.07	0.21 ± 0.01	0.32 ± 0.13	0.46 ± 0.11
GraphSAGE	0.67 ± 0.04	0.58 ± 0.05	0.44 ± 0.03	0.35 ± 0.06	0.3 ± 0.08	0.64 ± 0.06
GraphSAGE loops	0.65 ± 0.03	0.54 ± 0.04	0.48 ± 0.06	0.49 ± 0.16	0.48 ± 0.06	0.65 ± 0.14
Naive	0.18 ± 0.02	0.11 ± 0.01	0.12 ± 0.01	0.1 ± 0.05	0.1 ± 0.04	0.06 ± 0.01

Table 2.8: Precision of the anomalies for A_6 on several graphs (G_0 - G_5) in the testing set.

results show that CoBaGAD has better results with self-loops for 16 out of the 24 experiments which confirms that self-loops are mandatory for the detection of these types of anomalies.

To sum up, the results show that our algorithm achieves state-of-the-art performance across all datasets and anomalies. More specifically, for A_0 , A_1 and A_2 which all are anomalies based on the pattern $B \wedge G$, our method always outperforms the other competitors (except for A_0 , G_5 where it is still very relevant). We are able to improve upon GAT, our principal contender, by at least 2% on A_0 , G_0 up to 62% on A_0 , G_2 . Attention-based methods are better than the others (GCN, GraphSage, Node2vec + LOF) when dealing with these types of anomalies. Anomalies A_3 to A_6 rely on the pattern \bar{Y} which means that the considered nodes are yellow. Thus, information about the node itself is required. A_3 is a very simple pattern where anomalies are defined by the simplest pattern: nodes are just yellow. In that case, we can suppose that it is easy for many algorithms to perform well in detecting those nodes. In fact, GraphSAGE shows good performance for most of the graphs but lacks a bit of consistency. GCN is more consistent but results are worse than those provided by GraphSAGE. While GAT fails to show good performance, our method is the most consistent and shows very good results in general. Then, for A_4 to A_6 , as the pattern becomes more complex, CoBaGAD remains the only method that, except a few cases, correctly detects the anomalies.

2.5.4 Number of layers: hyper-parameter tuning

The key parameter in our method is the number of layers of the graph neural network. A layer aggregates information from the set of neighbors of a node as described in Equation 2.34. Thus, a network with two layers will aggregate information from the 2-hop neighborhood which is the set of neighbors of the neighbors of a node. Accordingly, a neural network made of C layers will aggregate information from nodes at distance C . While more information is necessary in some cases, algorithms tend to be less accurate when given too much senseless data. To verify this, we test one hypothesis:

- Anomalies defined by a context at distance C are better detected with a neural network with C layers.

This involves using networks with more layers to increase the "field of view" and also studying an anomaly defined by a context of larger diameter. Thus, we introduce the anomaly A_7 defined by the context: if a node v_i has at least one blue neighbor v_j that has at least one blue neighbor v_k which is not v_i , then v_i is an anomaly. Thus, information at distance two is necessary to detect anomalies of type A_7 . To test our hypothesis, we conducted the exact same experiments as in the previous section but, this time, we change the number of layers. Intermediate layers have a *ReLU* activation while the final one has a *softmax*. We still do the experiments twelve times, find the best three based on the validation set and average their results on the test set. The results for G_4 and G_5 for every type of anomaly are given in Tables 2.9 and 2.10

G_4	A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7
1 Layer	0.85 ± 0.13	0.72 ± 0.15	0.64 ± 0.15	0.89 ± 0.15	0.62 ± 0.10	0.51 ± 0.23	0.73 ± 0.08	0.48 ± 0.00
2 Layers	0.50 ± 0.25	0.31 ± 0.07	0.23 ± 0.03	0.85 ± 0.03	0.61 ± 0.17	0.44 ± 0.07	0.46 ± 0.06	0.39 ± 0.01
3 Layers	0.31 ± 0.03	0.28 ± 0.03	0.42 ± 0.20	0.77 ± 0.16	0.22 ± 0.04	0.23 ± 0.02	0.39 ± 0.01	0.41 ± 0.08

Table 2.9: Precision of the detection of anomalies A_0 - A_7 on the graph G_4 in the testing set.

G_5	A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7
1 Layer	0.92 ± 0.12	0.85 ± 0.12	0.72 ± 0.06	0.93 ± 0.02	0.89 ± 0.09	0.84 ± 0.22	0.40 ± 0.04	0.29 ± 0.03
2 Layers	0.31 ± 0.06	0.27 ± 0.04	0.18 ± 0.03	0.72 ± 0.20	0.63 ± 0.18	0.55 ± 0.18	0.15 ± 0.04	0.14 ± 0.02
3 Layers	0.26 ± 0.05	0.29 ± 0.08	0.22 ± 0.09	0.29 ± 0.09	0.46 ± 0.14	0.45 ± 0.12	0.22 ± 0.06	0.33 ± 0.14

Table 2.10: Precision of the detection of anomalies A_0 - A_7 on the graph G_5 in the testing set.

What we expect from the results is that the 1 layer case is better for all anomalies except A_7 where the 2 layer case should be better. Results show that for every anomaly A_0 to A_6 whose context only needs information at distance 1, the model with only 1 layer is the best every time. As for A_7 , where information at distance 2 is necessary, we see that the 1 layer model is better in the case of G_4 while the 3 layer model is better for G_5 . In the end, our hypothesis is partially confirmed. For contexts at distance 1, 1 layer is enough. Whereas for contexts at a larger distance, more experiments have to be conducted to better understand the impact of the number of layers on the performances.

2.6 Conclusion

Throughout this chapter, we have defined a new kind of graph anomaly based on a context. Such anomalies follow a simple pattern. We also proposed a generator to create datasets with contextual anomalies that we use to conduct our experiments. Then, we have presented a Context-Based Graph Anomaly Detector, CoBaGAD, an extension of the Graph Attention Networks that focuses on detecting contextual anomalies. With intensive transductive experiments, we demonstrate the ability of our method to identify such anomalies and to outperform state-of-the-art algorithms.

Different improvements can be addressed in future work such as scoring anomalies instead of binary classifying. We would like also to address the case of numerical attributes or at least

more complex features when generating our datasets instead of categorical attributes like colors. Another particularly interesting field of research in the domain of anomaly detection is the explainability of the detected anomalies. The objective is to be able to recover the context that defines an anomaly. That is the topic of the next chapter.

Chapter 3

Explaining anomaly classification in graphs.

Graph mining has been radically changed by machine learning in the last few years. Community detection, node classification, or link prediction are a few of the many tasks that have been revisited and whose results have been significantly improved. Even though machine learning led to successful enhancements, it has brought its black-box models whose decisions are not well human-understandable. Understanding the predictions of a model can be very beneficial to trust the model itself. Nowadays, models are usually evaluated by accuracy metrics on available data but their performances in real-world cases can be very different. In addition, some models are very sensitive to small modifications of the training data [Mannino et al., 2009]. Thus, understanding single predictions can lead to an overall better model and can avoid errors when using the model. Moreover, the lack of transparency of black-box models makes them not very acceptable when they are responsible for important decisions in our daily life, for instance for health applications or for fraud detection where the identification of an anomaly can lead to a legal proceeding that can not be launched only on a probability score.

In this chapter, we present a new model to learn an explanation for node classification prediction in the context of anomaly detection in attributed graphs. In the specific case of anomaly detection, nodes of the anomalous class are not necessarily densely connected in the graph. It is notably true for contextual anomalies, studied in this chapter, where anomaly depends on their neighborhood. This kind of anomaly is relatively frequent in practice and, usually not easy to detect. If black-box models can detect them efficiently, they are not able to justify the reasons for which they have assigned the label "anomalous" to a particular node.

To overcome this limit, we propose to explain their prediction by learning a new understandable model for every instance identified as anomalous. More precisely, in a LIME-like fashion [Ribeiro et al., 2016], we perturb the neighborhood of the node to create new instances relatively similar to this node. Then, the black-box classifier is used to predict the class of those perturbed instances. Finally, using this new dataset, composed of the perturbed instances with their class predicted by the black-box model, a new local and interpretable classifier is learned

that tells whether the perturbation changed the node’s class or not. Thus, the local classifier allows explaining the decisions of the global black-box model.

In this chapter, we present:

- A new framework to explain the predictions of black-box models in the context of anomaly detection in attributed graphs,
- The definition of measures to quantify the effectiveness of an explanation,
- An experimental validation of our method on several kinds of graphs that confirms its interest.

The chapter is organized as follows. First, we review related works. We define the problem studied in Section 3.2 and present our method to explain contextual anomalies in attributed graphs in Section 3.3. Then, we describe our evaluation protocol in Section 3.4 and the experiments carried out to evaluate the ability of our method to explain the decisions taken by a black-box classification model in Section 3.5. Finally, we discuss the obtained results and compare them to those provided by state-of-the-art methods in Section 3.6.

3.1 Related work

3.1.1 Explainability in the context of vector data

Recent approaches, based on deep learning have proved to be particularly effective for solving many tasks. However, due to the "black-box" effect, they also suffer from a lack of understandability which limits their use. The interpretability of the models, i.e. their understanding by the AI specialists but also their explainability, i.e. the understanding by the end-user of the reasons that led the algorithm to make a decision, have become real societal issues notably for applications in many areas such as medicine or criminal justice [Arrieta et al., 2020]. This led the European Union to introduce a right to explanation in General Data Protection Right (GDPR). Different methods have been proposed to tackle the issue of explaining a black-box prediction. According to [Arrieta et al., 2020], predictive models are either transparent, *i.e.* self explanatory like decision trees, or need post-hoc explainability. Our study is within the scope of model-agnostic post-hoc explainability. This branch can be divided into several families: local explanations, visual explanations, feature relevance explanation, and explanation by simplification. Feature relevance explanation with sensitivity analysis and local explanations with additive feature attribution are the closest methods to our work.

3.1.1.1 Sensitivity analysis (SA)

The basic idea of sensitivity analysis is to look over all the space of features to evaluate the response of the model. SA quantifies how much an input of a model has to change to modify its

output. [Krause et al., 2016] provide interactive partial dependence diagnostics. In the context of risk management in medicine, the authors want to find how features affect the prediction of a model with partial dependence plots. The concept of partial dependence has been introduced by [Friedman, 2001]. This is a visualization tool to summarize the dependence of a function ϕ on the values of its input variables $\{\mathbf{x}_i\}_{i=1}^n$ where $\mathbf{x}_i \in \mathbb{R}^f$. It is easy to plot a function depending on only one or two variables but viewing functions of high-dimensional arguments is more difficult. To formally define the partial dependence, let $S \subset \{1, \dots, f\}$ and C be the complement of S . S and C are the index subsets. For example, if $S = 1, 2, 3, 4$, then $\mathbf{x}_{i,S}$ refers to the 4×1 vector containing the values of the first four coordinates of $\mathbf{x}_i = \mathbf{x}_{i,S} \cup \mathbf{x}_{i,C}$. It is possible to rewrite the output of the model as: $\phi(\mathbf{x}_i) = \phi_{\mathbf{x}_{i,C}}(\mathbf{x}_{i,S}) = \phi(\mathbf{x}_{i,S}|\mathbf{x}_{i,C})$. Then, the partial dependence is defined as:

$$\phi_S(\mathbf{x}_{i,S}) = \frac{1}{n} \sum_{j=1}^n \phi_{\mathbf{x}_{j,C}}(\mathbf{x}_{j,S}) \quad (3.1)$$

This partial dependence allows one to study the influence of a subset of features. It tells us for given values of S what the average marginal effect on the prediction is. For example, assume we learn a model ϕ to predict the risk of a person having cancer. It is possible to study the partial dependence of the model to feature "Age" by plotting its partial dependency plot. The set S corresponds to the singleton containing only the feature "Age". Then, for each value of "Age", it is possible to compute $\phi_S(\mathbf{x}_{i,S})$. To do so, the age of each instance of the data is modified to a specific value. Then, the model predicts a risk for each instance that is averaged. Finally, the process is repeated for every value of "Age". The partial dependence of the predicted risk with respect to the age of a population is given in Figure 3.1.

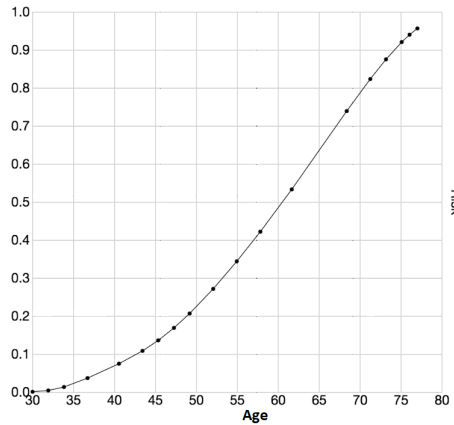


Figure 3.1: Partial dependence plot of the average predicted risk with respect to the age [Krause et al., 2016]

Thus, partial dependence plots give an insight into the dependence of a predictive model with regards to some specific features. These plots are very useful to have a first look at the dependence of the data. But more precise measures have also been developed.

For their part, [Cortez and Embrechts, 2011] proposed different measures of sensitivity analysis. SA allows assessing input relevance and effects on the model's responses. First, the data vectors $\mathbf{x}_i \in \mathbb{R}^f$ are aggregated into a baseline vector $\mathbf{b} \in \mathbb{R}^f$ composed by the mean attribute values where $b_j = \frac{1}{n} \sum_{i=1}^n x_{ij}$. Then, from the baseline vector \mathbf{b} each input varies through its range. For example, to study the influence of the j -th coordinate, its $1d$ space is divided into K regular subspaces from $\min_{i=1,\dots,n} x_{ij}$ to $\max_{i=1,\dots,n} x_{ij}$. Then, a new dataset \mathbf{z}^j is constructed where each $\mathbf{z}_k^j \in \mathbb{R}^f$ is recursively defined by:

$$\mathbf{z}_0^j = \left(b_1, b_2, \dots, \min_{i=1,\dots,n} x_{ij}, \dots, b_f \right) \quad (3.2)$$

$$\mathbf{z}_k^j = \mathbf{z}_{k-1}^j + \left(0, 0, \dots, \frac{\max_{i=1,\dots,n} x_{ij} - \min_{i=1,\dots,n} x_{ij}}{K}, \dots, 0 \right) \quad (3.3)$$

Then, the respective model responses are used to compute a sensitivity metric. Let us denote $\hat{z}_{k,j}$ the sensitivity response, *i.e.* the response of the model, for \mathbf{z}_k^j . The range (r) is defined as:

$$r_j = \max_{k=0,\dots,K} (\hat{z}_{k,j}) - \min_{k=0,\dots,K} (\hat{z}_{k,j}) \quad (3.4)$$

The gradient (g) is:

$$g_j = \sum_{k=2}^K |\hat{z}_{k,j} - \hat{z}_{k-1,j}| / (K - 1) \quad (3.5)$$

and the variance (v) is:

$$v_j = \sum_{k=2}^K (\hat{z}_{k,j} - \bar{\hat{z}}_{k,j})^2 / (K - 1) \quad (3.6)$$

where $\bar{\hat{z}}_{k,j}$ is the average of $\hat{z}_{k,j}$ over all K instances. Finally, the relative importance R_j of each metric can be computed:

$$R_j = s_j / \sum_{l=1}^f s_l \quad (3.7)$$

where s can be either r , g or v . These relative importance allows ranking the features to find the most influential one.

3.1.1.2 Inverse classification

Inverse classification is a form of local sensitivity analysis: it focuses on one instance at a time and consists of modifying one input of a black-box model to change its output to a specific class. The main issue is to find the minimal change that leads to the desired class such that this modification gives an explanation of the initial classification. [Laugel et al., 2018] proposed an inverse classification approach. They have two objectives with their explanatory method: the explanation must be accurate and human-understandable. They consider a binary classifier ϕ where the observation $\mathbf{x} \in \mathbb{R}^f$ must be interpreted with $\phi(\mathbf{x})$ its associated prediction. The goal is to explain \mathbf{x} with \mathbf{e} , another observation, belonging to the other class such that $\phi(\mathbf{x}) \neq \phi(\mathbf{e})$. The final explanation is $\mathbf{e} - \mathbf{x}$ which is the minimal modification to do to change the class of the

instance \mathbf{x} . Indeed, if a small change of a specific coordinate involves a change in the predicted class, then this feature is probably important. Conversely, if the predicted class changes due to a very large modification of a coordinate, then it is probably unimportant to the classification. Thus, they aim at solving the following problem:

$$\mathbf{e}^* = \underset{\phi(\mathbf{x}) \neq \phi(\mathbf{e})}{\operatorname{argmin}} c(\mathbf{x}, \mathbf{e}) \quad (3.8)$$

where $c(\mathbf{x}, \mathbf{e}) = \|\mathbf{x} - \mathbf{e}\|_2 + \gamma \|\mathbf{x} - \mathbf{e}\|_0$ with γ a weighting hyperparameter and $\|\mathbf{x}\|_0$ is the number of non-zero elements of \mathbf{x} . To find the element \mathbf{e}^* that minimizes this cost function, the authors use a growing sphere algorithms which consists of finding the minimal sphere layer SL around \mathbf{x} defined as:

$$SL(\mathbf{x}, a_0, a_1) = \{\mathbf{z} | a_0 \leq \|\mathbf{x} - \mathbf{z}\|_2 \leq a_1\} \quad (3.9)$$

This is done by changing the radius of this sphere such that, first there is no point of the other class in it. Then, the sphere is expanded to only have one instance of the other class. In the end, the explanation provided to the user is $\mathbf{e}^* - \mathbf{x}$.

3.1.1.3 Additive feature attribution methods

The model that we propose to explain the prediction of a black-box model belongs to this family.

Additive feature attribution methods use a linear function of binary variables as explanatory model [Lundberg and Lee, 2017] [Shrikumar et al., 2017]. An explanation must be faithful to the predictive model f , local, and interpretable. More specifically, let \mathbf{x}_0 be an input and $\phi(\mathbf{x}_0)$ the prediction to be explained where ϕ is the black-box model. These methods use simplified inputs \mathbf{x}' (for instance binary vectors) instead of \mathbf{x} and they define a local mapping around \mathbf{x}_0 , $h_{\mathbf{x}_0}$, from the simplified inputs to the original inputs such that $h_{\mathbf{x}_0}(\mathbf{x}'_0) = \mathbf{x}_0$ (\mathbf{x}'_0 is the simplified version of \mathbf{x}_0). Then a local explanatory model $g_{\mathbf{x}_0}$ is learned to try to preserve the same output as ϕ in the neighborhood of \mathbf{x}_0 : for all simplified inputs \mathbf{x}' such that $h_{\mathbf{x}_0}(\mathbf{x}')$ is "close" to \mathbf{x}_0 , $g_{\mathbf{x}_0}(\mathbf{x}') \approx \phi(h_{\mathbf{x}_0}(\mathbf{x}'))$. The faithfulness of the model g with respect to ϕ is the fact that g should find the same class as ϕ . Also, the model g is local since it approximates the model ϕ around a specific instance \mathbf{x}_0 . The different additive feature attribution methods differ by the choice of their mapping functions h , of their loss function and, of the kind of explanatory model g .

[Ribeiro et al., 2016] proposed LIME, for Local Interpretable Model-agnostic Explanations, an explanation technique that can explain the predictions of any classifier. LIME introduces a measure $\Omega(g)$ of the complexity of the model g such that this model is interpretable. For example, it can be the depth of a decision tree. Then, $\pi_{\mathbf{x}}(\mathbf{z})$ is a proximity measure between \mathbf{x} and \mathbf{z} , another instance of the data, to define locality around \mathbf{x} . Finally, the explanation model g for the instance \mathbf{x} must minimize both $\Omega(g)$ and $L(\phi, g, \pi_{\mathbf{x}})$ a fidelity function. To do so, they

sample instances \mathbf{z}' around \mathbf{x}' by drawing non-zero elements of \mathbf{x}' uniformly at random. Then, from a perturbed instance \mathbf{z}' , they recover the original representation \mathbf{z} and obtain $\phi(\mathbf{z})$. This set of recovered perturbed instances and their prediction can then be used to learn the model g . The authors promote the use of linear models and sparse linear explanations most specifically but any regression model could be used as long as it is interpretable. If we note M the number of simplified input features, according to [Lundberg and Lee, 2017]:

$$g(\mathbf{z}') = \psi_0 + \sum_{i=1}^M \psi_i \mathbf{z}'_i \quad (3.10)$$

where $\mathbf{z}' \in \{0, 1\}^M$ and $\psi_i \in \mathbb{R}$ are learned.

