UNIVERSITÀ DI PISA

DIPARTIMENTO DI INFORMATICA

Corso di Laurea Triennale in Informatica

CF-GNNExplainer++

An Extension on Counterfactual Explanations for Graph Neural Networks

Supervisors:

Anna Monreale

Riccardo Guidotti

Candidate:

Antonello Dettori

Academic year 2022/2023

Contents

Chapter 1

Introduction

The last couple of decades have seen an explosive growth of the field of machine learning, and more specifically the sub-field of deep learning, with a series of breakthroughs, applications in different domains of knowledge and frequently reaching the eyes of the non-expert consumer of generalist news media. These recent advancements are due to different factors: the continued research of experts in the different fields that compose machine learning, the increase in available computational power and the availability of vast amount of data [15]. The computational resources aspects is particularly important due to the ever increasing size and complexity [12] of the statistical models used to solve the required tasks. The need for complexity can be explained by the increasingly information-rich data that is being handled, the need for more specialised model architectures to handle different data structures, among which vectors, images, series and graphs, and also due to the need for bigger models in terms of parameters to extract better computer-friendly representations of the data at hand, the essence of modern deep learning.

The research interest in ML applied to graphs in particular has seen a resurgence in the last few years, well after the foundational works of Sperduti and Starita [13], due to the expressivity of this data structure. Graphs are interesting due to their ability to represent relational information between entities, making them ideal in scenarios such as the study of users in social networks of any kind, chemistry and biology, to name a few and are central to the method explored in this work. Deep learning successes, however, have also come at various costs, some more obvious such as the increased monetary costs incurred to obtain more powerful hardware to keep up with the demands of bigger models and the environmental costs due to the amounts of energy needed to train a single model [12], but also costing in terms of the transparency of the process that a model goes through to obtain its results.

The problems of interpreting a model's behaviour and results are explored by the field of Explainable AI (XAI) [7], which aims to find methods that, given a model, are able to confidently explain to a user how it came to its conclusions. This field of research becomes ever more relevant due to the fact that the most promising results are usually accompanied by the use of models that are difficult for humans to

understand and analyze without external tools. Therefore it seems for now inevitable that the increase in predictive power comes at a cost of an increased opaqueness of the decision process of the model. These opaque models are sometimes referred to as "black-box", this term is used to distinguish between models such as neural networks from other more transparent ones such as decision trees or rule-based ones, which allow for an easier interpretation of the process by humans but that, in most cases, do not offer the same level of performance.

The need to make sense of the results given by these models can be justified in multiple ways, among which:

- Experts need the ability to understand why a model behaves the way it does in order to be sure that the *reasoning* is correct and the results are not just incidental. For example, by showing it a series of images of birds perched on tree branches, a model might learn to identify trees in the image while ignoring the birds which are the relevant target.

- In order to exploit the full potential offered by this technology the public at large needs to be able to trust and rely on it in their daily lives. This is particularly important when the sensitive characteristics (race, sex, gender, religion) of a person are fed to the model.

- Experts need to assess the correctness of a model in order to prevent the exploitation of potential vulnerabilities by malevolent actors.

- Explaining an accurate model has the potential of being a useful exploratory tool to expand other frontiers of human knowledge.

The need for accountability for the decisions taken and the handling of sensitive data, such as medical one, makes XAI an important part of the vision of democracy of the European Union, which informs the content of the laws proposed by the European Commission, such as the General Data Protection Regulation (GDPR) [6].

An intuitive approach to tackle the explainability problem is to generate counterfactual explanations of the model and its outcome. A counterfactual explanation consists of, given a problem instance and a model's solution for it, finding the minimum number of modifications to the problem needed in order for the model to propose an alternative result [1]. For example, in order to qualify for a bank loan a person needs to have at least some of these desiderata for the institution to trust that the amount lent will be payed back: a certain minimum income, a sufficient level of education or property to use as collateral. Therefore to get their loan approved the user has to either get a better job, get a better degree or acquire some property. Note that depending on the individual characteristics of the user, some of these changes might not be *actionable* making it impossible to get the desired result, however the explanation can still be useful in explaining the process.

Counterfactual explanations are at the core of the method CF-GNNExplainer proposed in [9] which is the starting point of the work presented in this thesis. CF-GNNExplainer aims to produce a counterfactual explanation by removing a minimal number of relations among the ones present in the graph. This thesis presents a variant of the original approach which differs in two aspects. First, it introduces the possibility of perturbing the graph's relations by also adding edges between entities previously not linked. Second, it adds the ability to generate pertinent positive (PP) and pertinent negative (PN) type of explanations proposed in the Contrastive Explanation Methods (CEM) framework by Dhurandhar et al. [5].

Pertinent negatives are very similar to counterfactual explanations, the main difference is that the only type of operation permitted in finding the minimal change is an addition of *features*, nothing can be removed from the starting instance, in this case *features* refers to the connections in a graph. The aim is to find the features of the instance whose *absence* is necessary to be able to keep the model's result the same. *Pertinent positives* instead aim to find the smallest set of features whose *presence* is sufficient to classify an instance the same as its current class. Both of these type of explanations are useful in getting a better understanding of the model's decision making and how it is possible to affect it. In order to clarify the intuition behind these two notions, consider the following non-graph related example. In a case of criminal law, in order to prove that the defendant is innocent it is sufficient, ideally, to show that the defendant was not near the scene of the crime when the event took place (PP) or that the DNA evidence does not match the defendant's one (PN, the absence of a positive DNA test result).

This changes allow for more compact explanations for the user, making it easier to understand if the underlying model conforms to their expectations. The ability of adding links makes it possible for the explainer to find a smaller number of changes needed, compared with the case in which only deletions are allowed. Pertinent positives, meanwhile, allow for a different point of view on the model's behaviour by showing only what it thinks is strictly necessary to achieve the proposed results, the result of this search for a minimal set of relevant elements in the graph also results in small explanations.

In order to validate the proposed method experiments have been conducted on a total of four datasets, three of which are synthetic and a real-world one [4]. The results show that counterfactual explanations generated using both addition and removal of edges result in better results, in terms of size and ability to explain difficult predictions, in the majority of datasets. The newly introduced Pertinent Positive problem formulation seems also a promising way to get a more complete understanding of a model, apart from the CF approach, and is able to generate explanations while removing up to 98% of the edges of the graph.

The thesis is organized as follows. Chapter 2 contains a brief recap of the concepts necessary to understand the proposed approach. Chapter 3 proposes a survey of some of the relevant works on the problem of explanation generation on graphs. A formal definition of the changes proposed in this thesis is presented in Chapter 4,

while a discussion of the experiments' results is presented in Chapter 5. Finally, Chapter 6 concludes the thesis with a discussion on future works.

Chapter 2

Background

2.1 Machine Learning

The term machine learning (ML) is used to broadly refer to the field of scientific research that studies how to *teach* computers to solve a variety of tasks. ML is a joint effort of different domains knowledge, ranging from natural sciences such as computer science, mathematics and biology to social sciences like sociology and psychology. Generally, in order to teach a computer to solve something in this paradigm, the scientist has to provide the machine with:

- A task: a problem to be solved, for which an appropriate measure of error and experiences need to be provided.
- Experiences: a dataset containing useful information to solve the task at hand and from which the computer can learn.
- A model: a mutable structure used by the computer to process the given experiences
- A measure of error: a way for the computer to tell if the solution that it proposes is correct or not.
- A learning algorithm: the instructions on how to adjust the model in case it is not already able to solve the task applied on the available experiences.

