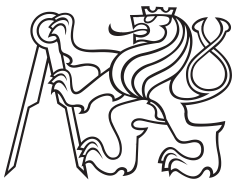
Master's Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Task-Aware Relation Type Selection for Machine Learning Algorithms on Graphs

Michal Mareš

Supervisor: Ing. Pavel Procházka, Ph.D.
Field of study: Open Informatics
May 2023

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Mareš Michal** Personal ID number: **474428**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence**

II. Master’s thesis details

Master’s thesis title in English:

Task aware relation-types selection for machine learning algorithms on graph

Master’s thesis title in Czech:

Výběr vhodných typů relací pro algoritmy strojového učení na grafech

Guidelines:

Machine learning algorithms on heterogenous graphs present a powerful framework for AI problems exhibiting both entity-specific and relational information. Whereas the primary focus of the research community is related to design of a particular method (typically GNN) best performing on a given graph dataset, industrial applications typically need to establish the dataset from the available data prior application of these powerful methods. Since the source data is usually very specific for a given problem and often also with confidentiality restrictions, the dataset creation is typically outside of the main research stream. In traditional machine learning, this task is known as feature engineering, where the goal is to represent each example by features as relevant to the downstream task as possible.
This thesis aims to relevant relation-types selection or their creation for a given task from input data (graph-variant feature engineering). The particular goals of the thesis can be summarised as follows:
* Provide a survey of related literature.
* Describe the problem formally.
* Identify and describe an algorithm/method of suitable relation types selection or creation:
* For a given graph algorithm, preferably a custom graph algorithm that is used in Cisco for malicious content retrieval.
* Discuss/provide generalization of this method to an arbitrary graph algorithm.
* Implement and verify the suggested algorithm on:
* Malicious content retrival taks with priprietary Cisco data,
* A suitable public dataset for demonstration of the proposed algorith.

Bibliography / sources:

Dvorak, Stepan, Pavel Prochazka, and Lukas Bajer. "GNN-Based Malicious Network Entities Identification In Large-Scale Network Data." In NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium, pp. 1-4. IEEE, 2022.
P. Procházka, M. Mareš, M. Dědič, Downstream Task Aware Scalable Graph Size Reduction
for Efficient GNN Application on Big Data, in: Information Technologies - Applications
and Theory (ITAT 2022), Zuberec, Slovakia, 2022
Khalil, Issa, Bei Guan, Mohamed Nabeel, and Ting Yu. "Killing two birds with one stone: Malicious domain detection with high accuracy and coverage." arXiv preprint arXiv:1711.00300 (2017).
Wu, Zonghan, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S. Yu Philip. "A comprehensive survey on graph neural networks." IEEE transactions on neural networks and learning systems 32, no. 1 (2020): 4-24
Schlichtkrull, Michael, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. "Modeling relational data with graph convolutional networks." In European semantic web conference, pp. 593-607. Springer, Cham, 2018.

Name and workplace of master's thesis supervisor:

Ing. Pavel Procházka, Ph.D. Cisco

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **01.02.2023** Deadline for master's thesis submission: **26.05.2023**

Assignment valid until: **22.09.2024**

Ing. Pavel Procházka, Ph.D. Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgements

Throughout my studies, I have been fortunate to have the unwavering support of numerous individuals, without whom this thesis would have remained a distant dream. I would like to express my heartfelt gratitude to all those who have supported me on my path and offered their wisdom, love, and encouragement.

To my esteemed supervisor, who has been a source of inspiration, guidance, and knowledge. I am truly grateful for your belief in my abilities and your tireless dedication to my progress which have been instrumental for this work, especially when I doubted myself. Your keen insights and patience have not only made me a better researcher but also a more inquisitive and thoughtful individual.

To my loving family, who have always been the bedrock of my life, thank you for your unconditional love, support, and understanding. Your sacrifices, faith in my dreams, and countless words of encouragement have been the pillars upon which I have built my academic pursuits. I dedicate this achievement to you, for you have instilled in me the values and determination.

To my dearest girlfriend, whose love and support have been my refuge in times of doubt and my celebration in times of triumph. Your presence has been a constant reminder of the beauty and joy that life has to offer, and your belief in me has been a source of immeasurable strength. I am honored to share this milestone with you.

To my friends, who have been an incredible source of encouragement, I extend my deepest appreciation. Our shared laughter, shared struggle with some, and mutual support have enriched my academic journey and created memories that I will cherish forever.

To my colleagues, who have become my friends over such short period of time and had kind and supportive words ready any day of the week. Our engaging discussions and exchange of ideas have not only enhanced my research but also fostered a sense of community and shared purpose.

Thank you everyone, I could not have done it without you.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

In Prague on May 26, 2023

Abstract

This thesis explores the connection between graph structure and the performance of graph algorithms in retrieval task. The main goal is to identify structural features that contribute to better performance and can be used to select relevant relation types. The research is conducted on private datset of network flows provided by Cisco and public graph datasets.