To sum up, the additive feature attribution methods consist of two steps. First, draw samples \mathbf{z} around an instance \mathbf{x} to explain. Then, learn a new model g with these samples \mathbf{z} .

There are many approaches to deal with explainability in the context of vector data. Among them, sensitivity analysis looks for the minimal change of input to modify its prediction by the machine learning model. On the other hand, additive feature attribution methods rely on the fact they perturb input instances to learn a local explanation model. These methods have proven to be efficient for vector data but, for graph data, other approaches exist.

3.1.2 Explainability in the context of relational data

Most of the works done in XAI (Explainable Artificial Intelligence) concerns tabular data but graph mining has also been revisited recently especially thanks to graph embedding techniques and graph neural networks and, it suffers also from the lack of ability to understand the process applied for classifying a node, predicting a link or detecting a community. A new taxonomy of the different methods has been proposed in [Yuan et al., 2021] which distinguishes:

- **Gradient-based** methods [Pope et al., 2019] come from image and text processing. The main idea is to differentiate the output of the model with respect to its input. This creates a heat-map that gives the feature relative importance. Higher gradients usually imply higher importance for the input features.
- **Perturbation** methods [Ying et al., 2019] [Luo et al., 2020] [Schlichtkrull et al., 2021] are based on perturbations, small changes, of the input of the black-box model. The aim is to study the change of output when modifying the input.

3.1.2.1 Gradient-based methods

Gradient-based methods [Pope et al., 2019] come from image and text processing. The underlying idea is to differentiate the output of the model with respect to its input. This creates a heat-map that gives the features relative importance. Higher gradients usually imply higher importance for the input features. These methods focus on Graph Convolutional Networks.

For an attributed graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, X)$ with adjacency matrix A , the matrix formulation of a GCN layer H^l is:

$$H^l(X, A) = \sigma(CH^{l-1}(X, A)W^l) \quad (3.11)$$

where σ is an activation function, W^l are the weights learned at layer l and $C = \hat{D}^{-1/2}\hat{A}\hat{D}^{1/2}$ is the convolution matrix with $\hat{d}_{ii} = \sum_j \hat{a}_{ij}$ and $\hat{A} = A + I_n$ is the adjacency matrix with added self-loops. Then, the authors define the k -th graph convolutional feature map at layer l as:

$$\mathbf{h}_k^l(X, A) = \sigma(CH^{l-1}(X, A)\mathbf{w}_k^l) \quad (3.12)$$

where \mathbf{w}_k^l is the k -th column of W^l . Thus, for the node v_i , the k -th feature at layer l is $h_{k,i}^l$. It is possible to calculate the global average pooling feature after the final layer L as:

$$e_k = \frac{1}{n} \sum_{i=1}^n h_{k,i}^L \quad (3.13)$$

and the class score, for class c , is $y^c = \sum_k w_k^c e_k$. Finally, different heat-maps can be computed. First, the gradient-based heat-map is:

$$L_{gradient}^c(v_i) = ||ReLU\left(\frac{\partial y^c}{\partial \mathbf{x}_i}\right)|| \quad (3.14)$$

Second, the Class Activation Mapping (CAM) is defined by:

$$L_{CAM}^c(v_i) = ReLU\left(\sum_k w_k^c h_{k,n}^L(X, A)\right) \quad (3.15)$$

As for GradCAM, an extension of CAM, the class c specific weights are calculated by:

$$\alpha_k^{c,l} = \frac{1}{n} \sum_{i=1}^n \frac{\partial y^c}{\partial h_{k,i}^l} \quad (3.16)$$

To then compute the heat-map with:

$$L_{Grad-CAM}^c(v_i) = ReLU\left(\sum_k \alpha_k^{c,l} h_{k,n}^l(X, A)\right) \quad (3.17)$$

All these heat-maps allow evaluating the relative importance of the features at each layer of the graph convolutional neural network. The authors conducted experiments to prove that they are able to identify relevant substructures of the graph for a given classification. These subgraphs are explanations of the classification.

3.1.2.2 Perturbation methods

GraphLIME [Huang et al., 2020] is a surrogate approach presented as an extension of the LIME model [Ribeiro et al., 2016] to the graph domain. The aim of the method is to explain the classification of a node v_i by the model ϕ , described by its feature vector \mathbf{x}_i . The general workflow of the method is the same as LIME. The first step consists of sampling around the

data to explain. The second step is the process of learning a new model that is faithful and interpretable. The sampling strategy is very simple: it samples the k -hop neighborhood $\mathcal{N}_k(v_i)$ around a node v_i . For each node $v_j \in \mathcal{N}_k(v_i)$, it is possible to know its predicted class $\hat{y}_j = \phi(v_j)$. Then, GraphLIME learns an interpretable model g thanks to the aggregated information of the neighborhood of v_i with the HSIC, Hilbert-Schmidt Independence Criterion, Lasso framework which is a non-linear feature selection method. To do so, it uses the features $\mathbf{x}_j \in \mathbb{R}^f$ associated with each node $v_j \in \mathcal{N}(v_i)$ as input and their predicted class $\hat{y}_j = \phi(x_j)$ as label. The goal of the HSIC Lasso is to find the set of positive parameters β_k that minimizes the following loss:

$$\frac{1}{2} \|\bar{L} - \sum_{k=1}^d \beta_k \bar{K}^{(k)}\|_F^2 + \rho \|\beta\|_1 \quad (3.18)$$

where \bar{L} is the normalized centered Gram matrix that comes from the \hat{y}_j , $\bar{K}^{(k)}$ is the normalized centered Gram matrix for the k -th feature that comes from the \mathbf{x}_j and ρ is a regularization parameter. The minimization is done via Least Angle Regression [Efron et al., 2004a]. In the end, the β_k are known and depict the relative importance of each feature. Finally, the most important features are the explanation of the classification. The key limitation of GraphLIME is that it cannot distinguish feature importance among neighbors. In particular, the features of the node v_i are inherently as important as those of its neighbors. In fact, GraphLIME outputs the relative importance of each of the f features which is the same for the node v_i and its neighbors.

On the other hand, GNNExplainer [Ying et al., 2019] is a perturbation method that jointly learns graph structural and node feature information to unveil what part of the graph is most relevant when a model ϕ predicts the class of node v_i . For this specific node v_i , the goal of GNNExplainer is to identify the subgraph \mathcal{G}_S and the set of features X_S that are important for the prediction $\hat{y}_i = \phi(v_i)$. The first step consists of finding \mathcal{G}_S . GNNExplainer formulates the problem as the maximization of the mutual information between the predicted class and the predicted class when a part of the graph or a part of the features is masked:

$$MI(\hat{Y}, (\mathcal{G}_S, X_S)) = H(\hat{Y}) - H(\hat{Y} | \mathcal{G} = \mathcal{G}_S, X = X_S) \quad (3.19)$$

where $H(\hat{Y})$ is the entropy of \hat{Y} and $H(\hat{Y} | \mathcal{G} = \mathcal{G}_S, X = X_S)$ is a conditional entropy where a part of the graph is masked. For a node v_i , MI quantifies the difference between the case where the full graph and the whole set of features are known with the case where only the subsets \mathcal{G}_S and X_S are known for training. The first term of the equation, $H(\hat{Y})$, is an entropy term that remains constant which is not taken into account for optimization purposes. The second term is the only one to be maximized. The second step of the method is the process of learning a feature selector for the nodes in \mathcal{G}_S . The algorithm only considers a subset of the features $X_S^F = X_S \odot F$ which is defined through a binary feature selector $F \in \{0, 1\}^f$. Thus, only several coordinates of the feature vectors will be selected. In the end, the following mutual information

objective is jointly optimized:

$$MI(\hat{Y}, (\mathcal{G}_S, F)) = H(\hat{Y}) - H(\hat{Y} | \mathcal{G} = \mathcal{G}_S, X = X_S^F) \quad (3.20)$$

Intuitively, the feature selector F acts as a mask. If a feature is not important in the classification process, then masking this feature should not modify a lot the final prediction. To conclude, GNNExplainer offers an explanation as to the set (\mathcal{G}_S, X_S) after optimization. This is a small amount of information needed to classify a specific node correctly. The explanation is the subset of nodes and features that are necessary for the model to predict precisely. A key limitation to this method is that the same mask is applied to the features of every node in the subgraph, this means it cannot highlight different features on different neighbor nodes. Thus, the authors chose to learn a uniform feature selector but this may not be suitable for many cases and, in particular, for contextual anomalies.

[Luo et al., 2020] proposed PGExplainer, a parameterized explainer for graph neural networks. PGExplainer focuses on explaining multiple instances collectively. It addresses this challenge with a deep neural network to parameterize the generation process of explanations. The main idea of this method is to learn a probabilistic graph generative model to provide the subgraph explaining a prediction. The general framework is the same as in GNNExplainer but focuses on the underlying computation graph. The algorithm aims at maximizing the following mutual information:

$$MI(\hat{Y}, \mathcal{G}_S) = H(\hat{Y}) - H(\hat{Y} | \mathcal{G} = \mathcal{G}_S) \quad (3.21)$$

where $\mathcal{G} = \mathcal{G}_S + \Delta\mathcal{G}$. \mathcal{G} is the original graph, \mathcal{G}_S is the subgraph explaining the predictions \hat{Y} and $\Delta\mathcal{G}$ is the part of the graph which does not contain important information. Then, the model collectively explains the predictions of a trained graph neural network on multiple instances. It uses a parameterized network to learn to generate explanations. The model consists of two steps. A vector representation is learned through a first layer GNN_0 :

$$Z = GNN_0(A, X) \quad (3.22)$$

then a classification layer GNN_1 is applied:

$$Y = GNN_1(Z) \quad (3.23)$$

Thus, Z will be used as input for the explanation network $\Sigma = g(G, Z)$. Several network g are proposed to learn an explanation. The authors promote the use of a multi-layer perceptron for both graph classification and node classification. In their experiments, they show that their method finds subgraphs that are more explanatory than GNNExplainer. But it relies on the output of a GNN. Thus, the information of the structure of the graph and the features are already mixed together. It is not possible to differentiate between them. Moreover, as GNNExplainer, PGExplainer lacks the opportunity to differentiate important features.

All the methods discussed previously have one major drawback. They are not able to distinguish the importance of features coming from different nodes. In particular, for the case of contextual anomalies, it is important to differentiate between the different nodes of a neighborhood. This led us to propose a new method dedicated to explaining the predictions of a graph neural network.

3.2 Problem definition

The problem studied in this chapter consists of explaining the prediction of a black-box model in the contextual anomaly detection setting. Thus, we assume that anomalous nodes have already been detected by a first global model and, we propose a local method to explain, in a human-understandable way, the reasons for which a specific node has been classified as anomalous. Before presenting our method, we introduce notations and notions used throughout this chapter.

3.2.1 Contextual anomaly detection

The detection of contextual anomalies has been extensively discussed in the previous chapter 2. We sum up its main components.

Formally, let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, X)$ be an attributed graph defined by a set of n nodes $\mathcal{V} = \{v_i\}$, a set of edges $\mathcal{E} = \{e_{ij}\}$ and a feature matrix $X \in \mathbb{R}^{n \times f}$ where f is the dimension of the feature vectors such that each row \mathbf{x}_i in matrix X is the feature vector describing the node v_i . Each node v_i of \mathcal{G} has a true label $y(v_i) \in \{1, 0\}$ where $y(v_i) = 1$ if v_i is anomalous and $y(v_i) = 0$ otherwise. Moreover, we suppose that the anomalies are contextual. This means that there exist several small subgraphs, called contexts, C_1, C_2, \dots and in each C_i there is a distinguished node. For each occurrence of C_i in \mathcal{G} , the node in G corresponding to this distinguished node in C_i is an anomaly. The aim of contextual anomaly detection is to learn a model that will classify unlabeled nodes into one class or the other. We do it with CoBaGAD and show that our model performs very well.

3.2.2 Classification problem formalization

We suppose that the contexts C_i are not known and that the anomalies are detected in a transductive setting: at training time, the labels of some of the nodes of \mathcal{G} are known and a classifier $\phi_{\mathcal{G}}$ is learned using this training set to detect which of the unlabeled nodes correspond to anomalies.

We assume that the output of the learning on this graph \mathcal{G} is a black-box classifier $\phi_{\mathcal{G}}$. Given a node v_i and a graph $N(v_i)$, $\phi_{\mathcal{G}}(v_i, N(v_i))$ provides the prediction for the label of v_i . The attributed graph $N(v) = (NV, NE, NX)$ denotes a neighborhood of v_i which is used by the model $\phi_{\mathcal{G}}$ to compute this prediction ($N(v)$ is also sometimes called the computation graph). It contains the nodes NV , edges NE and features NX of the nodes close to v_i in \mathcal{G} . Depending on

the class of models, $N(v)$ can be, e.g., empty or equal to the whole graph \mathcal{G} or a part of \mathcal{G} . For instance, when using GNNs, $N(v)$ is the k -hop neighborhood of v_i in \mathcal{G} where k is the number of layers in the GNN. In the rest of the chapter, to simplify the notations, $\phi_{\mathcal{G}}$ is denoted ϕ .

3.3 Our method

This section presents our method to find an explanation for the label $\phi(v_i, N(v_i))$ predicted by the black-box model ϕ for a given node v_i . Our strategy, inspired by LIME, is to first learn a simple local model g "around" v_i which is faithful (i.e., as good as ϕ locally) and interpretable (i.e., it will be possible to derive an explanation from g).

This means that a different model g is learned for each node identified as anomalous by the global black-box model ϕ and, usually g is simpler than ϕ : we do not expect that it can classify all nodes v_i of \mathcal{G} accurately. It will only need to be accurate on nodes v_j that are "close" in some sense to v_i . This closeness will not be relative to a distance between v_i and v_j in the graph \mathcal{G} as in GraphLIME. In our case, the nodes v_j will be copies of v_i with a randomly perturbed neighborhood described next.

We chose decision trees as the class of models for g . Indeed, they are easy to learn, understandable, and provide a way to evaluate the importance of each of the features they use to predict the class of a node but other types of models can also be used like for instance logistic regression.

An important interest of the proposed methodological framework is that, for a given context graph C_i , one can generate new training and testing data, as detailed below, and then compare the explanations produced by our method to the ground truth defined by C_i . Of course, the contexts C_i are not used in the learning phase (by the decision model) nor in the explanation phase (by the explanation model). They are just used as ground truth to compute the quality of the explanations, in terms of precision and recall, and to evaluate this explanation model using measures that we introduce in Section 3.4. That is clearly an advantage of our method which differs by this way from the other approaches where this ground truth is not available and thus the explanations are difficult to evaluate. Moreover, our method can distinguish the most important features depending on the node of interest. This is clearly an advantage compared to previous state-of-the-art methods.

3.3.1 Local sample as perturbations of the neighborhood

Given the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, X)$, a model ϕ which is a black-box model that predicts the class of the nodes and a node v_i whose predicted class has to be explained, the first step of the process consists of sampling the vicinity of the node v_i to generate training data for learning the local model g .

To be able to learn a faithful model g , the vicinity of v_i should consist of nodes that are similar to v_i in their features but also in their neighborhoods. The approach of GraphLIME

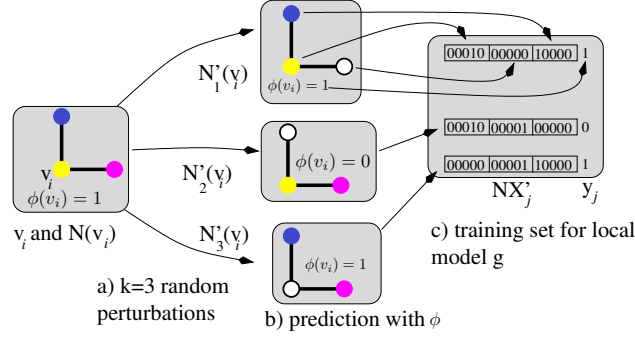


Figure 3.2: Generation of the training set for local model g . v_i is a node classified as an anomaly by the black-box model ϕ for which we want to compute an explanation. a) k perturbed neighborhoods are randomly generated (nodes represented in white have their features masked), b) ϕ is applied to label them, c) they are encoded as vectors to make the training set for g .

is to sample the neighbors of v_i in \mathcal{G} . But in our setting of contextual anomaly detection and because anomalies are rare, this method of sampling is more likely to give only normal nodes and, consequently, the model g learned on this sample will be constant and not useful.

Thus we propose another strategy closer to the masking technique of the original LIME and GNNexplainer: A perturbed neighborhood $N'(v_i) = (NV, NE, NX')$ is generated from $N(v_i) = (NV, NE, NX)$ by randomly masking the feature vectors of some nodes in $N(v_i)$ (which includes v_i itself).

Every node is masked with probability p . If it is selected, its feature vector in NX is replaced by a "void vector": two natural choices are the vector with only zeros or the vector with only $1/f$ where f is the number of features per node. It is not necessary to perturb the edges NE since our local models (decision trees) will only take as input the perturbed features NX' of the node v_i and its neighbors. This random perturbation is repeated k times to get a set of perturbed $\{NX'_j\}_{j=1}^k$. The higher k , the more data to learn g . This increases the ability of g to explain accurately but this also linearly increases the time needed to explain a prediction. For its part, p is the probability to mask the features of a node. The aim of a perturbation is to generate new instances relatively similar to v_i such that sometimes their predictions by ϕ change. Thus, low values of p will generate new data not too different from the instance that has to be explained. Typical values for p and k are 0.5 and 250.

Figure 3.2 shows an example where $k = 3$ perturbed neighborhoods are generated. In each case, the masked node (represented in white) has its features replaced by 0. It could also be $1/f$ as explained before. Here, each node feature vector is a one-hot representation of its color. Colors are ordered: blue, green, red, yellow, and purple such that for each node, its feature vector i -th element is a one if it has the corresponding color. Thus, the features of the yellow node v_i are $(0, 0, 0, 1, 0)$, they are $(1, 0, 0, 0, 0)$ for the blue node and $(0, 0, 0, 0, 1)$ for the purple one.

3.3.2 Learning the local model

The next step is to generate a learning set and learn the local model g on this set. The learning set contains the perturbed feature matrices $\{NX'_j\}_{j=1}^k$ but also the labels for these nodes.

Since we aim to explain the predictions of the black-box model ϕ , we use ϕ to compute these labels (see Fig. 3.2b) The complete training set with the associated labels is thus :

$$\{(NX'_j, y_j)\}_{j=1}^k \text{ where } y_j = \phi(v_i, (NV, NE, NX'_j)).$$

If d_i is the degree of v_i in the graph, each $\{NX'_j\}$ is a matrix of $(d_i + 1)$ rows (one for v_i and one for each of its neighbors) and f columns. From now on, we will see it as a vector of $(d_i + 1)f$ input features by concatenating the rows of the matrix. For example, Fig. 3.2c shows the generated training set for g . Each vector is a concatenation of the features of node v_j and its two neighbors and the corresponding label y_j computed with ϕ .

Remark that since the size of the vectors NX'_j depends on the degree of v_i , it would not be possible to train directly a decision tree on the whole graph. This decision tree can only be learned locally. As stated before, we chose to train a decision tree but any human-understandable model could be used. For a node v_i , a decision tree g is trained with the k different vectors $\{NX'_j\}_{j=1}^k$ as features and the corresponding y_j as labels (step d in Fig. 3.3). There is no test set for g as we are not interested in evaluating its accuracy. However, it is obvious that g is only learned if ϕ achieves good performances in terms of accuracy. Decision trees are human interpretable (at least if they are not too large). So we could stop here and output the tree as an explanation. However, we want to evaluate the importance of each feature in the decision and be able to compare it with the ground truth consisting of the context subgraph C_i .

3.3.3 Feature importance in a decision tree

To explain the decision of a black-box model ϕ , as presented in previous section, we train a binary decision tree g using the training set $\{(NX'_j, y_j)\}_{j=1}^k$.