This approach differs dramatically from the classical approach of computer science, consisting in the formulation and study of *algorithms*, sequences of instructions to perform to complete a task, that are then applied to specific problems. It instead tackles problems by using the available experiences and by optimizing the measure of error to automatically find a solution. Following is a more in detail rundown of the components needed for machine learning.

2.1.1 Experiences

Experiences, in the setting of a classification task, is a set of N tuples

$$\mathbb{D} = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1...N}$$

where $\mathbf{x} \in \mathbb{R}^i$ represents the input that is given to the model and $\mathbf{y} \in \mathbb{Y}$ is the corresponding label of the input instance, that the model has to learn to predict.

2.1.2 Model

When talking about models is this work, the main focus is on neural networks (NN). A neural network, specifically a feedforward one, can be described as a function $f(\mathbf{x}, \theta)$ where $\mathbf{x} \in \mathbb{R}^i$ is the given input vector and θ are the parameters that need to be modified in order for the model to learn to solve the task at hand. The output of the model varies according to the task chosen, in the case of classification tasks the output is a vector $\hat{\mathbf{y}} \in \mathbb{R}^{|\mathbb{Y}|}$ of length equal to the number of unique labels in the given data $\mathbb{Y} \subset \mathbb{Z}$, and each component represents the confidence of the network on assigning the corresponding label to the given input instance $\mathbf{x}$.

One of the main concepts used in neural architectures is the idea of *layer*, which can be represented as n different functions f_i, each one with their separate matrix of parameters θ_i, that when composed together form the final model $f(\mathbf{x}) = f_n(\mathbf{x}, \theta_n)$ with $f_i(\mathbf{x}, \theta_i) = f_{i-1}(\mathbf{x}, \theta_{i-1})$. Each of these layers takes as input the output of the previous one up until the final layer, whose output $\mathbf{h}_n \in \mathbb{R}^{|\mathbb{Y}|}$ is a representation of the information given as input to the model, transformed in a way that the network has learned to be beneficial to solving the given task.

A standard example of a single layer neural network is the following:

$$\mathbf{h}_i = relu(\mathbf{h}_{i-1} \cdot \theta_i)$$

where $relu$ (rectified linear unit) is applied component-wise to the dot product:

$$relu(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$$

The number of layers, the type of parameters and other decisions regarding the architecture of the model are decided through the use of hyperparameters, which are special parameters that are not learned by the model and instead are given by the ML scientist.

2.1.3 Error Function

An error function $\mathcal{J}$ is used to measure the discrepancy between the prediction $\hat{\mathbf{y}}$ that the model is most confident on and the label $\mathbf{y}$ included in the experience dataset, where $\hat{y} = \max_i \mathbf{y}_i$ and $\mathbf{y}_i$ is the i-th component of the model's output vector $\mathbf{y}$.

The error function used is as follows:

$$\mathcal{J} = \mathbb{E}_{(\mathbf{x},\mathbf{y})\in\mathbb{D}}[\mathcal{L}(\mathbf{x},\mathbf{y},\theta)]$$

where, in this case, $\mathcal{L}(\mathbf{x},\mathbf{y},\theta) = -\mathbb{1}_{\hat{y}=\mathbf{y}} \cdot log(p_{model})$ is a loss function on the single data-point discrepancy, called Cross Entropy (CE), and $\mathbb{1}$ is an indicator function. CE is used to measure the distance between two probability distributions. In this case, the two distributions are the empirical distribution given by the dataset and the distribution learned by the model $p_{model}(x) = softmax(f(x))$ where

$$softmax(x) = \frac{e^{\hat{\mathbf{y}}}}{\sum_{i=1}^{|\mathbb{Y}|} e^{\mathbf{y}_i}}$$

Note that this last formulation of CE is also referred to as Negative Log-Likelihood (NLL). More succinctly, this means that, through the training process, the model learns to change it's internal distribution to fit the results outlined by the experiences.

2.1.4 Learning Algorithm

A learning algorithm is a description of the sequence of steps to perform in order to adapt the parameters θ of the model to the experience that is presented. Here the chosen algorithm is stochastic gradient descent (SGD), which starting from the error function value determines the impact of the parameters of each layer on the final prediction and uses these values to decide which components of θ need to be changed the most.

The algorithm works by computing the gradient of the error function with respect to the parameters θ of the model:

$$\nabla_\theta \mathcal{J}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \mathcal{L}(\mathbf{x}^{(i)},\mathbf{y}^{(i)},\theta)$$

The model's parameters are then updated:

$$\theta = \theta - \alpha \cdot \mathcal{J}(\theta)$$

where $\alpha \in [0,1]$ is an hyper-parameter that controls how much the parameters will be changed in each update. These updates are applied across all layers of the NN by using the backpropagation technique [11].

2.1.5 Training Process

The general workflow of training a model consists of processing the experiences using the model, taking the predictions given as output and comparing them with the labels $\mathbf{y}$ in the dataset, using the error function $\mathcal{J}$ to measure the discrepancy. Once the result of the error function is known, the learning algorithm can be applied to adapt the model's parameters θ to the given experience.

2.2 Graphs

Graphs are data structures that are used due to their ability to represent relational information between entities. The entities are referred to as *nodes* while the connections between them are represented using *edges* between nodes. The purpose of applying models on graphs is to manage to extract a computer-friendly representation of the relational information contained inside them to then apply additional techniques or models in order to solve the target task. A graph $\mathbf{G}$ is defined by the tuple ($\mathbf{V}$, $\mathbf{E}$), where $\mathbf{V}$ is the set of nodes belonging to the graph and $\mathbf{E} \subseteq \mathbf{V} \times \mathbf{V}$ is the set of edges connecting the nodes among themselves. A graph can be undirected or directed. The former meaning that the pair of nodes $v_1, v_2 \in \mathbf{V}$ that describe an edge are unordered $\mathbf{E} \subseteq \{\{v_1, v_2\} : v_1, v_2 \in \mathbf{V}\}$, the latter instead meaning that the pair is ordered $\mathbf{E} \subseteq \{(v_1, v_2) : v_1, v_2 \in \mathbf{V}\}$. Undirected edges can be represented as directed ones by including an edge for both directions. Given a node v_1, it's *neighbourhood* can be defined as the sets of nodes that are connected to v_1 by an edge: $v_2 : (v_1, v_2)$ or $(v_2, v_1) \in \mathbf{E}$.

In ML, graphs can be represented using matrices in the following way:

- An adjacency matrix $A \in \{0, 1\}^{N \times N}$ representing the edges, where N is the number of nodes in the graph. A value of 1 indicates the presence of an edge between the two nodes, otherwise a 0 will be present.
- A node features matrix $X \in \mathbb{R}^{N \times F}$ representing the features associated with each one of the N nodes in the graph, where each node has F features.

2.3 Neural Networks On Graphs

In the form described in Section 2.1.2, neural networks cannot be easily applied to graphs. New architectures need to be defined in order to extract a useful representation from graph structures. Here the idea of a layer consists of a process that produces a local representation for each node v by combining the features of its neighbourhood $\mathcal{N}(v)$. Multiple layers can again be stacked to increase the ability of the network to extract good representations and furthermore to increase the range in which local information is diffused to other nodes beyond the node's neighbourhood $\mathcal{N}(v)$. Due to the similarities shared with the feedforward neural networks previously introduced, this approach is also called *feedforward*. This family of architectures can be described via the *message passing* formalism [2], composed of the following two steps:

- Message dispatching: for each node v, compute the node's message containing information on its state/representation and dispatch it to the nodes in its neighbourhood $\mathcal{N}(v)$.
- State update: for each node v, collect the messages received from the nodes in $\mathcal{N}(v)$ and use them to update its state.

The models of interest in this work can be represented as $f(A, X; W)$ where $A \in \{0,1\}^{N \times N}$ and $X \in \mathbb{R}^{N \times F}$ are the inputs of the model and respectively the adjacency matrix and the matrix of the node's features, W are the model's parameters and y is the output vector representation of the input graph.

Following are the single layer versions of the two models used in the experimental part of this work.

Graph Convolutional Networks (GCN) [8]:

$$f(A, X; W) = softmax[\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}XW]$$

where $\tilde{A} = A + I$, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, and I is the identity matrix of size N.

Graph Attention Networks (GAT) [16]:

$$f(A, X; W)_i = softmax(\max_j [\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}XW]_j)$$

where $f(A, X; W)_i$ is the i-th component of the representation vector.

Note also that classification tasks on graphs can either consist of using a model to label the elements in a given set of graphs or in the set of nodes contained in a single graph. As usual, the dataset needs to be provided already labeled according to some property of interest. An example of this is using a model to predict the mutagenic effects of a molecule given as input, this is the case of the MUTAG dataset [4] used in Chapter 5.

2.4 Explanations

The work proposed in this thesis focuses on the design of approaches to explain a model's results. In particular, the proposed method is part of the family of *black-box* explainers, meaning that starting point is a model that is not interpretable by humans, and more specifically is a form of *outcome explanation* which means that the generated explanation can be used to explain only the targeted instance of the problem and not on any possible one [7].

An explanation of a model applied to graphs can be characterized by the tuple $v = (\mathbf{x}, A)$, where A is the adjacency matrix of the graph following the perturbation applied to change the prediction of the model and $\mathbf{x}$ are the features of the instance for which the explanation was required, which can be either an instance of a graph in case of a graph classification task or a node in case of a node classification.

Each type of explanation explored in this work can be formulated as an optimization problem using an opportunely defined loss function, starting from the following template:

$$\mathcal{L} = \mathcal{L}_{pred}(v, \bar{v}|f, g) + \beta \cdot \mathcal{L}_{dist}(v, \bar{v})$$

where $\mathcal{L}_{pred}$ is a loss term that measures if $f(v) = f(\bar{v})$ or not depending on the explanation required, $\mathcal{L}_{dist}$ measures the number of changes that have been applied to the explanation, f is the original model and g is the model used to optimize the chosen explanation loss [9].

The complete formulations are outlined in Chapter 3 and 4.

Chapter 3

State of the Art

This chapter presents a brief list of the works in the field of XAI applied to Graph Neural Networks that informed the changes presented in this thesis, followed by a formal introduction to the method proposed by Lucic et al. in [9].

3.1 Explainability methods for GNNs

One of the most impactful contributions is GNNExplainer [17], in it the authors present a method by which to generate an explanation of any model applicable to graphs that takes into account the effects on the classification of both the node features X and the edges present in the graph A. In order to obtain compact explanations, this method asks the user to choose an upper limit on the number of nodes allowed in the explanation. This idea has been further expanded upon by Zhang et al. in RelEx [18], which tackles the problem of applying GNNExplainer to a scenario in which the user has no access to the internal parameters of the model, which is a common scenario for Machine learning as a service (MLaaS). It also introduces the idea of explanation diversity which will be revisited in Chapter 4. Another interesting approach is the one explored in MEG by Bacciu et al. [10], where deep reinforcement learning techniques are used to generate counterfactual explanations for molecule-related problems, by penalizing any explanation that violates the laws of chemistry the authors teach the model to generate explanations that are also valid molecules.

3.2 CF-GNNExplainer

CF-GNNExplainer [9] differs substantially from the previous approaches: whereas in GNNExplainer the explanation consisted of assigning responsibility for the prediction to the edges/features of the graph and finding their most relevant subset, in [9] the objective is to generate a counterfactual explanations by perturbing the problem graph's edges. This approach ignores the contribution given by node features and focuses only on the presence of edges between nodes by progressively removing them

until the model's prediction changes. The final explanation, obtained after multiple iterations of the process in order to find a good result, consists of the minimal set of changes introduced with respect to the original graph that resulted in a different prediction. Depending on various factors such as domain knowledge or complexity of the explanation, not all results will be valid or acceptable therefore the explainer exposes different options in order to be able to tweak the results. The result is then handed over to the final user.

In order to highlight the differences with the additions introduced in this work, the explainer will be characterised along two distinct axes. The first axis refers to the way in which the differentiable perturbation matrix $\hat{P} \in \mathbb{R}^{N \times N}$ is transformed into the discrete one $P \in \{0,1\}^{N \times N}$. The second is the formulation of the operations that are used to perturb the instance to explain: the addition and removal of edges. First, however, a brief introduction to the method is necessary.

3.2.1 Generating Counterfactual Explanations

In order to generate an explanation, given a model f and an input instance $v = (\mathbf{x}, A)$ to explain, CF-GNNExplainer uses a new model g and a loss function $\mathcal{L}$ tailored for the type of explanation required. The idea behind using the new model g is to keep the same architecture and reuse the weights of f in order for the explanation to be faithful the model's decision making process. The model g weights are then frozen, since there is no more learning to be done at this stage, and the model is applied to both the original $\mathbf{y} = g(\mathbf{x}, A; W)$ and the perturbed $\hat{\mathbf{y}} = g(\mathbf{x}, \hat{A}; W)$ instances, where $\hat{A}$ is the perturbed adjacency matrix and its definition depends on the perturbation matrix P and the formulation of the operations in use. Finally, the predictions y and $\hat{y}$ are used to compute the explanation's loss function $\mathcal{L}$ and the explanation is generated by iteratively optimizing the elements of the perturbation matrix $P \in \{0,1\}^{N \times N}$ with respect to the loss $\mathcal{L}$ using the stochastic gradient descent (SGD) algorithm:

$$P^{(i+1)} \leftarrow P^{(i)} - \alpha \nabla_P \mathcal{L}$$

where $(i+1)$ is the current iteration of the algorithm and (i) the previous one.

The loss function for counterfactual explanations includes two terms:

- Negated negative log-likelihood term: used to measure the error between the current prediction and the expected prediction of the model, since the required explanation is a counterfactual one. The term is zeroed by an indicator variable if the predictions differ.
$$-\mathbb{1}_{\mathbf{y}=\hat{\mathbf{y}}} \cdot NLL(\mathbf{y}, \hat{\mathbf{y}})$$

- Distance term: used to minimize the amount of changes to the adjacency matrix required to achieve a different prediction. In case of undirected graphs the term

is halved.

$$\sum_{i=0}^{N}\sum_{j=0}^{N}|(\hat{A}-A)_{ij}|$$

Here is the complete loss:

$$\mathcal{L}=-\mathbb{1}_{\mathbf{y}=\hat{\mathbf{y}}}\cdot NLL(\mathbf{y},\hat{\mathbf{y}})+\beta\cdot\sum_{i=0}^{N}\sum_{j=0}^{N}|(\hat{A}-A)_{ij}|$$

3.2.2 P Matrix Discretization - *Standard* Approach

The learning process relies on applying a gradient-based learning algorithm to the model g, it is therefore required that the model be differentiable and that the parameters be real numbers. However the perturbation matrix P, which represents the elements of the adjacency matrix that need to be removed, needs to be discrete. The *standard* approach, introduced by the original work [9], uses two matrices P: a differentiable one $\hat{P}$ and a discrete one P. The discrete one P is obtained by applying the sigmoid function $\sigma(x)=\frac{1}{1+e^{-x}}$ element-wise to $\hat{P}$, then the threshold function

$$thr(x)=\begin{cases}1 & x\geq 0.5\\ 0 & else\end{cases}$$

is applied to the result. The differentiable matrix is used to compute the loss function and to optimize the model, the non-differentiable one instead is used to produce the final version that is given to the user.