At each step during the learning of g , *i.e.*, for each node T in the decision tree, a feature l is selected according to its predictive power evaluated thanks to an impurity function like for instance, entropy or Gini impurity. The information gain $IG(T)$ obtained at this step is then defined as the decrease of impurity resulting from the splitting of the node T in the decision tree according to feature l into its child nodes. Assuming only two child nodes ($left_T$ and $right_T$) without loss of generality, it is given by:

$$IG(T) = w_T imp_T - w_{left(T)} imp_{left(T)} - w_{right(T)} imp_{right(T)} \quad (3.24)$$

where w_T is the proportion of samples reaching node T and imp is the impurity function.

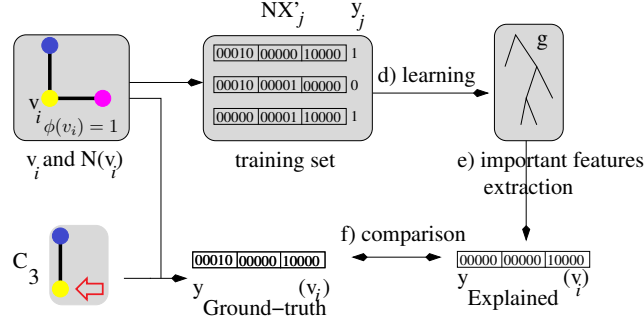


Figure 3.3: Explaining a prediction. d) From the training set (see Fig. 3.2), a local model g is learned. e) The important features are extracted to generate the explanation $y_{Explained}(v_i)$ which can be compared with the ground truth $y_{Ground-truth}(v_i)$. This ground truth is generated from $N(v_i)$ and the context (here we assume that the context C_3 generated the anomaly on v_i). In this example, $y_{Explained}(v_i)$ is different from $y_{Ground-truth}(v_i)$: the explanation only detected the blue neighbor but not the fact that the node v_i should be yellow.

Then, the global importance $f\text{-imp}(l)$ of an input feature l is computed as the ratio of the information gain obtained on the nodes in the decision tree where feature l has been retained:

$$f\text{-imp}(l) = \frac{\sum_{T \in J_l} \text{IG}(T)}{\sum_{k \in K} \text{IG}(k)} \quad (3.25)$$

where J_l is the set of nodes of the tree that split on feature l and K is the set of all nodes of the tree.

3.3.4 Explaining the prediction of a contextual anomaly by the black-box model

For a node v_i identified as anomalous by ϕ , the local model g , *i.e.* the decision tree, gives us the importance $f\text{-imp}(l)$ of each of those $(d_i + 1)f$ features, where d_i is the degree of the node v_i . We propose two scenarios to analyze and return the explanation features:

- most important features: if there are n_f features in the ground truth, select the n_f features l with the largest $f\text{-imp}(l)$
- all important features: select all the features l whose importance $f\text{-imp}(l)$ is more than 0.

We represent an explanation by a vector $y_{Explained}(v_i) \in \{0, 1\}^{(d_i+1)f}$ whose j -th element is 1 if the j -th feature has been selected and 0 otherwise (step e in Fig. 3.3).

The importance of a feature is related to the information gain that it provides and the number of times where it is used in the decision tree. Thus, a highly important feature is a feature that is expected to have a large impact on the output of g . Since g should predict the same labels as f on the local training set, the important features for g are also important (locally) for ϕ .

The fact that the explanation is local is also important: in our setting, the anomalies are defined by the occurrences of the context subgraphs C_i . Thus each anomaly in the graph \mathcal{G} is related to one particular context C_i (each C_i somehow defines a "type" of anomalies) and we need that our model can give a different explanation for each context.

It should be noted that this framework enables us to uncover not only the important features but also on which node they occur. In our illustrative example, we can know if the node has been classified because it has a specific color and/or because one of its neighbors (or more) have also a specific color (not necessarily the same) and which ones, both for the neighbors and the colors. For instance, in Fig. 3.3, $y_{Explained}(v_i)$ contains one 1, corresponding to the first feature of the second neighbor of v . This means that the model explains the anomaly by this particular feature of this particular neighbor. By the way, in this example, the model missed one other important feature explaining the anomaly as can be seen by comparing with $y_{Ground-truth}(v_i)$.

Finally, it is important to note that our framework can be used with any model f , such as Graph Neural Networks or Graph Embeddings, with any design of the features NX' describing the neighborhood of the node v_i and with any kind of human-understandable model g . In fact, we presented our model for contextual anomaly detection but our model can be used for any classification task on graphs with any model since the explanation model does not depend on the classification model itself but only on its output. Moreover, we use a decision tree to compute the importance of each feature but it could be also suitable to use any transparent model.

3.4 Measures: Fidelity, efficiency, and quality of an explanation

Different criteria can be evaluated when explaining node classification. First, fidelity is a measure of faithfulness. We can also evaluate the efficiency of an explanatory model depending on the ease of understanding and the time needed to give an explanation. Last, we can measure if the explanation is right by comparison with the ground truth. To take into account these different aspects, we defined three measures.

3.4.1 Fidelity

When locally approximating the model ϕ around the node v_i , the interpretable model g must at least be locally faithful to the black-box model ϕ . Thus, the predicted class of v_i must be the same for both models.

We can introduce a measure of fidelity by averaging over all nodes to be explained, the difference of prediction of class between ϕ and g :

$$Fidelity = 1 - \frac{1}{n_e} \sum_{v_i \in \mathcal{V}_{exp}} |\phi(v_i, N(v_i)) - g(NX_i)| \quad (3.26)$$

where \mathcal{V}_{exp} is the set of n_e nodes to be explained.

The fidelity measures the proportion of predictions that are equal between ϕ and the local model g . For a model g that reproduces well locally the predictions of ϕ , the fidelity should be close to 1. In the case where a local model g would not have a high enough fidelity, then it is pointless to use it to explain a prediction.

3.4.2 Efficiency

The efficiency of an explanation is very important. There are two main dimensions. First, the explanatory model g has to give a result that is easy to understand: the user should not have to spend a lot of time understanding the explanation. Second, the model g must not take a lot of time to give an explanation: the user should not wait too long to have an explanation. While the first is hard to measure since it would require humans to estimate the time they need to understand an explanation, the second is easier to evaluate. Indeed, we can compare the complexity of different models to have an idea of how long it would be to have an explanation as we will show in the experimental section.

3.4.3 Precision and recall

To evaluate the quality of an explanation $y_{Explained}(v_i)$, we need a ground truth. It is given by the definition of the context for the anomalous node. The ground truth is a vector $y_{Ground-truth}(v_i) \in \{0,1\}^{(d_i+1)f}$ (having the same size as $y_{Explained}(v_i)$) whose j -th element is 1 if stated so in the definition of the anomaly context.

For instance in Fig. 3.3, we consider that the node v_i is an anomaly because of context C_3 , i.e., because it is yellow and it has a blue neighbor. In graph \mathcal{G} , node v_i first neighbor is purple, and the second one is blue. Then, for this node, the ground truth related to C_3 is given by $y_{Ground-truth}(v_i) = (0, 0, 0, 1, 0; 0, 0, 0, 0, 0; 1, 0, 0, 0, 0)$ (the order of the colors in the one-hot representation is blue, green, red, yellow, purple). Indeed, the first 5 elements are the node v_i 's features since its color is in the context C_3 . Then, the purple neighbor does not appear in the context so we simply concatenate 5 zeros. Finally, the blue neighbor is in the context so we add its feature vector to the ground truth. In the end, $y_{Ground-truth}(v_i)$ is a natural encoding of the context that defines this anomaly on node v_i . It has exactly two 1s that correspond to the two important features that we want to recover: the fact that the node is yellow, and the fact that it has a blue neighbor.

In the case of a good explanation for the prediction of node v_i , we have $y_{Ground-truth}(v_i) = y_{Explained}(v_i)$ term by term. To measure to which extent an explanation is good, we compute the precision $Precision(v_i)$ and recall $Recall(v_i)$ of the 1 between $y_{Ground-truth}(v_i)$ and $y_{Explained}(v_i)$. In Fig. 3.3, $y_{Ground-truth}(v_i)$ contains two 1s and $y_{Explained}(v_i)$ recovered only one of them, so the recall is 50% and the precision is 100%.

Such measures are local, i.e. only for one node v_i . To have a global measure, we can average the precision and the recall over all anomalous nodes to be explained in a graph. Thus, we can

define the average precision and the average recall as:

$$P = \frac{1}{n_e} \sum_{v_i \in \mathcal{V}_{exp}} Precision(v_i) \quad (3.27)$$

$$R = \frac{1}{n_e} \sum_{v_i \in \mathcal{V}_{exp}} Recall(v_i) \quad (3.28)$$

where \mathcal{V}_{exp} is the set of n_e anomalous nodes to be explained.

3.5 Experiments

To show the ability of our methodological framework to explain individual predictions of a black-box model, we conducted experiments on many different datasets. These datasets as well as the code are publicly available in our anonymous GitHub repository ¹. The context of anomalies is natural for our framework. Indeed, we investigate the change of prediction of a node v_i by the model ϕ . Thus, a setup with only two classes is ideal. Many previous methods do not experiment on data with ground truth since, to our knowledge, there is no such dataset in the domain of explainable node classification. In our case, we can evaluate the performance of our method by conducting several experiments that show how the parameters of the model can be tuned.

3.5.1 Datasets

As datasets with contextual anomalies are not publicly available, we used the same methodology as in the previous chapter 2 with artificially introduced anomalies. In this way, we have been able to evaluate our explainable model and compare it with the state-of-the-art methods on many different graphs, real or synthetic. The graphs are the same as in the previous chapter. Datasets are created with our graph contextual anomaly generator ConAGen. Every experiment has been conducted on a machine with 1 CPU core at $2.5GHz$, 1 NVidia GTX 1080 GPU, and 20GB of RAM.

When explaining the classification of a node by the model ϕ , the final explanation should match the rule that defines the anomalies. Context subgraphs that define the anomalies that we use in our experiments are reminded in Table 3.1

3.5.2 Experimental setup

We retain as classifier ϕ for detecting the anomalies CoBaGAD since, as shown in the previous chapter, it is the best performing method to detect contextual anomalies which offer a natural ground truth for an explanation. Then, we compare the explanations given by our approach and two other contenders GNNExplainer [Ying et al., 2019] and GraphLIME [Huang et al.,

¹Code available in our repository: <https://github.com/Anonymous0001000/Explaining-anomaly-classification-in-graphs>










Anomalies	Definition	Context subgraphs
A_0	$B \wedge G$	
A_1	$(B \wedge G) \vee (B \wedge R)$	 / 
A_2	$(B \wedge G) \vee (Y \wedge R)$	 / 
A_3	\bar{Y}	
A_4	$\bar{Y} \wedge B$	
A_5	$Y \wedge \bar{Y} \wedge B$	
A_6	$\bar{Y} \vee (Y \wedge B)$	

Table 3.1: Anomalies characteristics. B = blue, G = green, R = red, Y = yellow, P = purple. \bar{Y} means that anomalies are nodes that have color Y . B means that anomalies have at least one neighbor whose color is B .

2020].

To learn the parameters of CoBaGAD, we split the datasets the exact same way and use the same parameters as in the previous chapter. Once we trained the black-box model ϕ , CoBaGAD in our case, we have to learn a binary decision tree g for each prediction of ϕ . We only explain nodes of the test set classified as anomalous by CoBaGAD. Their number can change depending on the precision of CoBaGAD for a specific experiment. The masking probability is $p = 0.1$, the number of features per node is $f = 5$ and the "void vector" is $(0, 0, 0, 0, 0)$. We make $k = 250$ perturbations of the graph for each explanation and use Gini impurity as a splitting criterion since the input features are boolean. The maximum depth of the tree is set to 3 to avoid over-fitting. In real cases, it could be useful to limit the depth of the tree to a higher value if a lot of features are expected to be important to explain the decision of the black-box model. Moreover, the input of the decision tree g is the set $\{(NX'_j, y_j)\}_{j=1}^k$ as defined previously and we choose the neighborhood of a node to be defined as the usual 1-hop neighbors.

The explanation vector of a node v_i is $y_{Explained}(v_i) \in \{0, 1\}^{(d_i+1)f}$ and the corresponding ground truth is a vector $y_{Ground-truth}(v_i) \in \{0, 1\}^{(d_i+1)f}$. To build the ground truth vector, we assume that the first f elements of these vectors are on the features of the node itself and the $d_i f$ last elements are about the features of its neighbors as defined in the section 3.4.

For its part, GraphLIME outputs the importance $importance_{GraphLIME} \in \mathbb{R}^f$ of each of our f features per node (the same for the node v_i and its neighborhood). The algorithm is used with parameters $hop = 1$ and $rho = 0.1$. As there is no distinction between the node and its neighbors, we compute a vector $y_{explained-GraphLIME}$ as the concatenation of $d_i + 1$ times $importance_{GraphLIME}$. For GNNExplainer, we train the model with 100 epochs for each explanation. It outputs the importance $importance_{GNNExplainer} \in \mathbb{R}^f$ of each of our f features per node and the subgraph of important nodes in the vicinity of v_i . Thus, an explanation is a vec-

tor $y_{\text{explained-GNNExplainer}}$ given by the concatenation of the vector $\text{importance}_{\text{GNNExplainer}}$ when the corresponding node belongs in the important subgraph and the vector $(0, 0, 0, 0, 0)$ otherwise.

3.6 Results

We conducted experiments on the six graphs and the seven contexts of anomalies described previously. We compare our method with GNNExplainer [Ying et al., 2019] and GraphLIME [Huang et al., 2020] and report the results for the three measures defined in Section 3.4.

3.6.1 Fidelity

The fidelity of the explanatory models g evaluates their ability to classify nodes the same way as the black-box model ϕ . We compared our approach with GraphLIME and GNNExplainer. We do not report the results here since all the methods have a fidelity equal to 1 for every dataset and every context.

3.6.2 Efficiency

As explained previously, efficiency can be divided into two parts. First, an explanation has to be easy to understand by the user. While it is hard to measure the ease of understanding, we can still compare the shape of the explanations given by the different methods. For a node v_i and its neighborhood, GraphLIME outputs the importance of each feature regardless of whether it is the studied node or its neighbors. In our case, we have $f = 5$ features per node in our graphs and GraphLIME can find important features among those. GNNExplainer, for its part, gives the importance of each of those $f = 5$ features and also gives the subgraph of the neighborhood that leads the model ϕ to classify the node v_i into a specific class. Our method can find the importance of each feature while distinguishing between the node v_i itself and its neighbors. All those three types of explanations are easy to understand. However, our method is more precise compared to GraphLIME and GNNExplainer. Indeed, as stated in the related work section, these last ones have only one feature mask for all neighbors and thus, they are not able to explain for which neighbor which feature is important.

The second part of the efficiency is the time needed to have an explanation of a prediction. We can compare the time complexity of each method. Let us denote $\text{call}(\phi)$ the time that the model ϕ needs to predict the class of a node v_i , d_i the degree of this node v_i and $\text{depth}(T)$ the maximum depth of our decision trees. When explaining the prediction for a node v_i , the time complexity of our method is as follows. First, we sample k different perturbations for a complexity of $O(k \cdot d_i \cdot f)$. Then, we call k times the model ϕ to predict the class of v_i with a complexity of $O(k \cdot \text{call}(\phi))$. Finally, we build the decision tree T whose complexity is $O(k \cdot \text{depth}(T) \cdot d_i \cdot f)$. To sum up, our time complexity is $O(k \cdot d_i \cdot f + k \cdot \text{call}(\phi) + k \cdot \text{depth}(T) \cdot d_i \cdot f) = O(k(\text{call}(\phi) + d_i \cdot f(1 + \text{depth}(T))))$. For its part, GraphLIME method can be decomposed into

several steps. First, it samples the neighborhood of the node v_i with complexity $O(d_i)$ at distance 1. Then, for each of these neighbors, the model ϕ has to predict their class for a total time of $O(d_i \cdot \text{call}(f))$. The final step is the learning of the HSIC Lasso parameters. The computation of the kernel and Gram matrices take $O(d_i^2 \cdot f)$ and the parameters are learned with the Lasso Lars algorithm whose time complexity is not explicit. In the end, the total time complexity of GraphLIME is $O(d_i \cdot \text{call}(\phi) + d_i^2 \cdot f + \text{Lasso_Lars})$. GNNExplainer does not provide a precise analysis of its time complexity but the learning is linear in the number of epochs chosen by the user. In the end, the comparison is tough but all the methods are relatively fast.

3.6.3 Quality of the explanation

Our method as well as the contenders, GNNExplainer and GraphLIME, output the importance of each feature. The main difference between them is their ability to accurately explain the predictions of the model ϕ . Thanks to our methodological framework, the quality of an explanation that they provide can be evaluated according to precision and recall as defined in equations 3.27 and 3.28. Averaged scores of precision (P) and recall (R) (\pm the standard deviation) obtained by each method are therefore computed on the anomalies correctly identified by the classifier ϕ (CoBaGad) for every graph and every type of contextual anomalies A_0 to A_6 . Table 3.2 shows the results if all the features whose importance is non-zero are retained as important features to explain the classification while Table 3.3 presents the results when choosing only the most important features as explanations, as indicated in Section 3.4. Bold numbers indicate the best score for each dataset and type of contextual anomaly.