3.2.3 Operations - *Original* Formulation

This formulation allows for two types of *mutually exclusive* operations:

- Edge deletion: $\hat{A}=P\odot A$

  The matrix P is initialised to be all 1s in order to have the first prediction be the same as the original one and then progressively zeroed out by optimizing the loss

- Edge addition/deletion: $\hat{A}=P$

  The perturbation matrix P is directly used as the adjacency matrix given as input to the model, allowing the model to add edges that were previously not present.

  Note: this operation was included in the implementation of [9], but not mentioned in the paper itself. It seemed, however, interesting enough to warrant some exploration.

Algorithm 1 CF-GNNExplainer: *original* operations formulation and *standard* approach to discretization of the P matrix. This represents the version of the algorithm as introduced in [9].

Input: instance $v = (\mathbf{x}, A)$, a trained model f, an explainer model g, loss function $\mathcal{L}$, learning rate α, distance hyper-parameter β, the number of iterations k and the set of allowed operations Ops.

Output: Instance explanation $\bar{v}$

$y \leftarrow f(v)$ // Save the original model's prediction
$\hat{P}^{(0)} \leftarrow InitPStd(Ops)$ // Init values depend on the allowed operations
for all $i \in range(k)$ **do**
    $\bar{v} \leftarrow GenExpl(v, \hat{P}^{(i)})$ // Generate a new explanation for v
    $\mathcal{L} \leftarrow \mathcal{L}(g, v, \bar{v}, \beta)$ // Compute the new loss
    $\hat{P}^{(i+1)} \leftarrow \hat{P}^{(i)} - \alpha \nabla_{\hat{P}} \mathcal{L}$ // Update the differentiable P matrix
end for
return $\bar{v}$

function GenExpl($v, \hat{P}$)
    $P \leftarrow thr(\sigma(\hat{P}))$ // Discretize matrix $\hat{P}$
    if $add \in Ops$ **then**
        $\hat{A} \leftarrow P$ // Add and remove edges from A
    else if $del \in Ops$ **then**
        $\hat{A} \leftarrow P \odot A$ // Remove edges from A
    end if
    $\bar{v}_{cand} \leftarrow (\mathbf{x}, \hat{A})$
    if $f(v) \neq f(\bar{v}_{cand})$ **then** // Check if CF generation was successful
        if $\mathcal{L}_{dist}(v, \bar{v}_{cand}) \leq \mathcal{L}_{dist}(v, \bar{v}^*)$ **then**
            $\bar{v}^* \leftarrow \bar{v}_{cand}$ // Save new best explanation
        end if
    end if
    return $\bar{v}^*$
end function

Chapter 4

Extending CF-GNNExplainer

This chapter introduces the new developments compared to the original method CF-GNNExplainer [9]: the *Bernoulli* approach to matrix discretization, the *delta* definition of the operations and the formulation used to generate Pertinent Negative (PN) and Pertinent Positive (PP) explanations. The changes proposed here aim to give the user more options with regards to the combination of operations allowed and the ability to generate explanations belonging to the CEM framework applied to graphs [5]. Note that unless otherwise specified, everything present in Section 3.2.1 still holds true for this chapter, including the loss function used.

4.1 P Matrix Discretization - *Bernoulli* Approach

The *Bernoulli* approach models each edge in the adjacency matrix as a *random variable* following a Bernoulli distribution and learns the best parameter for each one using stochastic gradient descent (SGD). Even though it is possible to take a *random sampling* approach in order to get the final edges' configuration, in practice the maximum likelihood sampling is preferred: the threshold function $thr(x)$ is directly applied on each Bernoulli's parameter. The main difference with the *standard* approach is that the function used to map the parameters is no longer a sigmoid but an identity function and that there is only one matrix P to handle. In order to train the parameters and avoid having to deal with non-differentiable functions, both the clipping and threshold operations are skipped for the purpose of performing backpropagation, following the pass-through approach to estimating a Bernoulli parameter outlined in [3].

4.2 Operations - *Delta* Formulation

In order to decouple the addition and deletion operations, a new variable δ is defined, representing the difference between the perturbed adjacency matrix $\hat{A}$ and the original one A. The idea is to use the adjacency matrix A to bound the values of $\hat{A}$ to the set $\{0, 1\}$, therefore avoiding either removing an edge that is already missing

or adding an already present one. In this formulation, the perturbation matrix P's entries are initialised to 0.

The operations are implemented as follows:

- Edge deletion: $\hat{A} = A + \delta_+$ where $\delta_- = -A \odot P$
- Edge addition: $\hat{A} = A + \delta_-$ where $\delta_+ = (\mathbf{1} - A) \odot P$

The variables δ_+ and δ_- can be added together to perform both operations simultaneously.

4.3 Contrastive Explanations

This section introduces an adaptation of the type of explanations introduced in [5] to the context of graphs. Due to the need to restrict the operations allowed on the graph to additions only, PN explanations cannot be implemented using the *original* operations formulation presented in 3.2.3, therefore in this section both PP and PN are presented using the *delta* approach.

4.3.1 Pertinent Negative

The pertinent negative formulation, due to their inherent similarities, is identical to the previously introduced counterfactual one when only edge addition operations are allowed. This can be seen by recalling that to generate a PN explanation means finding a minimal set of the instance's features whose *absence* is necessary in order for the model's prediction to stay the same, while a CF explanation is the minimal set of changes that are needed for the prediction to change. The same set of features can be seen as either a CF or a PN depending on the point of view of the user.

Formally, when working with graphs and given an instance $v = (\mathbf{x}, A)$ to explain, this can be formulated as finding the set of edges, represented using a matrix $\delta \in \{0, 1\}^{N \times N}$, such that the results obtained by applying the model on the original and perturbed instances are different $f(\mathbf{x}, A) \neq f(\mathbf{x}, A + \delta)$. The *minimal* aspect of the PN definition and the requirement to find a δ perturbation that changes the model's prediction can then be formulated by using the same loss terms introduced in Section 3.2.1 for counterfactual explanations, namely:

$$\mathcal{L} = -\mathbb{1}_{\mathbf{y}=\hat{\mathbf{y}}} \cdot NLL(\mathbf{y}, \hat{\mathbf{y}}) + \beta \cdot \sum_{i=0}^{N} \sum_{j=0}^{N} |(\hat{A} - A)_{ij}|$$

4.3.2 Pertinent Positive

The pertinent positive formulation differs substantially from the counterfactual one due to the need for a different loss function.

The loss of PP is composed, again, of two parts:

- Negative log-likelihood term: which is no longer negated since the objective here is to have $\mathbf{y} = \hat{\mathbf{y}}$

$$\mathbb{1}_{\mathbf{y} \neq \hat{\mathbf{y}}} \cdot NLL(\mathbf{y}, \hat{\mathbf{y}})$$

- Distance term: reformulated from minimising the amount of changes applied to the original graph, to maximising the amount of edges deleted from it

$$\sum_{i=0}^{N} \sum_{j=0}^{N} |\hat{A}_{ij}|$$

The complete loss is:

$$\mathcal{L}_{PP} = \mathbb{1}_{\mathbf{y} \neq \hat{\mathbf{y}}} \cdot NLL(\mathbf{y}, \hat{\mathbf{y}}) + \beta \cdot \sum_{i=0}^{N} \sum_{j=0}^{N} |\hat{A}_{ij}|$$

Limitations Of PP Generation

Depending on the outcome of the training of the original model f, its *default* prediction can vary. *Default* prediction here refers to the model's output when the given graph has all of its edges removed $A_{ij} = 0$ *for each* $i,j \in \{1...N\}$. If the instance that the user wants to explain has as its label the *default* one, the explanation yielded by the explainer will consist only of an empty graph, since the loss function $\mathcal{L}_{PP}$ aims to maximise the amount of edges removed. This result does not tell the user anything about the model except the *default* class itself, which in general is not very useful.

In order to get a relevant explanation, the adjacency matrices obtained *before* the complete removal of all edges are needed. A simple way to obtain them is to use a method to generate diverse explanations introduced in [18]:

$$\mathcal{L}_{div} = -\gamma \cdot CE(\hat{A}, \hat{A}_{old})$$

where $\hat{A}$ is the explanation currently being generated, $\hat{A}_{old}$ is the one previously generated for the same instance v, γ is an hyper-parameter and CE is the loss introduced in Chapter 2. This new loss term is added to the previously defined $\mathcal{L}_{PP}$ one and it's used to induce the model to produce an explanation that is diverse when compared to the preceding one. This solution also keeps the method general and applicable in the scenario in which the model is treated as a black-box and cannot be re-trained or modified in any way. The complete final loss formulation is:

$$\mathcal{L}_{PP} = \mathbb{1}_{\mathbf{y} \neq \hat{\mathbf{y}}} \cdot NLL(\mathbf{y}, \hat{\mathbf{y}}) + \beta \cdot \sum_{i=0}^{N} \sum_{j=0}^{N} |\hat{A}_{ij}| - \gamma \cdot CE(\hat{A}, \hat{A}_{old})$$

4.4 CF-GNNExplainer++

The CF-GNNExplainer++ algorithm, described in Algorithm 2, closely follows the structure of the original procedure of CF-GNNExplainer introduced in Section 3.2 and the ideas detailed in Section 3.2.1.

The algorithm, starting at line 1, saves a copy of the prediction on the input instance v given by the original model, which will be used later on to build the loss function $\mathcal{L}$. The permutation matrix P is then initialised (line 2) to $\mathbf{0}$, as expected when using the *delta* formulation. Following, the main program loop starts (line 3) and it is repeated k number of times given, where k is one of the input parameters. The loop's body consists of three steps needed to iteratively produce the instance's explanation. First, a new explanation is generated using the *GenExpl()* function (line 4) and the current permutation matrix $\hat{P}$. Then, once that call returns, all the components of the loss are ready and it can be computed (line 5). Finally, the P matrix values are updated using the new loss and the SGD algorithm (line 6).

The *GenExpl* function starts by thresholding the current $\hat{P}$ matrix and initialising $\hat{A}$ (lines 11, 12). Lines 13 through 18 build a new candidate explanation by using the *delta* variant of the perturbation operations, as described in Section 4.2, the result is then saved (line 19) to be later compared with the current best v^*. The conditions that make an explanation *valid*, depending on their type, are then checked (lines 20 - 22). In particular, for CF and PN explanations to be considered *valid* they need to have a different prediction compared to their starting instance v and are expected to have a smaller or equal amount of perturbations compared to the best explanation v^* obtained up that point, otherwise they are discarded. Meanwhile, PP explanations are expected to maintain the same prediction as their starting instance and no further constraint is placed on them. This is because forcing PP explanations to have the same number, or more, of edges removed from the graph as the best explanation v^* would result in a graph without edges, if the instance's prediction is the *default* label of the model. This would also make it impossible to generate diverse explanations from this point onward, since the only acceptable v would then be the same edge-less graph one. The updating of the adjacency matrix $\hat{A}_{old}$, used to keep track of the previous *valid* explanation generated and to compute the diversity term when the hyper-parameter $\gamma > 0$, is then handled (lines 23, 24). Finally, if all the conditions have been satisfied, the candidate explanation $\bar{v}_{cand}$ replaces the current best v^* and it is then returned to the main loop.

Algorithm 2 CF-GNNExplainer: *delta* operations formulation and *Bernoulli* approach to discretization of the P matrix. This version of the algorithm allows for both PP and CF generation.

Input: instance $v = (\mathbf{x}, A)$, a trained model f, an explainer model g, loss function $\mathcal{L}$, learning rate α, distance hyper-parameter β, diversity hyper-parameter γ, number of iterations k, type of explanation E_{type} and the set of allowed operations Ops.

Output: Instance explanation $\bar{v}$

$y \leftarrow f(v)$
$\hat{P}^{(0)} \leftarrow \mathbf{0}$
for all $i \in range(K)$ **do**
    $\bar{v} \leftarrow GenExpl(E_{type}, v, \hat{P}^{(i)})$
    $\mathcal{L} \leftarrow \mathcal{L}(g, v, \bar{v}, \beta, \gamma, \hat{A}_{old})$
    $\hat{P}^{(i+1)} \leftarrow \hat{P}^{(i)} - \alpha \nabla_{\hat{P}} \mathcal{L}$
end for
return $\bar{v}$

function GenExpl($E_{type}, v, \hat{P}$)
    $P \leftarrow thr(\hat{P})$
    $\hat{A} \leftarrow \mathbf{0}$
    if $add \in Ops$ **then**
        $\hat{A} += A - A \cdot P$
    end if
    if $del \in Ops$ **then**
        $\hat{A} += A + (\mathbf{1} - A) \cdot P$
    end if
    $\bar{v}_{cand} \leftarrow (\mathbf{x}, \hat{A})$
    $CondPP \leftarrow E_{type} = PP \text{ and } f(v) = f(\bar{v}_{cand})$
    $CondCF \leftarrow E_{type} \neq PP \text{ and } f(v) \neq f(\bar{v}_{cand})$
    if $CondPP \text{ or } (CondCF \text{ and } \mathcal{L}_{dist}(v, \bar{v}_{cand}) \leq \mathcal{L}_{dist}(v, \bar{v}^*))$ **then**
        if $\gamma > 0 \text{ and } \hat{A}_{old} \neq \hat{A}$ **then**
            $\hat{A}_{old} \leftarrow \hat{A}$
        end if
        $\bar{v}^* \leftarrow \bar{v}_{cand}$
    end if
    return $\bar{v}^*$
end function

Chapter 5

Experiments

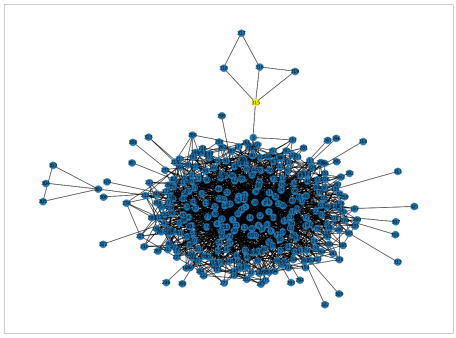
This chapter illustrates the setup and experiments conducted to evaluate the methods proposed in this thesis. The code and related material can be found in the project's GitHub[1]. All the experiments have been conducted on an Intel i7-4720HQ CPU and a Nvidia Geforce 965M GPU. The models and datasets setup have been replicated following [9] and, where needed, [17]. Specifically, the models used are 3-layer versions of GCN [8] for node classification tasks and GAT [16] for graph classification, with hyper-parameters and architecture identical to the previously mentioned works. The underlying models offer an accuracy of at least 84% for each dataset tested.