Results show that our method largely outperforms the other two approaches in both scenarios ('all important features' and 'most important features'). Globally speaking, our method performs the best in 42 cases out of 42 experiments in terms of precision for both 'all important features' and 'most important features' and outperforms the other methods in both average precision and average recall as shown in Table 3.4. In terms of precision, it obtains very high scores, often equal to 1, which means that the features returned to the user are all correct. Moreover, these scores are very significantly higher than those obtained with GNNExplainer and GraphLIME. As for the recall, the other two methods are sometimes better than our approach or equivalent. However, in these very rare cases where a contender achieves a better recall, its corresponding precision is very low compared to that obtained by our method. It is notably the case in Table 3.2 for A1 on G4 and G5, for A4 and, for A5 on G0, G1, G4, and G5 with GNNExp. As shown in Table 3.3, we observe the same behavior for GraphLIME in the second scenario where we consider only the most important features. To interpret these results, it is important to remember, that a precision of roughly 0.01 with a recall of 1.0 indicates that the method predicted that all features are important i.e. its explanations are pointless. More generally, a very low precision with a very high recall is a sign of poor explanations. These experiments demonstrate, without doubt, that our method is the only one able to explain these

Algo		G_0	G_1	G_2	G_3	G_4	G_5
A_0	Our	$P = \mathbf{0.7} \pm \mathbf{0.21}$	$P = \mathbf{0.48} \pm \mathbf{0.11}$	$P = \mathbf{0.38} \pm \mathbf{0.13}$	$P = \mathbf{0.5} \pm \mathbf{0.0}$	$P = \mathbf{0.56} \pm \mathbf{0.1}$	$P = \mathbf{0.73} \pm \mathbf{0.23}$
		$R = \mathbf{0.82} \pm \mathbf{0.25}$	$R = \mathbf{0.82} \pm \mathbf{0.23}$	$R = \mathbf{0.73} \pm \mathbf{0.29}$	$R = \mathbf{0.83} \pm \mathbf{0.17}$	$R = \mathbf{0.96} \pm \mathbf{0.17}$	$R = \mathbf{0.75} \pm \mathbf{0.29}$
	GraphLIME	$P = 0.06 \pm 0.09$	$P = 0.0 \pm 0.0$	$P = 0.0 \pm 0.0$	$P = 0.0 \pm 0.0$	$P = 0.04 \pm 0.07$	$P = 0.06 \pm 0.1$
		$R = 0.2 \pm 0.29$	$R = 0.02 \pm 0.09$	$R = 0.02 \pm 0.12$	$R = 0.0 \pm 0.0$	$R = 0.2 \pm 0.31$	$R = 0.16 \pm 0.23$
	GNNExp	$P = 0.04 \pm 0.04$	$P = 0.01 \pm 0.01$	$P = 0.0 \pm 0.01$	$P = 0.01 \pm 0.01$	$P = 0.04 \pm 0.04$	$P = 0.06 \pm 0.04$
		$R = 0.46 \pm 0.38$	$R = 0.44 \pm 0.39$	$R = 0.33 \pm 0.34$	$R = 0.37 \pm 0.31$	$R = 0.48 \pm 0.39$	$R = 0.51 \pm 0.32$
A_1	Our	$P = \mathbf{0.62} \pm \mathbf{0.32}$	$P = \mathbf{0.49} \pm \mathbf{0.11}$	$P = \mathbf{0.52} \pm \mathbf{0.06}$	$P = \mathbf{0.47} \pm \mathbf{0.11}$	$P = \mathbf{0.5} \pm \mathbf{0.17}$	$P = \mathbf{0.57} \pm \mathbf{0.35}$
		$R = \mathbf{0.5} \pm \mathbf{0.11}$	$R = \mathbf{0.47} \pm \mathbf{0.12}$	$R = \mathbf{0.51} \pm \mathbf{0.08}$	$R = \mathbf{0.44} \pm \mathbf{0.13}$	$R = 0.5 \pm 0.06$	$R = 0.48 \pm 0.2$
	GraphLIME	$P = 0.06 \pm 0.09$	$P = 0.0 \pm 0.01$	$P = 0.0 \pm 0.01$	$P = 0.0 \pm 0.0$	$P = 0.01 \pm 0.03$	$P = 0.07 \pm 0.1$
		$R = 0.2 \pm 0.28$	$R = 0.04 \pm 0.14$	$R = 0.02 \pm 0.09$	$R = 0.0 \pm 0.0$	$R = 0.1 \pm 0.2$	$R = 0.23 \pm 0.26$
	GNNExp	$P = 0.04 \pm 0.04$	$P = 0.01 \pm 0.01$	$P = 0.0 \pm 0.01$	$P = 0.01 \pm 0.01$	$P = 0.02 \pm 0.02$	$P = 0.06 \pm 0.04$
		$R = 0.49 \pm 0.39$	$R = 0.45 \pm 0.39$	$R = 0.45 \pm 0.41$	$R = \mathbf{0.44} \pm \mathbf{0.4}$	$R = \mathbf{0.56} \pm \mathbf{0.41}$	$R = \mathbf{0.52} \pm \mathbf{0.39}$
A_2	Our	$P = \mathbf{0.97} \pm \mathbf{0.15}$	$P = \mathbf{0.68} \pm \mathbf{0.26}$	$P = \mathbf{0.65} \pm \mathbf{0.28}$	$P = \mathbf{0.5} \pm \mathbf{0.34}$	$P = \mathbf{0.98} \pm \mathbf{0.08}$	$P = \mathbf{0.77} \pm \mathbf{0.37}$
		$R = \mathbf{0.71} \pm \mathbf{0.28}$	$R = \mathbf{0.7} \pm \mathbf{0.29}$	$R = \mathbf{0.63} \pm \mathbf{0.27}$	$R = \mathbf{0.53} \pm \mathbf{0.39}$	$R = \mathbf{0.9} \pm \mathbf{0.2}$	$R = \mathbf{0.63} \pm \mathbf{0.35}$
	GraphLIME	$P = 0.06 \pm 0.09$	$P = 0.0 \pm 0.0$	$P = 0.0 \pm 0.0$	$P = 0.0 \pm 0.0$	$P = 0.03 \pm 0.05$	$P = 0.06 \pm 0.1$
		$R = 0.19 \pm 0.27$	$R = 0.02 \pm 0.09$	$R = 0.02 \pm 0.11$	$R = 0.0 \pm 0.0$	$R = 0.22 \pm 0.3$	$R = 0.16 \pm 0.24$
	GNNExp	$P = 0.05 \pm 0.04$	$P = 0.01 \pm 0.01$	$P = 0.0 \pm 0.0$	$P = 0.0 \pm 0.0$	$P = 0.03 \pm 0.03$	$P = 0.05 \pm 0.04$
		$R = 0.51 \pm 0.4$	$R = 0.46 \pm 0.39$	$R = 0.39 \pm 0.4$	$R = 0.35 \pm 0.34$	$R = 0.49 \pm 0.35$	$R = 0.54 \pm 0.35$
A_3	Our	$P = \mathbf{0.98} \pm \mathbf{0.09}$	$P = \mathbf{1.0} \pm \mathbf{0.0}$	$P = \mathbf{1.0} \pm \mathbf{0.0}$	$P = \mathbf{0.97} \pm \mathbf{0.11}$	$P = \mathbf{1.0} \pm \mathbf{0.0}$	$P = \mathbf{0.99} \pm \mathbf{0.08}$
		$R = \mathbf{1.0} \pm \mathbf{0.0}$	$R = \mathbf{1.0} \pm \mathbf{0.0}$	$R = \mathbf{1.0} \pm \mathbf{0.0}$	$R = \mathbf{1.0} \pm \mathbf{0.0}$	$R = \mathbf{1.0} \pm \mathbf{0.0}$	$R = \mathbf{1.0} \pm \mathbf{0.0}$
	GraphLIME	$P = 0.05 \pm 0.08$	$P = 0.03 \pm 0.05$	$P = 0.03 \pm 0.04$	$P = 0.07 \pm 0.07$	$P = 0.02 \pm 0.04$	$P = 0.05 \pm 0.09$
		$R = 0.29 \pm 0.45$	$R = 0.77 \pm 0.42$	$R = 0.77 \pm 0.42$	$R = 0.94 \pm 0.23$	$R = 0.27 \pm 0.44$	$R = 0.29 \pm 0.45$
	GNNExp	$P = 0.05 \pm 0.03$	$P = 0.03 \pm 0.03$	$P = 0.03 \pm 0.03$	$P = 0.02 \pm 0.02$	$P = 0.03 \pm 0.04$	$P = 0.07 \pm 0.03$
		$R = 0.78 \pm 0.41$	$R = \mathbf{1.0} \pm \mathbf{0.0}$	$R = \mathbf{1.0} \pm \mathbf{0.0}$	$R = \mathbf{1.0} \pm \mathbf{0.0}$	$R = 0.6 \pm 0.49$	$R = 0.98 \pm 0.15$
A_4	Our	$P = \mathbf{1.0} \pm \mathbf{0.0}$	$P = \mathbf{1.0} \pm \mathbf{0.0}$	$P = \mathbf{1.0} \pm \mathbf{0.0}$	$P = \mathbf{1.0} \pm \mathbf{0.0}$	$P = \mathbf{1.0} \pm \mathbf{0.0}$	$P = \mathbf{1.0} \pm \mathbf{0.0}$
		$R = 0.45 \pm 0.09$	$R = 0.22 \pm 0.11$	$R = 0.25 \pm 0.14$	$R = 0.26 \pm 0.15$	$R = 0.38 \pm 0.1$	$R = 0.47 \pm 0.08$
	GraphLIME	$P = 0.11 \pm 0.1$	$P = 0.03 \pm 0.06$	$P = 0.03 \pm 0.03$	$P = 0.06 \pm 0.11$	$P = 0.06 \pm 0.05$	$P = 0.12 \pm 0.11$
		$R = 0.31 \pm 0.24$	$R = 0.21 \pm 0.11$	$R = 0.22 \pm 0.14$	$R = 0.21 \pm 0.15$	$R = 0.28 \pm 0.19$	$R = 0.34 \pm 0.27$
	GNNExp	$P = 0.08 \pm 0.05$	$P = 0.04 \pm 0.03$	$P = 0.03 \pm 0.02$	$P = 0.05 \pm 0.04$	$P = 0.04 \pm 0.04$	$P = 0.1 \pm 0.05$
		$R = \mathbf{0.67} \pm \mathbf{0.36}$	$R = \mathbf{0.65} \pm \mathbf{0.29}$	$R = \mathbf{0.68} \pm \mathbf{0.29}$	$R = \mathbf{0.67} \pm \mathbf{0.29}$	$R = \mathbf{0.47} \pm \mathbf{0.33}$	$R = \mathbf{0.64} \pm \mathbf{0.31}$
A_5	Our	$P = \mathbf{0.88} \pm \mathbf{0.22}$	$P = \mathbf{0.77} \pm \mathbf{0.14}$	$P = \mathbf{0.67} \pm \mathbf{0.0}$	$P = \mathbf{1.0} \pm \mathbf{0.0}$	$P = \mathbf{0.96} \pm \mathbf{0.09}$	$P = \mathbf{0.81} \pm \mathbf{0.26}$
		$R = 0.35 \pm 0.21$	$R = 0.55 \pm 0.12$	$R = \mathbf{0.67} \pm \mathbf{0.0}$	$R = \mathbf{0.83} \pm \mathbf{0.24}$	$R = 0.31 \pm 0.28$	$R = 0.41 \pm 0.28$
	GraphLIME	$P = 0.25 \pm 0.18$	$P = 0.1 \pm 0.04$	$P = 0.12 \pm 0.06$	$P = 0.14 \pm 0.06$	$P = 0.24 \pm 0.07$	$P = 0.19 \pm 0.18$
		$R = 0.44 \pm 0.3$	$R = 0.53 \pm 0.16$	$R = 0.51 \pm 0.19$	$R = 0.49 \pm 0.18$	$R = 0.61 \pm 0.12$	$R = 0.3 \pm 0.28$
	GNNExp	$P = 0.11 \pm 0.06$	$P = 0.05 \pm 0.02$	$P = 0.05 \pm 0.03$	$P = 0.06 \pm 0.02$	$P = 0.08 \pm 0.02$	$P = 0.15 \pm 0.06$
		$R = \mathbf{0.62} \pm \mathbf{0.34}$	$R = \mathbf{0.6} \pm \mathbf{0.27}$	$R = 0.58 \pm 0.32$	$R = 0.7 \pm 0.34$	$R = \mathbf{0.7} \pm \mathbf{0.21}$	$R = \mathbf{0.61} \pm \mathbf{0.28}$
A_6	Our	$P = \mathbf{0.85} \pm \mathbf{0.2}$	$P = \mathbf{1.0} \pm \mathbf{0.0}$	$P = \mathbf{1.0} \pm \mathbf{0.0}$	$P = \mathbf{1.0} \pm \mathbf{0.0}$	$P = \mathbf{0.64} \pm \mathbf{0.34}$	$P = \mathbf{0.86} \pm \mathbf{0.2}$
		$R = \mathbf{0.97} \pm \mathbf{0.14}$	$R = \mathbf{1.0} \pm \mathbf{0.0}$	$R = \mathbf{1.0} \pm \mathbf{0.0}$	$R = \mathbf{1.0} \pm \mathbf{0.0}$	$R = \mathbf{0.83} \pm \mathbf{0.37}$	$R = \mathbf{0.97} \pm \mathbf{0.14}$
	GraphLIME	$P = 0.05 \pm 0.09$	$P = 0.01 \pm 0.01$	$P = 0.0 \pm 0.01$	$P = 0.0 \pm 0.0$	$P = 0.04 \pm 0.04$	$P = 0.04 \pm 0.09$
		$R = 0.25 \pm 0.38$	$R = 0.14 \pm 0.23$	$R = 0.03 \pm 0.11$	$R = 0.0 \pm 0.0$	$R = 0.34 \pm 0.4$	$R = 0.16 \pm 0.33$
	GNNExp	$P = 0.05 \pm 0.03$	$P = 0.01 \pm 0.01$	$P = 0.01 \pm 0.01$	$P = 0.0 \pm 0.01$	$P = 0.04 \pm 0.03$	$P = 0.07 \pm 0.03$
		$R = 0.73 \pm 0.4$	$R = 0.47 \pm 0.39$	$R = 0.5 \pm 0.4$	$R = 0.22 \pm 0.34$	$R = 0.67 \pm 0.39$	$R = 0.88 \pm 0.24$

Table 3.2: Average precision and recall \pm standard deviation computed on all important features
- Bold font is best.

Algo	G_0	G_1	G_2	G_3	G_4	G_5
A_0	Our	$P = \mathbf{0.86} \pm \mathbf{0.25}$ $R = \mathbf{0.72} \pm \mathbf{0.29}$	$P = \mathbf{0.7} \pm \mathbf{0.32}$ $R = 0.68 \pm 0.33$	$P = \mathbf{0.67} \pm \mathbf{0.33}$ $R = 0.67 \pm 0.33$	$P = \mathbf{0.67} \pm \mathbf{0.33}$ $R = 0.67 \pm 0.33$	$P = \mathbf{0.88} \pm \mathbf{0.25}$ $R = \mathbf{0.67} \pm \mathbf{0.3}$
	GraphLIME	$P = 0.08 \pm 0.09$ $R = 0.54 \pm 0.42$	$P = 0.01 \pm 0.0$ $R = \mathbf{0.91} \pm \mathbf{0.27}$	$P = 0.01 \pm 0.0$ $R = \mathbf{0.96} \pm \mathbf{0.17}$	$P = 0.01 \pm 0.0$ $R = \mathbf{1.0} \pm \mathbf{0.0}$	$P = 0.06 \pm 0.08$ $R = 0.71 \pm 0.35$
	GNNEExp	$P = 0.08 \pm 0.11$ $R = 0.3 \pm 0.35$	$P = 0.01 \pm 0.02$ $R = 0.1 \pm 0.2$	$P = 0.01 \pm 0.01$ $R = 0.09 \pm 0.17$	$P = 0.01 \pm 0.01$ $R = 0.1 \pm 0.16$	$P = 0.05 \pm 0.08$ $R = 0.17 \pm 0.24$
						$P = 0.1 \pm 0.15$ $R = \mathbf{0.23} \pm \mathbf{0.28}$
A_1	Our	$P = \mathbf{0.7} \pm \mathbf{0.26}$ $R = 0.48 \pm 0.07$	$P = \mathbf{0.49} \pm \mathbf{0.11}$ $R = 0.47 \pm 0.12$	$P = \mathbf{0.52} \pm \mathbf{0.06}$ $R = 0.51 \pm 0.08$	$P = \mathbf{0.47} \pm \mathbf{0.11}$ $R = 0.44 \pm 0.13$	$P = \mathbf{0.54} \pm \mathbf{0.14}$ $R = 0.5 \pm 0.06$
	GraphLIME	$P = 0.08 \pm 0.07$ $R = \mathbf{0.64} \pm \mathbf{0.38}$	$P = 0.01 \pm 0.01$ $R = \mathbf{0.95} \pm \mathbf{0.17}$	$P = 0.0 \pm 0.0$ $R = \mathbf{0.94} \pm \mathbf{0.23}$	$P = 0.01 \pm 0.0$ $R = \mathbf{1.0} \pm \mathbf{0.0}$	$P = 0.04 \pm 0.03$ $R = \mathbf{0.83} \pm \mathbf{0.3}$
	GNNEExp	$P = 0.1 \pm 0.11$ $R = 0.37 \pm 0.36$	$P = 0.01 \pm 0.02$ $R = 0.2 \pm 0.25$	$P = 0.01 \pm 0.02$ $R = 0.2 \pm 0.23$	$P = 0.02 \pm 0.02$ $R = 0.26 \pm 0.23$	$P = 0.05 \pm 0.06$ $R = 0.3 \pm 0.31$
						$P = 0.13 \pm 0.13$ $R = \mathbf{0.46} \pm \mathbf{0.39}$
A_2	Our	$P = \mathbf{0.98} \pm \mathbf{0.13}$ $R = \mathbf{0.73} \pm \mathbf{0.26}$	$P = \mathbf{0.69} \pm \mathbf{0.29}$ $R = 0.56 \pm 0.22$	$P = \mathbf{0.67} \pm \mathbf{0.3}$ $R = 0.53 \pm 0.22$	$P = \mathbf{0.53} \pm \mathbf{0.37}$ $R = 0.41 \pm 0.3$	$P = \mathbf{1.0} \pm \mathbf{0.0}$ $R = \mathbf{0.9} \pm \mathbf{0.2}$
	GraphLIME	$P = 0.08 \pm 0.08$ $R = 0.61 \pm 0.4$	$P = 0.01 \pm 0.0$ $R = \mathbf{0.94} \pm \mathbf{0.22}$	$P = 0.0 \pm 0.0$ $R = \mathbf{0.99} \pm \mathbf{0.08}$	$P = 0.01 \pm 0.0$ $R = \mathbf{1.0} \pm \mathbf{0.0}$	$P = 0.05 \pm 0.04$ $R = 0.72 \pm 0.32$
	GNNEExp	$P = 0.09 \pm 0.12$ $R = 0.35 \pm 0.36$	$P = 0.01 \pm 0.02$ $R = 0.16 \pm 0.23$	$P = 0.01 \pm 0.01$ $R = 0.16 \pm 0.22$	$P = 0.01 \pm 0.01$ $R = 0.15 \pm 0.23$	$P = 0.05 \pm 0.09$ $R = 0.28 \pm 0.3$
						$P = 0.24 \pm 0.28$ $R = \mathbf{0.79} \pm \mathbf{0.36}$
A_3	Our	$P = \mathbf{1.0} \pm \mathbf{0.0}$ $R = \mathbf{1.0} \pm \mathbf{0.0}$	$P = \mathbf{1.0} \pm \mathbf{0.0}$ $R = \mathbf{1.0} \pm \mathbf{0.0}$	$P = \mathbf{1.0} \pm \mathbf{0.0}$ $R = \mathbf{1.0} \pm \mathbf{0.0}$	$P = \mathbf{0.94} \pm \mathbf{0.23}$ $R = \mathbf{0.94} \pm \mathbf{0.23}$	$P = \mathbf{1.0} \pm \mathbf{0.0}$ $R = \mathbf{1.0} \pm \mathbf{0.0}$
	GraphLIME	$P = 0.06 \pm 0.08$ $R = 0.51 \pm 0.5$	$P = 0.03 \pm 0.05$ $R = 0.77 \pm 0.42$	$P = 0.03 \pm 0.04$ $R = 0.79 \pm 0.41$	$P = 0.07 \pm 0.07$ $R = \mathbf{0.94} \pm \mathbf{0.23}$	$P = 0.03 \pm 0.04$ $R = 0.6 \pm 0.49$
	GNNEExp	$P = 0.07 \pm 0.12$ $R = 0.41 \pm 0.49$	$P = 0.03 \pm 0.1$ $R = 0.12 \pm 0.32$	$P = 0.03 \pm 0.1$ $R = 0.21 \pm 0.41$	$P = 0.06 \pm 0.13$ $R = 0.28 \pm 0.45$	$P = 0.12 \pm 0.2$ $R = 0.47 \pm 0.5$
						$P = 0.06 \pm 0.09$ $R = \mathbf{0.98} \pm \mathbf{0.15}$
A_4	Our	$P = \mathbf{1.0} \pm \mathbf{0.0}$ $R = 0.45 \pm 0.09$	$P = \mathbf{1.0} \pm \mathbf{0.0}$ $R = 0.22 \pm 0.11$	$P = \mathbf{1.0} \pm \mathbf{0.0}$ $R = \mathbf{0.25} \pm \mathbf{0.14}$	$P = \mathbf{1.0} \pm \mathbf{0.0}$ $R = \mathbf{0.26} \pm \mathbf{0.15}$	$P = \mathbf{1.0} \pm \mathbf{0.0}$ $R = 0.38 \pm 0.1$
	GraphLIME	$P = 0.12 \pm 0.1$ $R = \mathbf{0.47} \pm \mathbf{0.3}$	$P = 0.03 \pm 0.06$ $R = 0.21 \pm 0.11$	$P = 0.03 \pm 0.03$ $R = 0.22 \pm 0.14$	$P = 0.06 \pm 0.11$ $R = 0.21 \pm 0.15$	$P = 0.07 \pm 0.04$ $R = \mathbf{0.41} \pm \mathbf{0.28}$
	GNNEExp	$P = 0.16 \pm 0.15$ $R = 0.41 \pm 0.33$	$P = 0.07 \pm 0.08$ $R = \mathbf{0.25} \pm \mathbf{0.21}$	$P = 0.05 \pm 0.05$ $R = 0.23 \pm 0.22$	$P = 0.08 \pm 0.12$ $R = 0.22 \pm 0.2$	$P = 0.08 \pm 0.11$ $R = 0.31 \pm 0.34$
						$P = 0.19 \pm 0.17$ $R = \mathbf{0.47} \pm \mathbf{0.08}$
A_5	Our	$P = \mathbf{0.89} \pm \mathbf{0.21}$ $R = 0.36 \pm 0.22$	$P = \mathbf{0.77} \pm \mathbf{0.14}$ $R = \mathbf{0.55} \pm \mathbf{0.12}$	$P = \mathbf{0.67} \pm \mathbf{0.0}$ $R = \mathbf{0.67} \pm \mathbf{0.0}$	$P = \mathbf{1.0} \pm \mathbf{0.0}$ $R = \mathbf{0.83} \pm \mathbf{0.24}$	$P = \mathbf{0.96} \pm \mathbf{0.09}$ $R = 0.31 \pm 0.28$
	GraphLIME	$P = 0.27 \pm 0.16$ $R = \mathbf{0.6} \pm \mathbf{0.28}$	$P = 0.1 \pm 0.04$ $R = 0.53 \pm 0.16$	$P = 0.12 \pm 0.06$ $R = 0.53 \pm 0.19$	$P = 0.14 \pm 0.06$ $R = 0.54 \pm 0.17$	$P = 0.24 \pm 0.07$ $R = \mathbf{0.61} \pm \mathbf{0.12}$
	GNNEExp	$P = 0.26 \pm 0.2$ $R = 0.47 \pm 0.32$	$P = 0.1 \pm 0.07$ $R = 0.25 \pm 0.18$	$P = 0.09 \pm 0.08$ $R = 0.21 \pm 0.2$	$P = 0.09 \pm 0.09$ $R = 0.28 \pm 0.29$	$P = 0.14 \pm 0.12$ $R = 0.27 \pm 0.23$
						$P = 0.28 \pm 0.23$ $R = \mathbf{0.83} \pm \mathbf{0.24}$
A_6	Our	$P = \mathbf{0.95} \pm \mathbf{0.2}$ $R = \mathbf{0.94} \pm \mathbf{0.21}$	$P = \mathbf{1.0} \pm \mathbf{0.0}$ $R = \mathbf{1.0} \pm \mathbf{0.0}$	$P = \mathbf{1.0} \pm \mathbf{0.0}$ $R = \mathbf{1.0} \pm \mathbf{0.0}$	$P = \mathbf{1.0} \pm \mathbf{0.0}$ $R = \mathbf{1.0} \pm \mathbf{0.0}$	$P = \mathbf{0.68} \pm \mathbf{0.46}$ $R = \mathbf{0.83} \pm \mathbf{0.37}$
	GraphLIME	$P = 0.07 \pm 0.09$ $R = 0.42 \pm 0.44$	$P = 0.01 \pm 0.01$ $R = 0.66 \pm 0.41$	$P = 0.01 \pm 0.01$ $R = 0.89 \pm 0.29$	$P = 0.01 \pm 0.01$ $R = 0.94 \pm 0.23$	$P = 0.05 \pm 0.04$ $R = 0.44 \pm 0.36$
	GNNEExp	$P = 0.11 \pm 0.12$ $R = 0.5 \pm 0.46$	$P = 0.01 \pm 0.02$ $R = 0.16 \pm 0.28$	$P = 0.01 \pm 0.02$ $R = 0.22 \pm 0.31$	$P = 0.0 \pm 0.02$ $R = 0.08 \pm 0.25$	$P = 0.07 \pm 0.06$ $R = 0.41 \pm 0.38$
						$P = 0.04 \pm 0.08$ $R = \mathbf{0.91} \pm \mathbf{0.26}$