5.1 Datasets

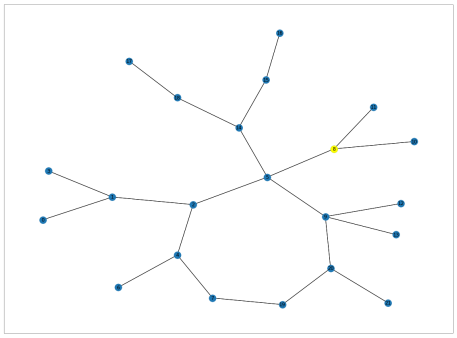
Four datasets, shown in Figure 5.1, have been used to explore the performance of the proposed method, three synthetic ones used to test the explainer on node classification tasks [17], and one real-world graph classification dataset. The synthetic datasets are essential due to the need to have ground-truths to evaluate the *accuracy* of the explainer. The classification task for these datasets consists of, given a node, determine if it belongs to the dataset's specific motif, the ground truth is knowing which nodes are necessary to have a valid motif and which are not.

- BA-shapes (*syn1*): base Barabasi-Albert (BA) graph with added *house-shaped* motifs randomly attached to it. Each node can belong to one of 4 classes: top, middle, bottom or not belonging to the house shape. This dataset is the most dense in terms of edges per node, making the generation of explanations more resource-intensive.
- Tree-Cycles (*syn4*): base 8-level balanced binary tree to which are randomly attached six-node cycle motifs.
- Tree-Grid (*syn5*): same as Tree-Cycles, but the motif is a 3×3 grid.

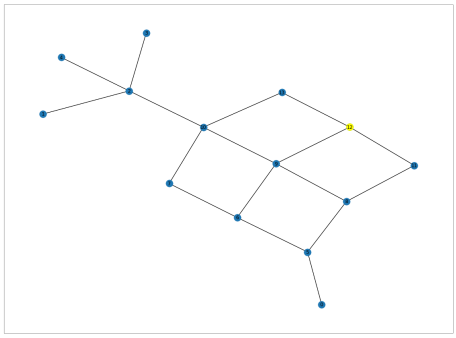
[1] `https://github.com/adettori/cf-gnnexplainer`



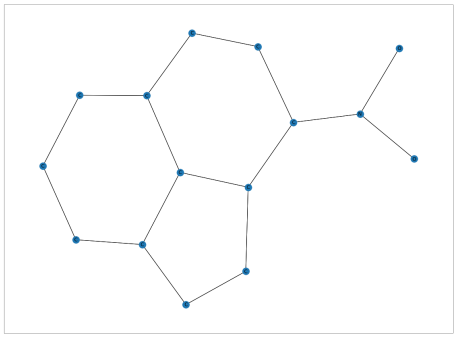
(a) Example of a syn1 *house* (5-node structure at the top)



(b) Example of a syn4 6-nodes cycle



(c) Example of a syn5 3x3 grid



(d) Example of a MUTAG molecule

Figure 5.1: Examples of instances taken from each dataset.

- Mutagenicity (*MUTAG*): set of graphs representing molecules, labeled according to their mutagenic effect [4].

All datasets have been taken unchanged from their respective repositories.

5.2 Metrics

The explanations are evaluated using the following metrics:

- Fidelity (*Fid.*): represents the ratio of generated explanations whose associated predictions y_g are the same as the ones from the original model y_f. When generating a counterfactual lower is better since the focus is in finding a perturbation that changes the result, but when generating PPs the higher the better since the aim is to generate explanations that remove at least one edge while keeping the same prediction.
- Graph distance (*Dist.*): average, taken across all instances, of the number of edges removed to generate the explanation.
- Relative change ratio (*Ch.*): ratio of the number of edges changed, both in

terms of addition and deletion of edges, compared to the original graph $\hat{A}$ in absolute value.

- Accuracy (*Acc.*): measures how *correct* each edge addition or removal is by using the original model's f predictions as ground-truths to verify that the nodes affected by the operation belong to the dataset's motif. Due to the need for these ground-truths to compute the metric, it is not applicable to non-synthetic datasets. It is computed by taking the ratio between the number of unique *edges* that have been affected by the addition/removal operations performed where *at least one* of the nodes that characterise the edge belongs to the datasets' motif and the number of total edges that have been added/removed. The average across all explanation instances is then taken to obtain the final value. The values for the accuracy measure of the addition and deletion operation are kept distinct and are presented as a pair (*Add Acc./Del Acc.*).

- History length (*Hist.*): to allow the user more flexibility in their explainability needs, the explainer stores the history of the explanations obtained for each given instance. This metric is the average of the length of these explanation histories, taken across all the instances explained. It's useful to find which of the explainer's hyper-parameter combinations promote explanation diversity the most. For example, a history length of 5 means that, on average, the explainer generated 5 different explanations for each instance given.

Note that the definition used here for the accuracy metric differs from the original one used in [9], where it is *node-based* instead of *edge-based*. In [9] it is defined as the ratio between the number of unique *nodes*, belonging to the motif and associated to the modified edges, and the total number of impacted unique nodes. Therefore, using this definition, any addition operation involving a node outside the motif would result in an *incorrect* modification. The *edge-based* approach avoids this problem and therefore allows to better represent the work of the explainer when edge addition operations are used.

5.3 Hyper-parameters and Grid-search

Extended searches have been conducted to explore the effect of different hyperparameters when training the explainer with SGD. Specifically, number of iterations $K \in \{300, 500\}$, learning rate $\alpha \in \{0.01, 0.1, 0.5, 1\}$, distance term $\beta \in \{0, 0.1, 0.5\}$, Nesterov's momentum $m \in \{0, 0.5, 0.9\}$, both approaches to the discretization of the perturbation matrix P and both operations formulation have been explored to generate counterfactual/PN and PP explanations. In addition, to overcome the limitations of the formulation, PP generation has been tested with diversity term $\gamma \in \{0.1, 0.5\}$. The complexity is the same as [9]: $O(KN^2)$, where K is the number of iterations and N is the size of the graph explained. The time required to complete the grid-search is ∼240 hours. Among the datasets tested only *syn*1 benefits from

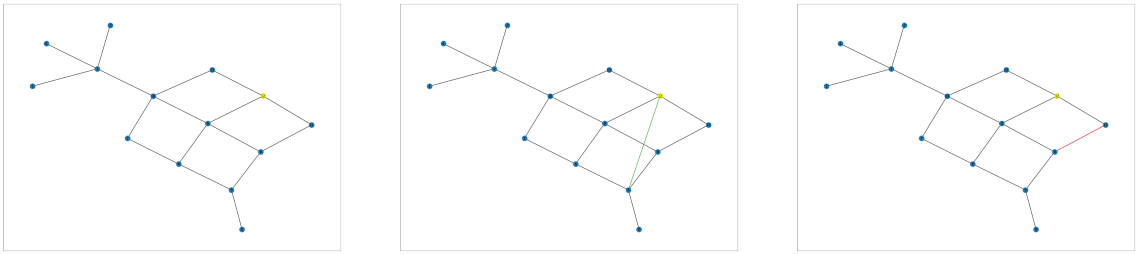
the use of the GPU, the others seem to be unable to, probably due to their smaller size.

5.4 Results

This section contains the best results obtained by performing the grid-search mentioned in the previous section and according to the outlined metrics. Note that, as mentioned in Section 4.3.1, the generation of Pertinent Negative explanations is identical to the generation of counterfactual ones using only the *add* operation in the *delta* approach, therefore the metrics are also the same.

5.4.1 Counterfactual Generation

The best combinations of hyper-parameters have been picked by minimising the fidelity, since that aligns with the objective of a counterfactual explanation, while also keeping the distance between the original instance and the explanation within a range of 1 to 4 operations in order to keep them compact and understandable at a glance.