Table 3.3: Average precision and recall \pm standard deviation computed on the most important features - Bold font is best.

Methods	Our	GraphLIME	GNNEExp
All	$P = 0.80 \pm 0.20$	$P = 0.05 \pm 0.06$	$P = 0.04 \pm 0.03$
	$R = 0.69 \pm 0.25$	$R = 0.25 \pm 0.22$	$R = 0.58 \pm 0.18$
Most	$P = 0.84 \pm 0.17$	$P = 0.06 \pm 0.06$	$P = 0.07 \pm 0.06$
	$R = 0.66 \pm 0.27$	$R = 0.64 \pm 0.25$	$R = 0.26 \pm 0.11$

Table 3.4: Average precision and recall over all graphs and anomaly types for the "all important features" and "most important features" scenario.

different types of contextual anomalies, even if for a few of them, notably A4 and A5 it returns only a part of the true features.

3.6.4 Hyper-parameter tuning

There are two important parameters in our method. First, the probability p of masking a feature influences the way the data is modified to create perturbed instances. Second, the number of perturbed samples k controls how populated our explanatory model g is. To find the best set of parameters, we conducted experiments with a wide range of p and k simultaneously. With the exact same experimental setup as described above, we cut the space of hyper-parameters with p ranging from 0 to 1 by step of 0.1 and k varying in 0, 10, 50, 100, 200, 500, 1000. To have a global overview of the results and to give an optimal set of parameters (p, k) , we present the results with the F – measure:

$$F - measure = 2 \frac{precision \times recall}{precision + recall} \quad (3.29)$$

The results of the grid-search for finding the best set of parameters (p, k) are presented. We show the results for the graph G_4 and the anomaly type A_0 in Figure 3.4 and 3.5 and for G_5 with A_3 . The results for the other graphs and anomaly types are presented in the Annexes 3.7.

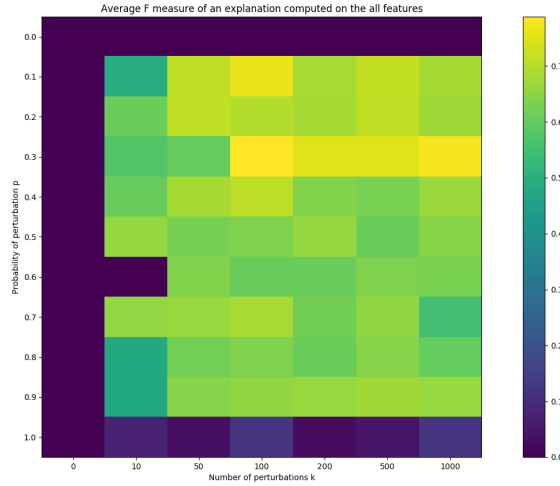


Figure 3.4: Grid search for the parameters p and k reporting the F-measure for the graph G_5 and anomaly A_0 computed on all important features.

The F-measure takes a value between 0 and 1 and is best when close to 1. On the figures, for a specific couple (p, k) , a dark blue rectangle indicates that the model g provided poor explanations. On the other hand, a yellow rectangle is a sign of a good explanation. The first observation is trivial, with $k = 0$ perturbed samples, the model g cannot learn anything since there is only the original sample in the dataset. The same applies for $p = 0$: the "perturbed"

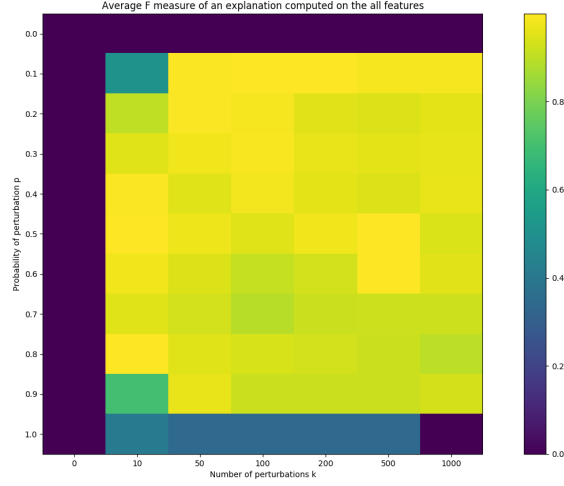


Figure 3.5: Grid search for the parameters p and k reporting the F-measure for the graph G_5 and anomaly A_3 computed on all important features.

samples are not perturbed since $p = 0$. Then, the training dataset for g is composed of the original sample duplicated k times. Then, our results suggest that high values of p imply that the perturbation is too strong and the new samples are too distorted from the original data. This distortion prevents the model g from being able to learn efficiently the important features. We recommend the use of values of p between 0.1 and 0.3. As for k , there is a minimum amount of perturbations that is necessary for the model g to learn correctly. But, on the contrary, too many samples may decrease the F-measure as in Figure 3.4 for $k = 1000$ and $p = 0.7$. We recommend to use values for k between 100 and 500. Moreover, sometimes as in Figure 3.5, the values of p and k do not change the F-measure a lot. We still see that lower values of p , between 0.1 and 0.5, are better.

To sum up, the experiments show that our method is the only one able to find the important features that lead a model to a specific decision. As for contextual anomalies, our explaining method can thus find the contexts that define an anomaly type. It is also relatively easy to tune the two parameters of the method. Clearly, the explaining method that we propose is better than current state-of-the-art models.

3.7 Conclusion

We present a novel method for explaining the predictions of black-box models on graphs, such as GNNs. For a specific node, our method can find important features characterizing the node or nodes in the vicinity of this specific node that led to its predicted class by a black-box model. State-of-the-art methods are not as precise as that. Indeed, our model is not only able to uncover the important features which explain the decision but also on which nodes in the vicinity of the studied node, including the node itself, they occur. Thanks to the use of

contextual anomalies as ground truth, we also proposed a methodological framework and a measure to evaluate the ability of the models to explain correctly a prediction. We conducted experiments that show that our model can accurately explain node classification. Moreover, our method largely outperforms state-of-the-art approaches. We presented our work in the context of anomaly detection with GNNs which is a special case of node classification but the extension of our method to the general case of node classification is straightforward. Indeed, other types of explanation models can also be used like for instance logistic regression and any type of model to explain can also be used. However, the lack of ground truth data will make the evaluation of the models harder in that case. Finally, we would like to consider attributed graphs with numerical features like age or height for instance.

Conclusion

The first main contribution of this thesis is the definition, the generation, and the detection of graph contextual anomalies with graph neural networks in the context of imbalanced data. Contextual anomalies are a specific case of node anomalies where nodes of the graph exhibit a singular arrangement. There are many different types of contextual anomalies depending on the "context". Thanks to our generator ConAGen, it is possible to create new datasets with contextual anomalies and the ground truth knowledge of which nodes are anomalies. Moreover, we presented CoBaGAD, our contextual anomaly detector. With a modified graph attention layer, we can build a neural network to classify the nodes into two categories: normal nodes and anomalies. Through intensive experiments, we show that our model can outperform state-of-the-art methods of classification on graphs by a large margin. While our model has some very good performances, we would like to address some possible improvements. First of all, our generator gives each a one-hot feature vector. That is, the nodes of a graph are only described by one categorical attribute (which can be thought of as a color). Thus, a natural improvement of our generator is to switch to more complex features creation like multiple categorical attributes or multiple numerical attributes. These two setups are closer to real cases of networks. This modification of features will also imply that we can define more complex contextual anomalies and this would be a new challenge for CoBaGAD to tackle. While contextual anomalies may become more complex, CoBaGAD should remain robust to changes and should not be modified, besides maybe the tuning of a few parameters like the number of layers to have a larger field of view. Secondly, we would like to provide an anomalous score. Indeed, in the context of fraud, for example, the experts will focus only on the most important and most probable frauds. Thus, it could be beneficial to have a ranking technique that can find the most relevant anomalies in the network.

The second contribution of this thesis is the explanation of the prediction of a black-box model that performs inference on graph data. Our model belongs to the family of perturbation methods whose goal is to generate new data around a specific instance to learn a local explainable model. For a specific node, our method relies on sampling many times the neighborhood of the node and perturbate it to create a new dataset. These perturbed examples are fed to the black-box model to output a label. Thanks to these perturbed examples associated with their labels, we can train a decision tree locally, *i.e.* for this specific node. The choice of the decision tree is not trivial since the decision process of a tree is human-understandable and it

also provides an importance score for each of the input features. Then, we can use the scores to find the relevant features that led the black-box model to a specific decision. Thanks to contextual anomalies that provide natural ground truth for explaining node classification, we conducted a lot of experiments to retrieve the context that defined the anomalies.

We believe that our work has a lot of potential applications. The detection of anomalies is very important in many domains such as fraud detection in public markets or tax evasion, security with network intrusions, finance with credit card fraud, malware detection, false advertising or fake news propagation. Many fraudulent activities exist and always more complex methods are developed to try not to get caught. Thus, new tools are designed too to detect the new forms of anomalies. This can be achieved with the use of powerful machine learning algorithms. But, machine learning methods also brought a lack of understanding of the functioning of the algorithms. This is mainly due to the large number of parameters that they use. Thus, explaining the decision process of a method is crucial. This is done via explainable artificial intelligence that is a set of new methods that aim at making sense of the inference process of a machine learning method.

Both anomaly detection and explainable artificial intelligence are necessary to make the world fairer. Fairer by finding fraudulent people thanks to anomaly detection and more fair by bringing artificial intelligence to anyone with XAI. The predictive power of machine learning algorithms shall be beneficial for everyone but also understandable to everyone.

Acknowledgement

First of all, I would like to thank Christine and Baptiste for having supervised my research work during the last three years and without whom nothing would have been possible. Their supervision allowed me to grow both scientifically and humanly. Scientifically, thanks to their expertise, their excellent knowledge of the field, their rigorous approach of science well done, their openness and their capacity of integration of the different approaches of graph mining. But also humanly thanks to the learning of the good management of a research project by going through each step from the establishment of a subject to the writing of a paper. Baptiste, you are always ready to help, to exchange ideas, and to shed light on a specific scientific subject. Christine, you knew how to give me a precise scientific framework to be able to advance in the good direction and to support me during the various tests that constitute the doctoral thesis. Baptiste and Christine, you really make a wonderful team and I had a great time working with you during these last three years. I would also like to thank all the Data Intelligence team and the Hubert Curien laboratory for having welcomed me during my thesis and done everything to have good working conditions despite the COVID obstacle.

I would also like to thank the members of the jury who reviewed and evaluated my thesis work and more particularly this manuscript. Thanks to Márton Karsai and Eric Gaussier for the special attention they gave to my thesis. Thanks to Nidhi Hegde for providing her expertise on the subject. And thanks to Christophe Gravier for following the progress of my research during my thesis.

I also pay a special attention to the support I received from my family. Thanks to my parents, Alain and Marie-Agnès, for always supporting me in my studies and for helping me to be able to realize them in excellent conditions. You have always been interested despite the difficulty of apprehending the subject and always available when needed. Thanks also to my darling, Cybèle, for supporting me every day despite the fatigue and the stress sometimes. This thesis would not be the same without you. Thanks to my brothers, Quentin, Robin and Vincent, for their curiosity and their unfailing support. Thanks to my family-in-law for their encouragement and their moral support. And finally, thank you to my friends for their good mood and the moments of comfort that they brought me and that they still bring me.

Bibliography

- Charu C. Aggarwal. *Social Network Data Analytics*. Springer International Publishing, 2011. ISBN 978-1-4419-8462-3. doi: 10.1007/978-1-4419-8462-3. 15
- Charu C. Aggarwal. *Outlier Analysis*. Springer International Publishing, 2016. ISBN 978-3-319-83772-7. doi: 10.1007/978-3-319-47578-3. 41, 44
- Leman Akoglu, Mary McGlohon, and Christos Faloutsos. Oddball: Spotting anomalies in weighted graphs. pages 410–421, 2010. 2, 43, 55, 56, 58
- Leman Akoglu, Hanghang Tong, Brendan Meeder, and Christos Faloutsos. Pics: Parameter-free identification of cohesive subgroups in large attributed graphs. *Proceedings of the 12th SIAM International Conference on Data Mining, SDM 2012*, pages 439–450, 04 2012. doi: 10.1137/1.9781611972825.38. 58
- Leman Akoglu, Hanghang Tong, and Danai Koutra. Graph based anomaly detection and description: a survey. *Data Min. Knowl. Discov.*, 29(3):626–688, 2015. 41
- Aris Anagnostopoulos, Ravi Kumar, and Mohammad Mahdian. Influence and correlation in social networks. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD 08*, page 7, Las Vegas, Nevada, USA, 2008. ACM Press. ISBN 978-1-60558-193-4. doi: 10.1145/1401890.1401897. URL <http://dl.acm.org/citation.cfm?doid=1401890.1401897>. 41
- Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using pagerank vectors. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 475–486, 2006. doi: 10.1109/FOCS.2006.44. 58
- Fabrizio Angiulli and Clara Pizzuti. Fast outlier detection in high dimensional spaces. In Tapio Elomaa, Heikki Mannila, and Hannu Toivonen, editors, *Principles of Data Mining and Knowledge Discovery*, pages 15–27, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. ISBN 978-3-540-45681-0. 50
- A.B. Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, and F. Herrera. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information Fusion*, 58:82–115, 2020. 76

- Tharindu R. Bandaragoda, Kai Ming Ting, David Albrecht, Fei Tony Liu, Ye Zhu, and Jonathan R. Wells. Isolation-based anomaly detection using nearest-neighbor ensembles. *Computational Intelligence*, 34(4):968–998, 2018. 48
- Albert-Laszlo Barabasi and Reka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, Oct 1999. ISSN 1095-9203. doi: 10.1126/science.286.5439.509. URL <http://dx.doi.org/10.1126/science.286.5439.509>. 16
- Belkin and Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Comput.*, 2003. doi: 10.1162/089976603321780317. URL <http://dx.doi.org/10.1162/089976603321780317>. 12, 26, 29
- Richard Bellman. Dynamic programming and stochastic control processes. *Information and Control*, 1(3):228–239, 1958. ISSN 0019-9958. doi: [https://doi.org/10.1016/S0019-9958\(58\)80003-0](https://doi.org/10.1016/S0019-9958(58)80003-0). URL <https://www.sciencedirect.com/science/article/pii/S0019995858800030>. 36
- Smriti Bhagat, Graham Cormode, and S. Muthukrishnan. Node classification in social networks. *Social Network Data Analytics*, page 115–148, 2011. doi: 10.1007/978-1-4419-8462-3_5. URL http://dx.doi.org/10.1007/978-1-4419-8462-3_5. 17
- L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984. 38
- Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: Identifying density-based local outliers. *SIGMOD Rec.*, 29(2):93–104, May 2000. ISSN 0163-5808. doi: 10.1145/335191.335388. URL <https://doi.org/10.1145/335191.335388>. 44, 46, 68
- Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. *arXiv:2005.14165 [cs]*, July 2020. URL <http://arxiv.org/abs/2005.14165>. arXiv: 2005.14165. 37
- Shaosheng Cao, Wei Lu, and Qionghai Xu. Deep neural networks for learning graph representations. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI’16, pages 1145–1152, Phoenix, Arizona, February 2016. AAAI Press. 27
- Deepayan Chakrabarti. Autopart: Parameter-free graph partitioning and outlier detection. In Jean-François Boulicaut, Floriana Esposito, Fosca Giannotti, and Dino Pedreschi, editors, *Knowledge Discovery in Databases: PKDD 2004*, pages 112–124, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-30116-5. 43, 54, 58, 60

-
- Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41(3):1–58, July 2009. ISSN 0360-0300, 1557-7341. doi: 10.1145/1541880.1541882. URL <https://dl.acm.org/doi/10.1145/1541880.1541882>. 60
- Paulo Cortez and Mark J. Embrechts. Opening black box Data Mining models using Sensitivity Analysis. In *2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, pages 341–348, Paris, France, April 2011. IEEE. ISBN 978-1-4244-9926-7. doi: 10.1109/CIDM.2011.5949423. URL <http://ieeexplore.ieee.org/document/5949423/>. 78
- Pedro Domingos and Matt Richardson. Mining the network value of customers. KDD '01, page 57–66, New York, NY, USA, 2001. Association for Computing Machinery. ISBN 158113391X. doi: 10.1145/502512.502525. URL <https://doi.org/10.1145/502512.502525>. 15
- Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. Least angle regression. *The Annals of Statistics*, 32(2):407 – 499, 2004a. doi: 10.1214/009053604000000067. URL <https://doi.org/10.1214/009053604000000067>. 82
- Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. Least angle regression. *The Annals of Statistics*, 32(2):407 – 499, 2004b. doi: 10.1214/009053604000000067. URL <https://doi.org/10.1214/009053604000000067>. 37
- P. Erdős and A. Rényi. On random graphs i. *Publicationes Mathematicae Debrecen*, 6:290, 1959. 15
- Martin Ester, Hans-Peter Kriegel, Joerg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. *KDD*, 96:226–231, 01 1996. 46
- Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75–174, Feb 2010. ISSN 0370-1573. doi: 10.1016/j.physrep.2009.11.002. URL <http://dx.doi.org/10.1016/j.physrep.2009.11.002>. 16
- Santo Fortunato and Darko Hric. Community detection in networks: A user guide. *Physics Reports*, 659, 07 2016. doi: 10.1016/j.physrep.2016.09.002. 16
- Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189 – 1232, 2001. doi: 10.1214/aos/1013203451. URL <https://doi.org/10.1214/aos/1013203451>. 77
- Thomas M. J. Fruchterman and Edward M. Reingold. Graph Drawing by Force-directed Placement. *Softw. Pract. Exper.*, 21(11):1129–1164, November 1991. ISSN 0038-0644. doi: 10.1002/spe.4380211102. URL <http://dx.doi.org/10.1002/spe.4380211102>. 12, 13
- Lise Getoor and Christopher P. Diehl. Link mining: A survey. *SIGKDD Explorations Newsletter*, 7(2):3–12, December 2005. ISSN 1931-0145. doi: 10.1145/1117454.1117456. URL <https://doi.org/10.1145/1117454.1117456>. 17
-