(a) Starting instance

(b) CF explanation using *add* operations only

(c) CF explanation using *del* operations only

Figure 5.2: Two examples of counterfactual explanations on an instance of the syn5 dataset

Figure 5.2 illustrates an example of the CF explanations generated by the CF-GNNExplainer++ algorithm: figure (a) is the base instance to explain where the target node (yellow) is classified as belonging to the 3×3 grid, figure (b) shows an example of where it is possible to add an edge (green) to the instance's graph in order for the underlying model to be unable to recognize the target node (yellow) as belonging to the 3×3 grid, finally figure (c) highlights which edge it is possible to remove so that again the model cannot classify the target node as belonging to the grid. The first explanation highlights that the model does not consider the target node to belong to the grid anymore once the new edge is added, this could indicate, for example, that the model learnt that only nodes with two edges can be considered to belong to the grid or that it learnt a concept of a 3×3 grid so strict that once two nodes inside the grid are connected the resulting structure is no longer a grid.

The second explanation shows that the model correctly understands that removing an edge from the grid changes the target node classification, since there is no longer a valid 3×3 grid to belong to.

Table 5.1: Comparison of results of the counterfactual explanations. Below each metric, ▼ means that a low value is preferred, while ▲ means that a high one is. The letters in the first column refer to the operations formulation used (O: *original*, D: *Delta*) and the operations allowed (A: add, D: delete).

Metric	BA-SHAPES				TREE-CYCLES				TREE-GRID				MUTAG		
	Fid. ▼	*Dist.* ▼	*Ch.* ▼	*Acc.* ▲	*Fid.* ▼	*Dist.* ▼	*Ch.* ▼	*Acc.* ▲	*Fid.* ▼	*Dist.* ▼	*Ch.* ▼	*Acc.* ▲	*Fid.* ▼	*Dist.* ▼	*Ch.* ▼
O-D	0.39	2.39	0.01	−/0.96	0.21	1.44	0.08	−/0.94	0.07	1.41	0.05	−/0.95	0.16	3.50	0.17
O-A/D	0.22	**2.09**	0.01	0.99/0.98	**0.0**	**1.13**	**0.01**	**0.96/1.0**	0.01	2.16	0.08	1.0/0.92	**0.00**	1.47	**0.02**
D-D	0.44	2.62	0.01	−/0.93	0.21	1.44	0.08	−/0.94	0.08	**1.30**	0.05	−/0.94	0.21	1.67	0.08
D-A	0.31	2.62	0.01	1.0/−	0.12	1.39	0.08	0.93/−	**0.0**	1.32	0.05	0.95/−	0.58	**1.0**	0.07
D-A/D	**0.13**	3.99	0.01	**1.0/1.0**	**0.0**	**1.13**	**0.01**	**0.96/1.0**	0.02	1.38	**0.04**	**1.0/0.95**	**0.0**	1.47	0.03

The results in Table 5.1 shows a benefit by using the combined operations, be it the *delta* or *original* approach to the formulation, providing the user with explanations that have a better fidelity, therefore explaining more of the dataset's instances, and lower distance resulting in smaller explanations. Intuitively these results were expected to hold for all datasets, however this is not true for the Tree-Grid one in which addition only and deletion only changes can offer better results, both in terms of fidelity and distance. The addition only approach seems to also offer some benefits in MUTAG by needing, on average, only 1 edge addition to generate a counterfactual, however it is able to explain less than half of the available instances, making this benefit circumstantial at best. Finally, the accuracy metric is high across the board, indicating that all the approaches tested remove or add edges that are, most of the time, directly connected to the instance's motif.

Table 5.2: Comparison of results of the counterfactual explanations. Below each metric, ▼ means that a low value is preferred, while ▲ means that a high one is. The letters in the first column refer to the P initialisation method used (Std: *standard*, Ber: *Bernoulli*)

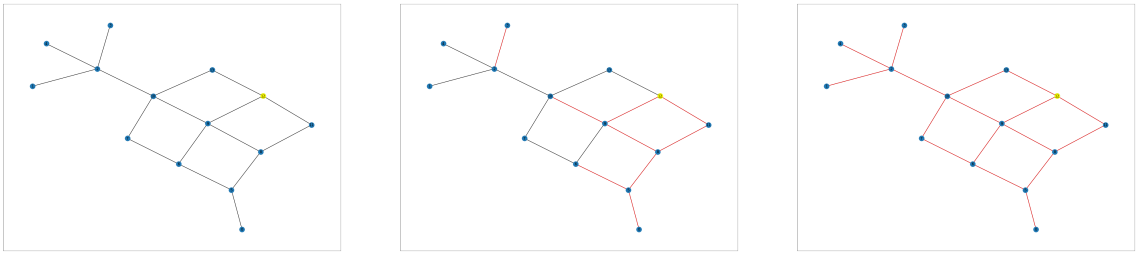
Metric	BA-SHAPES				TREE-CYCLES				TREE-GRID				MUTAG		
	Fid. ▼	*Dist.* ▼	*Ch.* ▼	*Acc.* ▲	*Fid.* ▼	*Dist.* ▼	*Ch.* ▼	*Acc.* ▲	*Fid.* ▼	*Dist.* ▼	*Ch.* ▼	*Acc.* ▲	*Fid.* ▼	*Dist.* ▼	*Ch.* ▼
Std	**0.13**	3.99	0.01	1.0/1.0	0.0	1.32	**0.01**	0.96/−	0.01	2.16	0.08	1.0/0.92	0.16	3.50	0.17
Ber	0.28	**1.45**	0.01	1.0/1.0	0.0	**1.13**	0.02	0.96/1.0	**0.0**	**1.32**	**0.05**	0.95/−	**0.0**	**1.47**	**0.03**

Table 5.2 shows the impact of changing the discretization approach, grouping by which outlines better results for the *Bernoulli* approach in all datasets except BA-Shapes, both in terms of fidelity and distance of the explanations. In BA-Shapes, which is the most dense dataset, while the *Bernoulli* approach generates

on average smaller explanations, it is unable to explain as many instances as the *standard* approach. This outcome is also in line with [14], where the use of the pass-through method for the estimation of the Bernoulli distribution parameters gave better results compared to the use of a sigmoid-based approach.

5.4.2 Pertinent Positive Generation

The results here have been obtained by first maximising the fidelity and then the graph distance, the amount of edges removed. Each combination of hyper-parameters produces multiple explanations, as shown in Figure 5.3, the user can then choose which ones present the most relevant information to understand the model and continue their analysis from there.



(a) Starting instance

(b) PP explanation where some edges are kept

(c) PP explanation where all edges are removed

Figure 5.3: Two examples of PP explanations on an instance of the syn5 dataset

The images in Figure 5.3 illustrate the benefit of introducing the diversity loss term for PP generation: by finding a good combination of hyperparameters, it is possible to gradually remove some of the edges from the instance as in (b), without immediately ending up in the situation shown in figure (c), where the model has as its *default* class the one where the target node belongs to the grid. It is therefore possible to see which edges the model considers necessary to have the target node belong to the grid. Without the use of the diversity term, figure (c) would be the only explanation generated for that instance of the dataset, resulting in no new information given to the user to explain their model's behaviour.