- G. H. Golub and C. Reinsch. Singular value decomposition and least squares solutions. *Numerische Mathematik*, 14(5):403–420, April 1970. ISSN 0945-3245. doi: 10.1007/BF02163027. URL <https://doi.org/10.1007/BF02163027>. 27
- Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>. 18, 22
- Aditya Grover and Jure Leskovec. node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, pages 855–864, San Francisco, California, USA, 2016. ACM Press. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939754. URL <http://dl.acm.org/citation.cfm?doid=2939672.2939754>. 24, 25, 26, 29, 44, 68
- Stephan Günnemann, Ines Färber, Brigitte Boden, and Thomas Seidl. Subspace clustering meets dense subgraph mining: A synthesis of two paradigms. In *2010 IEEE International Conference on Data Mining*, pages 845–850, 2010. doi: 10.1109/ICDM.2010.95. 57
- William L Hamilton, Rex Ying, and Jure Leskovec. Inductive Representation Learning on Large Graphs. page 19, 2017. 31, 32, 44, 68
- Zengyou He, Xiaofei Xu, and Shengchun Deng. Discovering cluster-based local outliers. *Pattern Recognition Letters*, 24(9-10):1641–1650, June 2003. ISSN 0167-8655. doi: 10.1016/S0167-8655(03)00003-5. URL [https://doi.org/10.1016/S0167-8655\(03\)00003-5](https://doi.org/10.1016/S0167-8655(03)00003-5). 47
- Dan Hendrycks and Kevin Gimpel. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, abs/1606.08415, 2016. 68
- Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL <https://doi.org/10.1162/neco.1997.9.8.1735>. 32
- Victoria Hodge and Jim Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22:85–126, 10 2004. doi: 10.1023/B:AIRE.0000045502.10941.a9. 41, 42
- Qiang Huang, Makoto Yamada, Yuan Tian, Dinesh Singh, Dawei Yin, and Yi Chang. Graphlime: Local interpretable model explanations for graph neural networks, 2020. 81, 91, 93
- Jimeng Sun, Huiming Qu, D. Chakrabarti, and C. Faloutsos. Neighborhood Formation and Anomaly Detection in Bipartite Graphs. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 418–425, Houston, TX, USA, 2005. IEEE. ISBN 978-0-7695-2278-4. doi: 10.1109/ICDM.2005.103. URL <http://ieeexplore.ieee.org/document/1565707/>. 55
- Tomihisa Kamada and Satoru Kawai. An algorithm for drawing general undirected graphs. *Inf. Process. Lett.*, 31:7–15, 04 1989. doi: 10.1016/0020-0190(89)90102-6. 12, 29

- Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL <http://arxiv.org/abs/1412.6980>. 69
- T. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. 2017. 31, 44, 68
- Thomas N. Kipf and Max Welling. Variational graph auto-encoders, 2016. 28
- Josua Krause, Adam Perer, and Kenney Ng. Interacting with predictions: Visual inspection of black-box machine learning models. pages 5686–5697, 05 2016. doi: 10.1145/2858036.2858529. 77
- Hans-Peter Kriegel, Matthias S hubert, and Arthur Zimek. Angle-based outlier detection in high-dimensional data. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD 08*, page 444, Las Vegas, Nevada, USA, 2008. ACM Press. ISBN 978-1-60558-193-4. doi: 10.1145/1401890.1401946. URL <http://dl.acm.org/citation.cfm?doid=1401890.1401946>. 51
- J. Kruskall. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29:1–27, 01 1964. 29
- A. Lancichinetti, S. Fortunato, and F. Radicchi. Benchmark graphs for testing community detection algorithms. *Phys. Rev. E*, 78:046110, 2008. 16, 67
- C. Largeron, P. N. Mougél, , O. Benyahia, and O. Zaïane. Dancer: dynamic attributed networks with community structure generation. *Knowl Inf Syst*, 53:109–151, 2017. 16, 67
- Christine Largeron, Pierre-Nicolas Mougél, Reihaneh Rabbany, and Osmar R. Zaïane. Generating attributed networks with communities. *PLOS ONE*, 10(4):1–21, 04 2015. doi: 10.1371/journal.pone.0122777. URL <https://doi.org/10.1371/journal.pone.0122777>. 17
- Thibault Laugel, Marie-Jeanne Lesot, Christophe Marsala, Xavier Renard, and Marcin Detryniecki. Comparison-based inverse classification for interpretability in machine learning. In *Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 100–111, 2018. 78
- Yann LeCun and Yoshua Bengio. *Convolutional Networks for Images, Speech, and Time Series*, page 255–258. MIT Press, Cambridge, MA, USA, 1998. ISBN 0262511029. 31
- Jure Leskovec and Eric Horvitz. Planetary-scale views on a large instant-messaging network. In *Proceedings of the 17th International Conference on World Wide Web, WWW '08*, page 915–924, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605580852. doi: 10.1145/1367497.1367620. URL <https://doi.org/10.1145/1367497.1367620>. 14

- David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American Society for Information Science and Technology*, 58(7):1019–1031, 2007. doi: <https://doi.org/10.1002/asi.20591>. URL <https://onlinelibrary.wiley.com/doi/abs/10.1002/asi.20591>. 17, 18
- F. Liu, K. Ting, and Z. Zhou. Isolation Forest. In *2008 Eighth IEEE International Conference on Data Mining*, Pisa, Italy, December 2008. IEEE. ISBN 978-0-7695-3502-9. doi: 10.1109/ICDM.2008.17. 48
- Scott M. Lundberg and Su-In Lee. A Unified Approach to Interpreting Model Predictions. *Advances in Neural Information Processing Systems*, 30:4765–4774, 2017. 79, 80
- Dongsheng Luo, Wei Cheng, Dongkuan Xu, Wenchao Yu, Bo Zong, Haifeng Chen, and Xiang Zhang. Parameterized explainer for graph neural network. In *Advances in Neural Information Processing Systems*, volume 33, pages 19620–19631, 2020. 80, 83
- Xiaoxiao Ma, Jia Wu, Shan Xue, Jian Yang, Chuan Zhou, Quan Z. Sheng, Hui Xiong, and Leman Akoglu. A comprehensive survey on graph anomaly detection with deep learning, 2021. 41
- Michael Mannino, Yanjuan Yang, and Young Ryu. Classification algorithm sensitivity to training data with non representative attribute noise. *Decision Support Systems*, 46(3): 743–751, 2009. ISSN 0167-9236. doi: <https://doi.org/10.1016/j.dss.2008.11.021>. URL <https://www.sciencedirect.com/science/article/pii/S0167923608002200>. Wireless in the Healthcare. 75
- Kishan Mehrotra, Chilukuri Mohan, and HuaMing Huang. *Anomaly Detection Principles and Algorithms*. 01 2017. ISBN 978-3-319-67524-4. doi: 10.1007/978-3-319-67526-8. 41
- Tomas Mikolov, Ilya Sutskever, Kai Chen, G.s Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems*, 26, 10 2013. 23
- Stanley Milgram. The small world problem. *Psychology Today*, pages 60–67, May 1967. 12
- Flavia Moser, Recep Colak, Arash Rafiey, and Martin Ester. Mining cohesive patterns from graphs with feature vectors. In *SDM*, pages 593–604. SIAM, 2009. ISBN 978-1-61197-279-5. 57
- E. Müller, P. Sánchez, Y. Mülle, and K. Böhm. Ranking outlier nodes in subspaces of attributed graphs. In *2013 IEEE 29th International Conference on Data Engineering Workshops (ICDEW)*, pages 216–222, April 2013. doi: 10.1109/ICDEW.2013.6547453. 56
- M. E. J. Newman. Assortative mixing in networks. *Physical Review Letters*, 89(20):208701, October 2002. ISSN 0031-9007, 1079-7114. doi: 10.1103/PhysRevLett.89.208701. URL <http://arxiv.org/abs/cond-mat/0205405>. arXiv: cond-mat/0205405. 41

- M. E. J. Newman. Mixing patterns in networks. *Physical Review E*, 67(2):026126, February 2003. ISSN 1063-651X, 1095-3787. doi: 10.1103/PhysRevE.67.026126. URL <http://arxiv.org/abs/cond-mat/0209450>. arXiv: cond-mat/0209450. 41
- M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Phys. Rev. E*, 69:066133, Jun 2004. doi: 10.1103/PhysRevE.69.066133. URL <https://link.aps.org/doi/10.1103/PhysRevE.69.066133>. 17
- M. E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006. ISSN 0027-8424. doi: 10.1073/pnas.0601602103. URL <https://www.pnas.org/content/103/23/8577>. 16, 58
- M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69:026113, Feb 2004. doi: 10.1103/PhysRevE.69.026113. URL <https://link.aps.org/doi/10.1103/PhysRevE.69.026113>. 17
- C. C. Noble and D. J. Cook. Graph-based anomaly detection. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, New York, NY, USA, 2003. Association for Computing Machinery. ISBN 978-1-58113-737-8. doi: 10.1145/956750.956831. 55
- Ou, Cui, Pei, Zhang, and Zhu. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22Nd ACM SIGKDD Conference*, 2016. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939751. 26, 27, 29
- M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu. A survey on network embedding. *CoRR*, abs/1711.08752, 2017. 23
- Guansong Pang, Longbing Cao, Ling Chen, and Huan Liu. Learning Representations of Ultrahigh-dimensional Data for Random Distance-based Outlier Detection. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining - KDD '18*, pages 2041–2050, 2018. doi: 10.1145/3219819.3220042. URL <http://arxiv.org/abs/1806.04808>. arXiv: 1806.04808. 52
- B. Perozzi and L. Akoglu. Discovering Communities and Anomalies in Attributed Graphs: Interactive Visual Exploration and Summarization. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 12(2):24, March 2018. ISSN 1556-4681. doi: 10.1145/3139241. 58, 60
- B. Perozzi, R. Al-Rfou, and S. Skiena. DeepWalk: online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '14, New York, NY, USA, August 2014a. Association for Computing Machinery. ISBN 978-1-4503-2956-9. doi: 10.1145/2623330.2623732. 24

- Bryan Perozzi, Leman Akoglu, Patricia Iglesias Sánchez, and Emmanuel Müller. Focused clustering and outlier detection in large attributed graphs. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '14*, pages 1346–1355, New York, New York, USA, 2014b. ACM Press. ISBN 978-1-4503-2956-9. doi: 10.1145/2623330.2623682. URL <http://dl.acm.org/citation.cfm?doid=2623330.2623682>. 57
- Phillip E. Pope, Soheil Kolouri, Mohammad Rostami, Charles E. Martin, and Heiko Hoffmann. Explainability methods for graph convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 80
- Tahereh Pourhabibi, Kok-Leong Ong, Booi H. Kam, and Yee Ling Boo. Fraud detection: A systematic literature review of graph-based anomaly detection approaches. *Decision Support Systems*, 133:113303, June 2020. ISSN 0167-9236. doi: 10.1016/j.dss.2020.113303. URL <https://www.sciencedirect.com/science/article/pii/S0167923620300580>. 53
- Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving Language Understanding by Generative Pre-Training. page 12, 2018. 37
- Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. page 24, 2019. 37
- Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient algorithms for mining outliers from large data sets. volume 29, pages 427–438, 06 2000. doi: 10.1145/335191.335437. 49
- Stephen Ranshous, Shitian Shen, Danai Koutra, Steve Harenberg, Christos Faloutsos, and Nagiza F. Samatova. Anomaly detection in dynamic networks: a survey. *Wiley Interdisciplinary Reviews: Computational Statistics*, 7(3):223–247, May 2015. ISSN 19395108. doi: 10.1002/wics.1347. URL <http://doi.wiley.com/10.1002/wics.1347>. 41
- Leonardo Ribeiro, Pedro Saverese, and Daniel Figueiredo. struc2vec: Learning node representations from structural identity. pages 385–394, 08 2017. doi: 10.1145/3097983.3098061. 29
- Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 1135–1144, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342322. doi: 10.1145/2939672.2939778. URL <https://doi.org/10.1145/2939672.2939778>. 75, 79, 81
- J. Rissanen. Paper: Modeling by shortest data description. *Automatica*, 14(5):465–471, September 1978. ISSN 0005-1098. doi: 10.1016/0005-1098(78)90005-5. URL [https://doi.org/10.1016/0005-1098\(78\)90005-5](https://doi.org/10.1016/0005-1098(78)90005-5). 54

- Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958. 21
- Roweis and Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 2000. doi: 10.1126/science.290.5500.2323. URL <http://science.sciencemag.org/content/290/5500/2323>. 12, 26, 29
- S. Sathe and C. Aggarwal. LODS: Local Density Meets Spectral Outlier Detection. In *Proceedings of the 2016 SIAM International Conference on Data Mining*, pages 171–179. Society for Industrial and Applied Mathematics, 2016. doi: 10.1137/1.9781611974348.20. 47
- F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009. doi: 10.1109/TNN.2008.2005605. 61
- Michael Sejr Schlichtkrull, Nicola De Cao, and Ivan Titov. Interpreting Graph Neural Networks for NLP With Differentiable Edge Masking. *arXiv:2010.00577 [cs, stat]*, 2021. 80
- Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML’17, page 3145–3153. JMLR.org, 2017. 79
- Aakash Sinha, Rémy Cazabet, and Rémi Vaudaine. Systematic biases in link prediction: comparing heuristic and graph embedding based methods, 2018. 5, 18
- X. Su and C.L. Tsai. Outlier detection. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(3):264–268, 2011. 41, 42
- Mahito Sugiyama and Karsten M. Borgwardt. Rapid distance-based outlier detection via sampling. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, NIPS’13, page 467–475, Red Hook, NY, USA, 2013. Curran Associates Inc. 52
- Kai Ting, Sunil Aryal, and Takashi Washio. Which outlier detector should i use? pages 8–8, 11 2018. doi: 10.1109/ICDM.2018.00015. 45, 50
- Kai Ming Ting, Ye Zhu, Mark Carman, Yue Zhu, and Zhi-Hua Zhou. Overcoming Key Weaknesses of Distance-based Neighbourhood Methods using a Data Dependent Dissimilarity Measure. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD ’16*, pages 1205–1214, San Francisco, California, USA, 2016. ACM Press. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939779. URL <http://dl.acm.org/citation.cfm?doid=2939672.2939779>. 50
- Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *Proceedings of the 33rd International*

- Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 2071–2080. JMLR.org, 2016. 18
- A. Tsitsulin, D. Mottin, P. Karras, and E. Müller. Verse: Versatile graph embeddings from similarity measures. *WWW '18*, 2018. doi: 10.1145/3178876.3186120. 29
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *arXiv:1706.03762 [cs]*, December 2017. URL <http://arxiv.org/abs/1706.03762>. arXiv: 1706.03762. 22
- Rémi Vaudaine, Christine Largeron, and Rémy Cazabet. Network properties captured by graph embeddings. In *Complex Network*, Cambridge, United Kingdom, 2018. URL <https://hal.archives-ouvertes.fr/hal-02016671>. 5
- Rémi Vaudaine, Rémy Cazabet, and Christine Largeron. Comparing the preservation of network properties by graph embeddings. In Michael R. Berthold, Ad Feelders, and Georg Kreml, editors, *Advances in Intelligent Data Analysis XVIII*, pages 522–534, Cham, 2020. Springer International Publishing. ISBN 978-3-030-44584-3. 5
- Rémi Vaudaine, Baptiste Jeudy, and Christine Largeron. Detection of contextual anomalies in attributed graphs. In Pedro Henriques Abreu, Pedro Pereira Rodrigues, Alberto Fernández, and João Gama, editors, *Advances in Intelligent Data Analysis XIX*, pages 338–349, Cham, 2021a. Springer International Publishing. ISBN 978-3-030-74251-5. 5
- Rémi Vaudaine, Rémy Cazabet, and Christine Largeron. Comparaison de la capacité des plongements de graphes à capturer les propriétés des réseaux. *Revue des Nouvelles Technologies de l'Information*, Extraction et Gestion des Connaissances, RNTI-E-37:517–518, 2021b. 5
- P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018. 31, 33, 34, 43, 44, 62, 68
- Timothy Vries, Sanjay Chawla, and Michael Houle. Finding local anomalies in very high dimensional space. pages 128–137, 12 2010. doi: 10.1109/ICDM.2010.151. 52
- D. Wang, P. Cui, and W. Zhu. Structural deep network embedding. In *SIGKDD*, 2016. doi: 10.1145/2939672.2939753. 27, 28, 29
- Xuhong Wang, Baihong Jin, Ying Du, Ping Cui, Yingshui Tan, and Yupu Yang. One-class graph neural networks for anomaly detection in attributed networks. *Neural Computing and Applications*, 33(18):12073–12085, Mar 2021. ISSN 1433-3058. doi: 10.1007/s00521-021-05924-9. URL <http://dx.doi.org/10.1007/s00521-021-05924-9>. 59, 60
- Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1):181–213, January 2015.

ISSN 0219-3116. doi: 10.1007/s10115-013-0693-z. URL <https://doi.org/10.1007/s10115-013-0693-z>. 59

Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnnexplainer: Generating explanations for graph neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL <https://proceedings.neurips.cc/paper/2019/file/d80b7040b773199015de6d3b4293c8ff-Paper.pdf>. 80, 82, 91, 93

Wenchao Yu, Wei Cheng, Charu Aggarwal, Kai Zhang, Haifeng Chen, and Wei Wang. Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks. pages 2672–2681, 07 2018. doi: 10.1145/3219819.3220024. 59, 60

Hao Yuan, Haiyang Yu, Shurui Gui, and Shuiwang Ji. Explainability in Graph Neural Networks: A Taxonomic Survey. *arXiv:2012.15445 [cs]*, 2021. 80

Wayne W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33(4):452–473, 1977. doi: 10.1086/jar.33.4.3629752. 13

Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: an efficient data clustering method for very large databases. *ACM SIGMOD Record*, 25(2):103–114, June 1996. ISSN 0163-5808. doi: 10.1145/235968.233324. URL <https://doi.org/10.1145/235968.233324>. 50

Annexes

Anomaly detection

We present here additional results for the detection of contextual anomalies with different methods. In Section 2.5.2, we presented tables of the precision of the detection of the anomaly class since this value is the most discriminating one. Here, we report the recall of the anomaly class, the precision and the recall of the normal class for the same experiments as in Section 2.5.2. Most of the recalls and precisions are close to 1. This does not allow for finding the best method. We see that GCN has poor performances in term of recall for the normal nodes.

Anomalies / Graph	G_0	G_1	G_2	G_3	G_4	G_5
CoBaGAD A_0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
CoBaGAD loops A_0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.53 ± 0.0	0.98 ± 0.02
GAT A_0	1.0 ± 0.0	0.99 ± 0.01	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
GAT loops A_0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.6 ± 0.09	1.0 ± 0.0
GCN A_0	0.94 ± 0.02	0.95 ± 0.02	0.96 ± 0.05	0.94 ± 0.0	0.89 ± 0.03	0.83 ± 0.04
GCN loops A_0	0.86 ± 0.03	0.96 ± 0.05	0.96 ± 0.01	1.0 ± 0.0	0.89 ± 0.08	0.85 ± 0.07
GraphSAGE A_0	1.0 ± 0.0	1.0 ± 0.01	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
GraphSAGE loops A_0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.99 ± 0.01

Table 3.5: Recall of the anomalies for A_0 on several graphs (G_0 - G_5) in the testing set.

Anomalies / Graph	G_0	G_1	G_2	G_3	G_4	G_5
CoBaGAD A_0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
CoBaGAD loops A_0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.97 ± 0.0	1.0 ± 0.0
GAT A_0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
GAT loops A_0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.97 ± 0.01	1.0 ± 0.0
GCN A_0	1.0 ± 0.0	0.99 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.99 ± 0.0	0.99 ± 0.0
GCN loops A_0	0.99 ± 0.0	1.0 ± 0.01	1.0 ± 0.0	1.0 ± 0.0	0.99 ± 0.01	0.99 ± 0.01
GraphSAGE A_0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
GraphSAGE loops A_0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0

Table 3.6: Precision of the normal nodes for A_0 on several graphs (G_0 - G_5) in the testing set.

Anomalies / Graph	G_0	G_1	G_2	G_3	G_4	G_5
CoBaGAD A_0	1.0 ± 0.0	1.0 ± 0.0	0.99 ± 0.01	0.95 ± 0.03	0.99 ± 0.01	0.99 ± 0.01
CoBaGAD loops A_0	0.98 ± 0.01	0.99 ± 0.01	1.0 ± 0.0	0.99 ± 0.01	0.85 ± 0.08	0.97 ± 0.01
GAT A_0	1.0 ± 0.0	0.91 ± 0.06	0.88 ± 0.01	0.95 ± 0.02	0.95 ± 0.03	0.99 ± 0.01
GAT loops A_0	0.97 ± 0.01	0.99 ± 0.0	0.89 ± 0.03	0.92 ± 0.03	0.83 ± 0.03	0.95 ± 0.0
GCN A_0	0.88 ± 0.01	0.5 ± 0.02	0.59 ± 0.02	0.74 ± 0.03	0.81 ± 0.03	0.85 ± 0.02
GCN loops A_0	0.82 ± 0.0	0.54 ± 0.05	0.63 ± 0.01	0.69 ± 0.05	0.87 ± 0.02	0.65 ± 0.05
GraphSAGE A_0	0.94 ± 0.01	0.94 ± 0.01	0.97 ± 0.01	0.92 ± 0.03	0.87 ± 0.03	0.94 ± 0.01
GraphSAGE loops A_0	0.94 ± 0.0	0.95 ± 0.01	0.95 ± 0.02	0.91 ± 0.02	0.88 ± 0.04	0.93 ± 0.01

Table 3.7: Recall of the normal nodes for A_0 on several graphs (G_0 - G_5) in the testing set.

Anomalies / Graph	G_0	G_1	G_2	G_3	G_4	G_5
CoBaGAD A_1	1.0 ± 0.0	1.0 ± 0.0	0.99 ± 0.02	0.96 ± 0.03	0.96 ± 0.06	0.98 ± 0.01
CoBaGAD loops A_1	1.0 ± 0.0	1.0 ± 0.0	0.98 ± 0.02	0.98 ± 0.03	0.64 ± 0.03	0.99 ± 0.01
GAT A_1	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
GAT loops A_1	1.0 ± 0.0	1.0 ± 0.0	0.99 ± 0.01	1.0 ± 0.0	0.58 ± 0.13	1.0 ± 0.0
GCN A_1	0.98 ± 0.01	0.92 ± 0.04	0.89 ± 0.06	0.96 ± 0.03	1.0 ± 0.0	0.86 ± 0.03
GCN loops A_1	0.95 ± 0.01	1.0 ± 0.01	0.9 ± 0.03	0.96 ± 0.03	1.0 ± 0.0	0.88 ± 0.1
GraphSAGE A_1	1.0 ± 0.0	0.99 ± 0.01	1.0 ± 0.0	0.98 ± 0.03	0.98 ± 0.03	1.0 ± 0.0
GraphSAGE loops A_1	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0

Table 3.8: Recall of the anomalies for A_1 on several graphs (G_0 - G_5) in the testing set.

Anomalies / Graph	G_0	G_1	G_2	G_3	G_4	G_5
CoBaGAD A_1	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
CoBaGAD loops A_1	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.98 ± 0.0	1.0 ± 0.0
GAT A_1	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
GAT loops A_1	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.97 ± 0.01	1.0 ± 0.0
GCN A_1	1.0 ± 0.0	0.99 ± 0.0	0.99 ± 0.01	1.0 ± 0.0	1.0 ± 0.0	0.99 ± 0.0
GCN loops A_1	1.0 ± 0.0	1.0 ± 0.0	0.99 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.99 ± 0.01
GraphSAGE A_1	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
GraphSAGE loops A_1	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0

Table 3.9: Precision of the normal nodes for A_1 on several graphs (G_0 - G_5) in the testing set.

Anomalies / Graph	G_0	G_1	G_2	G_3	G_4	G_5
CoBaGAD A_1	1.0 ± 0.0	0.99 ± 0.01	0.97 ± 0.04	0.97 ± 0.03	0.97 ± 0.02	0.99 ± 0.01
CoBaGAD loops A_1	0.98 ± 0.0	1.0 ± 0.0	0.98 ± 0.01	0.98 ± 0.01	0.92 ± 0.05	0.97 ± 0.0
GAT A_1	0.95 ± 0.06	0.93 ± 0.04	0.94 ± 0.04	0.96 ± 0.01	0.9 ± 0.06	0.98 ± 0.02
GAT loops A_1	0.97 ± 0.0	0.91 ± 0.02	0.89 ± 0.02	0.96 ± 0.02	0.88 ± 0.04	0.95 ± 0.01
GCN A_1	0.88 ± 0.01	0.52 ± 0.04	0.61 ± 0.03	0.74 ± 0.06	0.86 ± 0.01	0.84 ± 0.01
GCN loops A_1	0.8 ± 0.0	0.73 ± 0.05	0.63 ± 0.09	0.72 ± 0.05	0.87 ± 0.02	0.71 ± 0.04
GraphSAGE A_1	0.93 ± 0.01	0.93 ± 0.01	0.94 ± 0.01	0.9 ± 0.02	0.89 ± 0.03	0.93 ± 0.01
GraphSAGE loops A_1	0.91 ± 0.01	0.94 ± 0.0	0.94 ± 0.01	0.89 ± 0.03	0.85 ± 0.05	0.93 ± 0.01

Table 3.10: Recall of the normal nodes for A_1 on several graphs (G_0 - G_5) in the testing set.

Anomalies / Graph	G_0	G_1	G_2	G_3	G_4	G_5
CoBaGAD A_2	1.0 ± 0.0	0.99 ± 0.0	0.98 ± 0.01	1.0 ± 0.0	1.0 ± 0.0	0.99 ± 0.01
CoBaGAD loops A_2	1.0 ± 0.01	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.67 ± 0.14	0.98 ± 0.02
GAT A_2	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.96 ± 0.06	1.0 ± 0.0	1.0 ± 0.0
GAT loops A_2	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.67 ± 0.05	0.99 ± 0.01
GCN A_2	0.93 ± 0.03	0.84 ± 0.01	0.91 ± 0.02	1.0 ± 0.0	0.96 ± 0.03	0.85 ± 0.08
GCN loops A_2	0.9 ± 0.02	0.97 ± 0.02	0.94 ± 0.02	0.9 ± 0.07	0.82 ± 0.11	0.83 ± 0.09
GraphSAGE A_2	0.99 ± 0.01	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.93 ± 0.09	1.0 ± 0.0
GraphSAGE loops A_2	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.99 ± 0.01

Table 3.11: Recall of the anomalies for A_2 on several graphs (G_0 - G_5) in the testing set.

Anomalies / Graph	G_0	G_1	G_2	G_3	G_4	G_5
CoBaGAD A_2	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
CoBaGAD loops A_2	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.98 ± 0.01	1.0 ± 0.0
GAT A_2	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
GAT loops A_2	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.97 ± 0.0	1.0 ± 0.0
GCN A_2	0.99 ± 0.0	0.98 ± 0.0	0.99 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.99 ± 0.01
GCN loops A_2	0.99 ± 0.0	1.0 ± 0.0	0.99 ± 0.0	0.99 ± 0.01	0.99 ± 0.01	0.98 ± 0.01
GraphSAGE A_2	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.01	1.0 ± 0.0
GraphSAGE loops A_2	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0

Table 3.12: Precision of the normal nodes for A_2 on several graphs (G_0 - G_5) in the testing set.

Anomalies / Graph	G_0	G_1	G_2	G_3	G_4	G_5
CoBaGAD A_2	0.96 ± 0.01	0.96 ± 0.0	0.94 ± 0.05	0.92 ± 0.05	0.96 ± 0.02	0.97 ± 0.01
CoBaGAD loops A_2	0.93 ± 0.0	0.96 ± 0.01	0.88 ± 0.01	0.86 ± 0.07	0.86 ± 0.06	0.93 ± 0.02
GAT A_2	0.94 ± 0.01	0.91 ± 0.01	0.85 ± 0.02	0.87 ± 0.06	0.85 ± 0.02	0.93 ± 0.03
GAT loops A_2	0.92 ± 0.01	0.91 ± 0.03	0.82 ± 0.01	0.88 ± 0.03	0.8 ± 0.04	0.93 ± 0.01
GCN A_2	0.84 ± 0.01	0.59 ± 0.02	0.63 ± 0.05	0.72 ± 0.01	0.81 ± 0.02	0.82 ± 0.03
GCN loops A_2	0.7 ± 0.01	0.5 ± 0.02	0.64 ± 0.05	0.78 ± 0.06	0.77 ± 0.04	0.68 ± 0.03
GraphSAGE A_2	0.87 ± 0.01	0.89 ± 0.0	0.92 ± 0.01	0.92 ± 0.02	0.83 ± 0.04	0.9 ± 0.0
GraphSAGE loops A_2	0.87 ± 0.02	0.89 ± 0.01	0.92 ± 0.01	0.91 ± 0.01	0.87 ± 0.02	0.86 ± 0.03

Table 3.13: Recall of the normal nodes for A_2 on several graphs (G_0 - G_5) in the testing set.

Anomalies / Graph	G_0	G_1	G_2	G_3	G_4	G_5
CoBaGAD A_3	1.0 ± 0.01	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
CoBaGAD loops A_3	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
GAT A_3	0.98 ± 0.02	0.9 ± 0.08	0.96 ± 0.04	0.81 ± 0.15	0.84 ± 0.08	0.89 ± 0.06
GAT loops A_3	1.0 ± 0.0	1.0 ± 0.01	0.99 ± 0.02	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
GCN A_3	0.58 ± 0.04	0.59 ± 0.09	0.73 ± 0.06	0.67 ± 0.08	0.58 ± 0.11	0.93 ± 0.02
GCN loops A_3	1.0 ± 0.0	1.0 ± 0.0	0.99 ± 0.01	1.0 ± 0.0	1.0 ± 0.0	0.98 ± 0.02
GraphSAGE A_3	1.0 ± 0.0	1.0 ± 0.0	0.99 ± 0.01	1.0 ± 0.0	0.98 ± 0.03	1.0 ± 0.0
GraphSAGE loops A_3	1.0 ± 0.0	1.0 ± 0.0	0.99 ± 0.02	0.98 ± 0.03	0.98 ± 0.03	1.0 ± 0.0

Table 3.14: Recall of the anomalies for A_3 on several graphs (G_0 - G_5) in the testing set.

Anomalies / Graph	G_0	G_1	G_2	G_3	G_4	G_5
CoBaGAD A_3	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
CoBaGAD loops A_3	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
GAT A_3	1.0 ± 0.0	0.99 ± 0.01	1.0 ± 0.0	0.99 ± 0.01	0.99 ± 0.01	0.97 ± 0.03
GAT loops A_3	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
GCN A_3	0.95 ± 0.0	0.93 ± 0.0	0.94 ± 0.0	0.93 ± 0.01	0.95 ± 0.01	0.95 ± 0.01
GCN loops A_3	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
GraphSAGE A_3	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
GraphSAGE loops A_3	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0

Table 3.15: Precision of the normal nodes for A_3 on several graphs (G_0 - G_5) in the testing set.

Anomalies / Graph	G_0	G_1	G_2	G_3	G_4	G_5
CoBaGAD A_3	0.72 ± 0.01	0.83 ± 0.0	0.92 ± 0.01	0.77 ± 0.01	0.86 ± 0.05	0.56 ± 0.02
CoBaGAD loops A_3	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.99 ± 0.01	0.99 ± 0.01	0.99 ± 0.0
GAT A_3	0.68 ± 0.02	0.87 ± 0.07	0.91 ± 0.04	0.79 ± 0.1	0.84 ± 0.12	0.35 ± 0.14
GAT loops A_3	0.99 ± 0.01	0.98 ± 0.0	0.98 ± 0.01	0.95 ± 0.01	0.92 ± 0.0	0.96 ± 0.01
GCN A_3	0.49 ± 0.06	0.37 ± 0.06	0.29 ± 0.05	0.29 ± 0.05	0.47 ± 0.1	0.1 ± 0.01
GCN loops A_3	1.0 ± 0.0	1.0 ± 0.0	0.99 ± 0.01	0.98 ± 0.03	1.0 ± 0.01	0.98 ± 0.0
GraphSAGE A_3	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.97 ± 0.01	0.94 ± 0.06	0.98 ± 0.01
GraphSAGE loops A_3	0.99 ± 0.01	1.0 ± 0.0	0.99 ± 0.01	0.97 ± 0.02	0.95 ± 0.05	1.0 ± 0.0

Table 3.16: Recall of the normal nodes for A_3 on several graphs (G_0 - G_5) in the testing set.

Anomalies / Graph	G_0	G_1	G_2	G_3	G_4	G_5
CoBaGAD A_4	1.0 ± 0.0	0.98 ± 0.02	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
CoBaGAD loops A_4	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
GAT A_4	1.0 ± 0.0	0.99 ± 0.01	0.98 ± 0.02	1.0 ± 0.0	0.98 ± 0.03	1.0 ± 0.0
GAT loops A_4	1.0 ± 0.0	0.99 ± 0.01	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.99 ± 0.01
GCN A_4	0.7 ± 0.02	0.82 ± 0.05	0.88 ± 0.17	0.72 ± 0.2	0.69 ± 0.06	0.7 ± 0.08
GCN loops A_4	1.0 ± 0.0	1.0 ± 0.0	0.99 ± 0.02	0.98 ± 0.03	1.0 ± 0.0	0.99 ± 0.01
GraphSAGE A_4	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.98 ± 0.03	0.98 ± 0.03	0.99 ± 0.01
GraphSAGE loops A_4	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.98 ± 0.03	1.0 ± 0.0	0.98 ± 0.02

Table 3.17: Recall of the anomalies for A_4 on several graphs (G_0 - G_5) in the testing set.

Anomalies / Graph	G_0	G_1	G_2	G_3	G_4	G_5
CoBaGAD A_4	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
CoBaGAD loops A_4	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
GAT A_4	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
GAT loops A_4	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
GCN A_4	0.97 ± 0.0	0.97 ± 0.0	0.99 ± 0.02	0.94 ± 0.06	0.96 ± 0.01	0.97 ± 0.01
GCN loops A_4	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
GraphSAGE A_4	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
GraphSAGE loops A_4	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0

Table 3.18: Precision of the normal nodes for A_4 on several graphs (G_0 - G_5) in the testing set.

Anomalies / Graph	G_0	G_1	G_2	G_3	G_4	G_5
CoBaGAD A_4	0.99 ± 0.01	0.99 ± 0.0	1.0 ± 0.0	0.98 ± 0.01	0.94 ± 0.03	0.95 ± 0.02
CoBaGAD loops A_4	0.99 ± 0.01	1.0 ± 0.0	0.99 ± 0.0	0.96 ± 0.0	0.96 ± 0.02	0.99 ± 0.01
GAT A_4	0.92 ± 0.01	0.97 ± 0.0	0.98 ± 0.0	0.91 ± 0.05	0.93 ± 0.03	0.87 ± 0.03
GAT loops A_4	0.93 ± 0.02	0.98 ± 0.02	0.95 ± 0.02	0.95 ± 0.05	0.92 ± 0.03	0.92 ± 0.02
GCN A_4	0.69 ± 0.01	0.38 ± 0.04	0.18 ± 0.23	0.31 ± 0.07	0.5 ± 0.11	0.67 ± 0.04
GCN loops A_4	0.92 ± 0.0	0.97 ± 0.0	0.97 ± 0.01	0.95 ± 0.02	0.96 ± 0.01	0.88 ± 0.01
GraphSAGE A_4	0.92 ± 0.01	0.98 ± 0.0	0.97 ± 0.0	0.95 ± 0.02	0.96 ± 0.01	0.91 ± 0.01
GraphSAGE loops A_4	0.92 ± 0.01	0.98 ± 0.0	0.98 ± 0.0	0.94 ± 0.02	0.93 ± 0.02	0.89 ± 0.01

Table 3.19: Recall of the normal nodes for A_4 on several graphs (G_0 - G_5) in the testing set.

Anomalies / Graph	G_0	G_1	G_2	G_3	G_4	G_5
CoBaGAD A_5	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.98 ± 0.03	1.0 ± 0.0	1.0 ± 0.0
CoBaGAD loops A_5	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.96 ± 0.03	1.0 ± 0.0	1.0 ± 0.0
GAT A_5	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
GAT loops A_5	0.98 ± 0.03	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
GCN A_5	0.72 ± 0.05	0.81 ± 0.04	0.78 ± 0.11	0.74 ± 0.09	0.6 ± 0.09	0.69 ± 0.07
GCN loops A_5	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.98 ± 0.03	0.99 ± 0.01
GraphSAGE A_5	1.0 ± 0.0	1.0 ± 0.0	0.99 ± 0.01	1.0 ± 0.0	0.98 ± 0.03	0.98 ± 0.03
GraphSAGE loops A_5	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.93 ± 0.05	0.99 ± 0.01

Table 3.20: Recall of the anomalies for A_5 on several graphs (G_0 - G_5) in the testing set.

Anomalies / Graph	G_0	G_1	G_2	G_3	G_4	G_5
CoBaGAD A_5	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
CoBaGAD loops A_5	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
GAT A_5	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
GAT loops A_5	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
GCN A_5	0.97 ± 0.0	0.97 ± 0.0	0.96 ± 0.01	0.96 ± 0.01	0.96 ± 0.01	0.98 ± 0.0
GCN loops A_5	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
GraphSAGE A_5	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
GraphSAGE loops A_5	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0

Table 3.21: Precision of the normal nodes for A_5 on several graphs (G_0 - G_5) in the testing set.

Anomalies / Graph	G_0	G_1	G_2	G_3	G_4	G_5
CoBaGAD A_5	0.99 ± 0.0	0.99 ± 0.0	0.99 ± 0.01	0.96 ± 0.01	0.92 ± 0.05	0.99 ± 0.02
CoBaGAD loops A_5	0.95 ± 0.01	0.98 ± 0.01	0.98 ± 0.01	0.92 ± 0.04	0.93 ± 0.01	0.95 ± 0.0
GAT A_5	0.97 ± 0.02	0.98 ± 0.01	0.94 ± 0.02	0.95 ± 0.01	0.94 ± 0.02	0.93 ± 0.03
GAT loops A_5	0.9 ± 0.03	0.98 ± 0.01	0.97 ± 0.01	0.92 ± 0.04	0.91 ± 0.0	0.91 ± 0.02
GCN A_5	0.64 ± 0.05	0.41 ± 0.03	0.36 ± 0.18	0.41 ± 0.09	0.58 ± 0.02	0.68 ± 0.02
GCN loops A_5	0.88 ± 0.01	0.97 ± 0.01	0.96 ± 0.01	0.96 ± 0.01	0.92 ± 0.05	0.83 ± 0.02
GraphSAGE A_5	0.87 ± 0.02	0.97 ± 0.0	0.97 ± 0.01	0.93 ± 0.01	0.95 ± 0.01	0.85 ± 0.05
GraphSAGE loops A_5	0.88 ± 0.01	0.97 ± 0.0	0.97 ± 0.01	0.94 ± 0.01	0.94 ± 0.01	0.88 ± 0.02

Table 3.22: Recall of the normal nodes for A_5 on several graphs (G_0 - G_5) in the testing set.