Changing the discretization approach of the P matrix, as recorded in Table 5.3, shows mixed results depending on the dataset used. Fidelity scores clearly benefit from the use of the *Bernoulli* approach, increasing it substantially for BA-Shapes and Tree-Grid. The average length of the history also benefits from using the *Bernoulli* approach, showing an improvement for each dataset except MUTAG. The graph distance and relative change ratio metric improvements are less one-sided, while BA-Shapes and Tree-Grid show better results with *Bernoulli*, the other two datasets invert the trend and perform marginally better with the *standard* approach. It is interesting to note that, in general, this PP formulation is able to reduce the average size of the explanation considerably for each dataset, from a 79% reduction

Table 5.3: Comparison of results of the Pertinent Positive explanations. Below each metric, ▼ means that a low value is preferred, while ▲ means that a high one is. The first column refers to the P discretization method used (Std: *standard*, Ber: *Bernoulli*)

Metric	BA-SHAPES				TREE-CYCLES				TREE-GRID				MUTAG			
	Fid. ▲	*Hist.* ▲	*Dist.* ▲	*Ch.* ▲	*Fid.* ▲	*Hist.* ▲	*Dist.* ▲	*Ch.* ▲	*Fid.* ▲	*Hist.* ▲	*Dist.* ▲	*Ch.* ▲	*Fid.* ▲	*Hist.* ▲	*Dist.* ▲	*Ch.* ▲
Std	0.75	4.8	1004.5	0.75	1.0	5.4	**18.7**	**0.93**	0.76	2.8	26.1	0.75	1.0	**4.4**	**19.3**	**0.98**
Ber	**0.86**	**6.2**	**1064.4**	**0.80**	1.0	**8.3**	18.1	0.90	**0.90**	**6.9**	**27.9**	**0.79**	1.0	3.2	19.2	0.97

for Tree-Grid and up to a 98% one for MUTAG. Depending on the size of the datasets and instances used, this indicates that generating PP explanations can be a feasible approach to getting a different perspective on the model, compared to CF explanations. Unfortunately, for bigger datasets such as BA-Shapes, the explanations generated are still too complex for a human to intuitively understand, but they could still be used as a starting point to conduct more targeted analyses.

Chapter 6

Conclusions

This work has replicated the results from [9], experimented with some aspects of the explanation generation approach and introduced a formulation of Pertinent Positive and Pertinent Negative explanations applied to graphs. Regarding results, this thesis show that, for CF generation, combining both addition and removal can result in better explanations, as in more compact and able to explain a greater number of predictions, and also that using a *Bernoulli*-based approach to the discretization of the perturbation matrix can substantially improve the explanation's quality. The PP formulation is also promising in terms of quality of the generated explanations and it can potentially provide the user with different insights into the model's behaviour compared to CF at the same computational cost.

The limitations outlined in [9], however, still remain. CF-GNNExplainer++ takes into account only the edges of the graph to generate an explanation, while ignoring the features of the nodes and edges. It also does not support a way to generate explanations according to a given set of rules of what is and is not acceptable, for example the explanations generated for MUTAG might be invalid from a chemistry point of view and the user might find more useful directing their computational efforts towards realistic ones. Finally, as is the case for outcome-based XAI [7] methods, explaining the predictions on an entire dataset is computationally expensive and should be therefore limited to targeted instances of interest.

Future works could focus on solving these issues, in particular taking into account the features of edges and nodes should be relatively straightforward by extending the perturbation matrix approach already in use. An additional interesting line of research on the generation of explanations is also represented by the work done in MEG [10], which could be expanded upon either by incorporating reinforcement learning techniques into the CF-GNNExplainer framework itself or by adding support for more explanation types, this in order to tackle the problem of generating *valid* explanations, following a specific set of rules, for the user.

Acknowledgements

I'd like to thank my supervisors, without whose extensive reservoirs of patience this work would have never seen the light of day.
A heartfelt thank you to the friends and flatmates that I've had through the years, for being there and for keeping me company in the innumerable occasions when I just couldn't focus on my studies.
And finally, a big thank you to my family, who made this all possible and whose support has been unwavering since the beginning of this journey.
Thank you.

Bibliography

[1] André Artelt and Barbara Hammer. *On the computation of counterfactual explanations – A survey.* 2019. DOI: 10.48550/ARXIV.1911.07749. URL: https://arxiv.org/abs/1911.07749.

[2] Davide Bacciu et al. "A gentle introduction to deep learning for graphs". In: *Neural Networks* 129 (2020), pp. 203–221. DOI: 10.1016/j.neunet.2020.06.006. URL: https://doi.org/10.1016%5C%2Fj.neunet.2020.06.006.

[3] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. *Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation.* 2013. DOI: 10.48550/ARXIV.1308.3432. URL: https://arxiv.org/abs/1308.3432.

[4] Asim Kumar Debnath et al. "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity". In: *Journal of Medicinal Chemistry* 34.2 (1991), pp. 786–797. DOI: 10.1021/jm00106a046. eprint: https://doi.org/10.1021/jm00106a046. URL: https://doi.org/10.1021/jm00106a046.

[5] Amit Dhurandhar et al. *Explanations based on the Missing: Towards Contrastive Explanations with Pertinent Negatives.* 2018. DOI: 10.48550/ARXIV.1802.07623. URL: https://arxiv.org/abs/1802.07623.

[6] European Commission. *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance).* 2016. URL: https://eur-lex.europa.eu/eli/reg/2016/679/oj.

[7] Riccardo Guidotti et al. "A Survey of Methods for Explaining Black Box Models". In: *ACM Comput. Surv.* 51.5 (Aug. 2018). ISSN: 0360-0300. DOI: 10.1145/3236009. URL: https://doi.org/10.1145/3236009.

[8] Thomas N. Kipf and Max Welling. *Semi-Supervised Classification with Graph Convolutional Networks.* 2016. DOI: 10.48550/ARXIV.1609.02907. URL: https://arxiv.org/abs/1609.02907.

[9] Ana Lucic et al. *CF-GNNExplainer: Counterfactual Explanations for Graph Neural Networks*. 2021. DOI: 10.48550/ARXIV.2102.03322. URL: https://arxiv.org/abs/2102.03322.

[10] Danilo Numeroso and Davide Bacciu. *MEG: Generating Molecular Counterfactual Explanations for Deep Graph Networks*. 2021. DOI: 10.48550/ARXIV.2104.08060. URL: https://arxiv.org/abs/2104.08060.

[11] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323 (1986), pp. 533–536.

[12] Roy Schwartz et al. *Green AI*. 2019. DOI: 10.48550/ARXIV.1907.10597. URL: https://arxiv.org/abs/1907.10597.

[13] A. Sperduti and A. Starita. "Supervised neural networks for the classification of structures". In: *IEEE Transactions on Neural Networks* 8.3 (1997), pp. 714–735. DOI: 10.1109/72.572108.

[14] Suraj Srinivas, Akshayvarun Subramanya, and R. Venkatesh Babu. *Training Sparse Neural Networks*. 2016. DOI: 10.48550/ARXIV.1611.06694. URL: https://arxiv.org/abs/1611.06694.

[15] Rocio Vargas, Amir Mosavi, and Ramon Ruiz. *Deep learning: A review*. Working Paper. Jan. 2017. URL: https://eprints.qut.edu.au/127354/.

[16] Petar Veličković et al. *Graph Attention Networks*. 2017. DOI: 10.48550/ARXIV.1710.10903. URL: https://arxiv.org/abs/1710.10903.

[17] Rex Ying et al. *GNNExplainer: Generating Explanations for Graph Neural Networks*. 2019. DOI: 10.48550/ARXIV.1903.03894. URL: https://arxiv.org/abs/1903.03894.

[18] Yue Zhang, David Defazio, and Arti Ramesh. *RelEx: A Model-Agnostic Relational Model Explainer*. 2020. DOI: 10.48550/ARXIV.2006.00305. URL: https://arxiv.org/abs/2006.00305.