Anomalies / Graph	G_0	G_1	G_2	G_3	G_4	G_5
CoBaGAD A_6	0.68 ± 0.04	0.91 ± 0.01	0.91 ± 0.03	0.78 ± 0.05	0.76 ± 0.03	0.72 ± 0.19
CoBaGAD loops A_6	1.0 ± 0.0	1.0 ± 0.0	0.99 ± 0.02	0.98 ± 0.03	0.69 ± 0.08	0.98 ± 0.01
GAT A_6	0.62 ± 0.02	0.92 ± 0.02	0.89 ± 0.03	0.89 ± 0.05	0.8 ± 0.0	0.61 ± 0.21
GAT loops A_6	1.0 ± 0.0	0.99 ± 0.02	0.99 ± 0.01	1.0 ± 0.0	0.76 ± 0.08	0.99 ± 0.01
GCN A_6	0.52 ± 0.01	0.83 ± 0.01	0.79 ± 0.03	0.83 ± 0.08	0.82 ± 0.08	0.54 ± 0.12
GCN loops A_6	0.86 ± 0.02	0.95 ± 0.06	0.86 ± 0.06	0.89 ± 0.0	0.87 ± 0.11	0.87 ± 0.02
GraphSAGE A_6	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.96 ± 0.06	0.99 ± 0.01
GraphSAGE loops A_6	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0

Table 3.23: Recall of the anomalies for A_6 on several graphs (G_0 - G_5) in the testing set.

Anomalies / Graph	G_0	G_1	G_2	G_3	G_4	G_5
CoBaGAD A_6	0.98 ± 0.0	0.99 ± 0.0	0.99 ± 0.0	0.99 ± 0.0	0.98 ± 0.0	0.97 ± 0.0
CoBaGAD loops A_6	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.98 ± 0.01	1.0 ± 0.0
GAT A_6	0.97 ± 0.0	0.99 ± 0.0	0.99 ± 0.0	0.99 ± 0.0	0.98 ± 0.0	0.96 ± 0.0
GAT loops A_6	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.98 ± 0.01	1.0 ± 0.0
GCN A_6	0.96 ± 0.0	0.98 ± 0.0	0.98 ± 0.0	0.98 ± 0.01	0.99 ± 0.01	0.96 ± 0.01
GCN loops A_6	0.99 ± 0.0	1.0 ± 0.01	0.99 ± 0.01	0.99 ± 0.0	0.99 ± 0.01	0.99 ± 0.0
GraphSAGE A_6	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.01	1.0 ± 0.0
GraphSAGE loops A_6	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0

Table 3.24: Precision of the normal nodes for A_6 on several graphs (G_0 - G_5) in the testing set.

Anomalies / Graph	G_0	G_1	G_2	G_3	G_4	G_5
CoBaGAD A_6	0.95 ± 0.06	0.89 ± 0.04	0.93 ± 0.04	0.93 ± 0.01	0.84 ± 0.06	0.64 ± 0.42
CoBaGAD loops A_6	0.99 ± 0.0	0.99 ± 0.0	0.92 ± 0.06	0.95 ± 0.03	0.86 ± 0.04	0.91 ± 0.01
GAT A_6	0.76 ± 0.03	0.88 ± 0.04	0.88 ± 0.01	0.92 ± 0.01	0.79 ± 0.09	0.64 ± 0.36
GAT loops A_6	0.94 ± 0.03	0.95 ± 0.02	0.9 ± 0.02	0.93 ± 0.03	0.87 ± 0.01	0.9 ± 0.01
GCN A_6	0.77 ± 0.03	0.6 ± 0.03	0.67 ± 0.03	0.68 ± 0.09	0.77 ± 0.01	0.68 ± 0.16
GCN loops A_6	0.91 ± 0.02	0.79 ± 0.1	0.76 ± 0.09	0.79 ± 0.02	0.85 ± 0.08	0.93 ± 0.03
GraphSAGE A_6	0.97 ± 0.01	0.95 ± 0.01	0.92 ± 0.01	0.88 ± 0.03	0.85 ± 0.05	0.96 ± 0.01
GraphSAGE loops A_6	0.97 ± 0.0	0.94 ± 0.01	0.93 ± 0.02	0.92 ± 0.04	0.93 ± 0.02	0.96 ± 0.03

Table 3.25: Recall of the normal nodes for A_6 on several graphs (G_0 - G_5) in the testing set.

Introduction

Ces dernières années, les données de type réseau sont devenues omniprésentes et ont suscité un grand intérêt dans la communauté de l'exploration de données. Elles sont utilisées dans de nombreux domaines différents. Les réseaux sociaux tels que Facebook, Twitter ou les cercles sociaux de la vie réelle sont par nature des graphes avec de nombreuses applications potentielles telles que la recommandation d'amis, la recherche de groupes de personnes, de bots ou de spammeurs, de comptes malveillants qui diffusent de fausses nouvelles, d'orateurs haineux, des personnes les plus influentes et même l'étude de la diffusion d'informations ou de maladies. Le Web est aussi un réseau dans lequel nous voulons identifier les hubs et essayer d'améliorer la précision des recherches. Dans les réseaux de télécommunication comme les réseaux 4G et 5G ou les réseaux d'e-mails, on peut vouloir identifier les entités dont le comportement s'écarte de la normalité ou celles qui diffusent des informations malveillantes. La distribution d'électricité, de gaz, d'eau sont aussi des réseaux dans lesquels on veut trouver des anomalies pour prévenir les pannes ou prédire les pics de consommation. Du côté de la sécurité numérique, la propagation de virus, l'intrusion de machines, et la détection d'ordinateurs non autorisés sont des applications de l'analyse de réseau. Les graphes ne cessent de grandir, avec environ 3 milliards d'utilisateurs de Facebook, 200 millions d'utilisateurs quotidiens de Twitter et 45 milliards de pages web. De nouvelles méthodes sont donc nécessaires pour traiter de tels ensembles de données. Récemment, l'exploration de graphes a été révolutionnée par les modèles d'apprentissage automatique. Les plongements de graphes et les réseaux neuronaux sur graphes sont des outils très efficaces pour réduire la complexité de l'analyse des graphes. De plus, ils améliorent considérablement la capacité à effectuer certaines tâches sur les graphes.

D'autre part, la détection d'anomalies est un problème important dans de nombreux domaines d'application. La détection d'anomalies vise à trouver des instances anormales dans les données. Il existe de nombreuses applications possibles telles que la surveillance de la santé avec la détection d'anomalies dans les radiographies ou les électrocardiogrammes, la détection de fraudes, la détection d'événements spéciaux, les défauts dans l'industrie ou les comportements anormaux. Par exemple, on estime qu'il y a entre 25 et 100 milliards d'euros de fraude fiscale en France. En 2020, 25% des contrôles fiscaux ont été automatisés grâce à des outils d'analyse de données. Ainsi, les recettes de l'impôt ont augmenté de 30% entre 2019 et 2020. L'essor des outils de détection des anomalies entraîne des améliorations majeures dans beaucoup de domaines. Il est nécessaire de poursuivre ces développements et d'améliorer les méthodes actuelles. Un autre défi de la détection d'anomalies est le fait qu'elle traite de l'occurrence de cas rares dans les données. Cela implique souvent de traiter des données très déséquilibrées. Le traitement de données déséquilibrées complique également l'analyse des données.

Enfin, l'explicabilité dans le domaine de l'intelligence artificielle, XAI (eXplainable Artificial Intelligence), est devenue très importante avec l'essor des modèles d'apprentissage profond. En effet, l'apprentissage profond a apporté d'énormes améliorations dans l'exploration du texte, la vision assistée par ordinateur ou le traitement du signal. Mais il a également entraîné un

manque de compréhensibilité car la plupart des meilleurs modèles ne sont pas interprétables. Par exemple, les convolutions ou les mécanismes d'attention ne sont pas intrinsèquement compréhensibles. Ainsi, un domaine de recherche émergent est l'intelligence artificielle explicable (XAI) qui se concentre sur la façon de générer des explications significatives qui peuvent améliorer la confiance dans les différents modèles, améliorer la transparence des décisions, éviter les biais des modèles ou des ensembles de données ou simplement permettre de comprendre la sortie d'un outil d'apprentissage automatique.

Cette thèse se situe à l'intersection de ces trois domaines : la fouille de graphes pour la détection d'anomalies et l'explicabilité.

Contexte de cette thèse . Les travaux présentés dans cette thèse ont été réalisés dans l'équipe Data Intelligence du laboratoire Hubert Curien qui est une unité mixte de recherche (UMR 5516) entre l'Université Jean Monnet de Saint-Étienne, l'Université de Lyon, le CNRS et l'Institut d'Optique Graduate School.

Plan de la thèse Ce manuscrit est composé d'un chapitre de mise en contexte suivi de deux chapitres présentant chacun une de nos contributions.

- Le chapitre 1 est une introduction aux différents sujets abordés dans cette thèse. Tout d'abord, nous introduisons le concept de graphe et les outils pour les analyser. Quelques définitions sont données et une présentation générale de l'analyse de réseau, des techniques de plongement de graphes et des réseaux neuronaux sur graphes est fournie. Ensuite, nous introduisons la notion de détection d'anomalies et discutons de l'une de ses principales limites : la malédiction de la dimensionnalité. Enfin, nous donnons un aperçu de la notion d'intelligence artificielle explicable. Les modèles linéaires et les arbres de décision sont introduits pour, ensuite, présenter leurs limites. Des modèles plus précis sont également introduits tels que les réseaux de neurones profonds. Si ces nouveaux modèles d'apprentissage automatique ont amélioré de nombreux aspects du processus de décision, ils ne sont cependant pas intrinsèquement compréhensibles. Pour résoudre ce problème, de nouvelles méthodes d'explication sont nécessaires.
- Le chapitre 2 est consacré à la définition et à la détection des anomalies contextuelles dans les graphes attribués, qui constituent un nouveau type d'anomalies. Les anomalies dans les graphes ne sont pas toujours clairement définies et dépendent souvent de l'application. Par exemple, les lois de puissance ont été largement utilisées dans l'exploration de graphes et les anomalies pourraient être les nœuds qui s'écartent beaucoup de ces lois. Bien que cette définition soit facile à comprendre, elle est spécifique au modèle car le choix des lois est crucial. De plus, ces méthodes ne fournissent pas une définition claire des anomalies car elles ne se concentrent pas sur des nœuds spécifiques du graphe. Dans un cadre non supervisé, où aucune donnée étiquetée n'est disponible pendant l'apprentissage, un modèle ne peut pas se concentrer sur des nœuds spécifiques du graphe puisqu'il ne peut

pas disposer d'informations guidées par l'utilisateur. En revanche, si quelques exemples étiquetés sont fournis au modèle pendant l'entraînement, il peut se concentrer sur les nœuds du même type. Cette approche permet de détecter les nœuds du graphe qui partagent le même type. Ainsi, si nous connaissons déjà quelques anomalies dans le graphe, alors nous pouvons détecter toutes les anomalies du même type. Dans cette thèse, nous avons défini les anomalies contextuelles comme des nœuds du graphe représentés par un contexte local particulier autour de ce nœud. Ensuite, de manière semi-supervisée, ce type d'anomalie peut être détecté grâce à des réseaux de neurones sur graphes avec des mécanismes d'attention. Des expériences intensives prouvent l'efficacité de l'approche et montrent qu'elle surpasse les modèles de pointe semi-supervisés et non supervisés dans la détection d'anomalies contextuelles. L'ajustement des paramètres est également discuté afin que chaque utilisateur puisse facilement détecter ses propres anomalies.

- Le chapitre 3 est consacré à la présentation de notre modèle explicatif de classification sur graphes. L'explicabilité dans le contexte de l'apprentissage automatique s'est considérablement développée au cours des dernières années. L'amélioration des modèles d'apprentissage automatique en termes de vision par ordinateur et de traitement du langage naturel a entraîné un manque de compréhensibilité des méthodes. Ceci est principalement dû à l'utilisation de réseaux neuronaux. Dans le contexte de la classification des nœuds dans les graphes, les modèles explicatifs reposent souvent sur la recherche de nœuds importants du graphe qui ont conduit à une classification spécifique ou de caractéristiques importantes décrivant les nœuds. Une nouvelle méthode est présentée, qui est simple, compréhensible et qui peut faire la distinction entre les nœuds et les caractéristiques pour extraire uniquement les nœuds pertinents et leurs caractéristiques qui impliquent la détection d'anomalies. Les expériences démontrent que ce nouveau modèle peut trouver avec précision les informations pertinentes qui conduisent un modèle profond à une prédiction spécifique. Il surpasse aussi largement les méthodes de l'état de l'art car il est plus précis. Enfin, l'ajustement des hyperparamètres est discuté.

Conclusion

La première contribution principale de cette thèse est la définition, la génération et la détection d'anomalies contextuelles dans les graphes avec des réseaux de neurones sur graphes dans le contexte de données déséquilibrées. Les anomalies contextuelles sont un cas spécifique d'anomalies où les nœuds du graphe présentent un arrangement singulier. Il existe de nombreux types d'anomalies contextuelles en fonction du "contexte". Grâce à notre générateur ConA-Gen, il est possible de créer de nouveaux jeux de données avec des anomalies contextuelles ainsi que la vérité de terrain sur les nœuds qui sont des anomalies. De plus, nous avons présenté CoBaGAD, notre détecteur d'anomalies contextuelles. Avec une couche d'attention de graphe modifiée, nous pouvons construire un réseau neuronal pour classer les nœuds en deux catégories

: nœuds normaux et anomalies. Grâce à des expériences intensives, nous montrons que notre modèle peut surpasser les méthodes de classification sur graphes de l'état de l'art d'une importante marge. Bien que notre modèle présente de très bonnes performances, nous aimerions aborder certaines améliorations possibles. Tout d'abord, notre générateur donne à chaque nœud un vecteur de caractéristiques de type one-hot. C'est-à-dire que les nœuds d'un graphe ne sont décrits que par un seul attribut catégorique (qui peut être considéré comme une couleur). Ainsi, une amélioration naturelle de notre générateur est de passer à la création de caractéristiques plus complexes comme des attributs catégoriels multiples ou des attributs numériques multiples. Ces deux configurations sont plus proches des cas réels. Cette modification des caractéristiques impliquera également que nous puissions définir des anomalies contextuelles plus complexes, ce qui constituerait un nouveau défi à relever pour CoBaGAD. Bien que les anomalies contextuelles puissent devenir plus complexes, CoBaGAD devrait rester robuste aux changements et ne devrait pas être modifié, à part peut-être l'ajustement de quelques paramètres comme le nombre de couches pour avoir un plus grand champ de vision. Deuxièmement, nous aimerions pouvoir fournir un score d'anomalie. En effet, dans le contexte de la fraude, par exemple, les experts ne se concentreront que sur les fraudes les plus importantes et les plus probables. Ainsi, il pourrait être bénéfique d'avoir une technique de classement qui puisse trouver les anomalies les plus pertinentes dans le réseau.

La deuxième contribution de cette thèse est l'explication de la prédiction d'un modèle boîte noire qui effectue des inférences sur des données de type graphes. Notre modèle appartient à la famille des méthodes de perturbation dont le but est de générer de nouvelles données autour d'une instance spécifique pour apprendre un modèle explicable local. Pour un nœud spécifique, notre méthode s'appuie sur un échantillonnage multiple du voisinage du nœud et le perturbe pour créer un nouveau jeu de données. Ces exemples perturbés sont introduits dans le modèle de la boîte noire pour produire une étiquette. Grâce à ces exemples perturbés associés et à leurs étiquettes, nous pouvons entraîner localement un arbre de décision, *i.e.* pour ce nœud spécifique. Le choix de l'arbre de décision n'est pas trivial car le processus de décision d'un arbre est compréhensible par l'homme et il fournit également un score d'importance pour chacune des caractéristiques d'entrée. Ensuite, nous pouvons utiliser ces scores pour trouver les caractéristiques pertinentes qui ont conduit le modèle boîte noire à une décision spécifique. Grâce aux anomalies contextuelles qui fournissent une vérité terrain naturelle pour expliquer la classification des nœuds, nous avons mené de nombreuses expériences pour retrouver le contexte qui a défini les anomalies.

Nous pensons que notre travail a beaucoup d'applications potentielles. La détection des anomalies est très importante dans de nombreux domaines tels que la détection de fraudes sur les marchés publics ou l'évasion fiscale, la sécurité avec les intrusions dans les réseaux, la finance avec la fraude à la carte de crédit, la détection des logiciels malveillants, la publicité mensongère ou encore la propagation de fausses nouvelles. De nombreuses activités frauduleuses existent et des méthodes toujours plus complexes sont développées pour tenter de ne pas se faire prendre.

Ainsi, de nouveaux outils sont également conçus pour détecter les nouvelles formes d'anomalies. Cela peut être réalisé à l'aide de puissants algorithmes d'apprentissage automatique. Mais les méthodes d'apprentissage automatique ont également entraîné un manque de compréhension du fonctionnement des algorithmes. Ceci est principalement dû au grand nombre de paramètres qu'ils utilisent. Il est donc crucial d'expliquer le processus de décision d'une méthode. Cela se fait par le biais de l'intelligence artificielle explicable, qui est un ensemble de nouvelles méthodes visant à donner un sens au processus d'inférence d'une méthode d'apprentissage automatique.

La détection des anomalies et l'intelligence artificielle explicable sont toutes deux nécessaires pour rendre le monde plus juste. Plus juste en trouvant les personnes frauduleuses grâce à la détection des anomalies et plus juste en mettant l'intelligence artificielle à la portée de tous grâce à XAI. Le pouvoir prédictif des algorithmes d'apprentissage automatique doit être bénéfique pour tous, mais aussi compréhensible pour tous.

Remerciements

Je voudrais avant tout remercier Christine et Baptiste pour avoir encadrer mes travaux de recherche au cours de ces trois dernières années et sans qui rien n'aurait été possible. Leur encadrement m'a permis de grandir à la fois scientifiquement mais aussi humainement. Scientifiquement grâce à leur expertise, à leur excellente connaissance du domaine, à leur approche rigoureuse de la science bien faite et à leur capacité d'ouverture et d'intégration des différentes approches de graph mining. Mais aussi humainement grâce à l'apprentissage de la bonne conduite d'un projet de recherche en passant par chaque étape depuis l'établissement d'un sujet jusqu'à l'écriture d'un papier. Baptiste, tu es toujours prêt à aider, à échanger des idées, et à apporter un éclairage sur un sujet scientifique pointu. Christine, tu as su me donner un cadre scientifique précis pour pouvoir avancer dans la bonne direction et me soutenir lors des différentes épreuves que constitue la thèse de doctorat. Baptiste et Christine, vous faites vraiment une merveilleuse équipe et j'ai passé de très bons moments à travailler avec vous au cours de ces trois dernières années. J'aimerais aussi remercier toute l'équipe Data Intelligence et le laboratoire Hubert Curien pour m'avoir accueilli durant ma thèse et tout fait pour avoir de bonnes conditions de travail malgré l'obstacle du COVID.

J'aimerais remercier aussi les membres du jury qui ont relu et évalué mes travaux de thèse et plus particulièrement ce manuscrit. Merci à Márton Karsai et Eric Gaussier pour l'attention particulière qu'ils ont apporté à ma thèse de doctorat. Merci à Nidhi Hegde d'avoir apporté son expertise sur le sujet. Et merci à Christophe Gravier d'avoir suivi l'avancée de mes travaux de recherche au cours de ma thèse.

Je porte aussi une attention très spéciale au soutien que m'a procuré ma famille. Merci à mes parents, Alain et Marie-Agnès, de m'avoir toujours soutenu dans mes études et de m'avoir aidé à pouvoir les réaliser dans d'excellents conditions. Vous avez toujours été intéressés malgré la difficulté d'appréhension du sujet et toujours disponible quand il le faut. Merci aussi à ma

chérie, Cybèle, pour me supporter au quotidien malgré la fatigue et le stress parfois. Cette thèse ne serait pas la même sans toi. Merci à mes frères, Quentin, Robin et Vincent, pour leur curiosité et leur soutien indéfectible. Merci à ma belle-famille pour leurs encouragements et leur aide morale. Et enfin, merci à mes amis pour leur bonne humeur et les moments de réconfort qu'ils m'ont apporté et qu'ils m'apportent encore.