A novel approach is introduced for evaluating the importance of structural features on graphs. This method uses a regression meta-model on datapoints consisting of graph features and task performance obtained using the studied graph machine learning algorithm. The thesis studies multiple graph features, and proposes a novel definition of an edge-based confusion matrix for retrieval tasks on a graph.

Experimental results indicate that important graph features vary across datasets, algorithms and performance metrics. An additional application of the proposed approach is shown in accelerating hyperparameter search, which shows promising results but requires further research to provide theoretical boundaries.

Despite its limitations, this work contributes to the field of graph machine learning by providing a more comprehensive and explainable approach for evaluating the importance of graph features. Future work should focus on normalizing graph features, refining feature importance analysis, and conducting a thorough investigation of the hyperparameter search application.

Keywords: graph theory, graph algorithms, graph neural networks, node classification, message passing, indicators of compromise, cybersecurity, information retrieval, malware detection, network traffic, machine learning

Supervisor: Ing. Pavel Procházka, Ph.D.

Abstrakt

V této závěrečné práci zkoumám souvislost mezi strukturou grafu a výkonem grafových algorithmů pro úlohu vyhledávání uzlů se specifickou vlastností. Hlavním cílem je identifikovat strukturální vlastnosti, které přispívají k lepším výsledkům a mohou být použity pro výběr vhodných typů relací při stavbě grafu. Výzkum probíhá na soukromém datasetu síťové komunikace poskytnutém společností Cisco a na volně dostupných grafových datasetech.

Představuji nový přístup k vyhodnocování důležitých strukturálních vlastnostní na grafech. Metoda využívá "meta-model" pro regresi využívající strukturální vlastnosti grafu jako příznaky a výsledek výkonnostní metriky algorithmu jako predikovanou hodnotu. Strukturální vlastnosti jsou navrženy podle odborné literatury a zároveň zavádím novou definici matice záměn orienovanou na hrany při úkolu vyhledávání uzlů cílové třídy.

Experimentální vyhodnocení naznačuje, že důležité grafové vlastnosti se liší napříč datsety, algoritmy a metrikami výkonu. Zajímavým uplatněním navrhovaného přístupu je vyhledávání hyperparametrů, kde dosahujeme povzbudivých výsledků které by mohly mít za následek snížení náročnosti optimalizace hyperparametrů pro algoritmy strojového učení na grafech. Tato aplikace ale vyžaduje další výzkum, aby mohly být stanoveny hranice aplikace a její teoretický základ.

Navzdory svým limitacím tato práce přispívá oblasti strojového učení na grafech zavedením nové metody vysvětlování a určování důležitosti strukturálních grafových vlastností. V budoucnu bych se rád zaměřil na normalizaci grafových vlastností, zlepšení analýzy výsledků důležitosti vlastnostní a důkladnější prozkoumání aplikace této metody pro hledání hyperparametrů.

Klíčová slova: teorie grafů, grafové algoritmy, grafové neuronové sítě, klasifikace uzlů grafu, kyberbezpečnost, získávání informací, detekce malwaru, síťová komunikace, strojové učení

Překlad názvu: Výběr vhodných typů relací pro algoritmy strojového učení na grafech

Contents

Chapter 1

Introduction

Graphs are mathematical structures used to represent objects called vertices or nodes and their relations using links also called edges. In computer science, they are especially useful for capturing network topology, network flows or other non-euclidean data.

As a result, developing graph algorithms capable of solving different tasks with graph data on the input has been of major interest to the scientific community. Graph neural networks (GNNs) have been especially popular in the last few years and have shown excellent performance in a number of tasks, such as node and graph classification, link prediction or community detection. Successful applications range from link prediction in social networks [1] and drug discovery [2] to cybersecurity [3], [4], [5].

However, simpler methods are still actively researched with one of the reasons for their advantage being explainability, which is usually lacking in the GNN architectures. This property can be especially valuable in critical applications such as cybersecurity or medicine.

Very little research has been done on the topic of graph building strategies. The data variability across different applications and sometimes even confidentiality restrictions limit the mainstream research in this area. Oftentimes, edges between nodes can be defined in multiple ways based on the input data and its interpretation with small nuances possibly influencing the algorithm performance substantially.

What complicates things even more is the fact, that the optimal graph may vary across the machine learning graph algorithms and tasks. Works such as [6] exist which present the idea of graph rewiring in order to improve the performance of message passing algorithms. Nevertheless, in some applications, this is simply not possible as this introduces artificial relations between the data which do not correspond with reality and therefore have a detrimental impact on the ability to explain the results.

The goal of this work is to study graph structures in connection to the downstream task and find the structural features contributing to better performance. Based on these findings, relevant relation types will be selected

to improve the graph algorithm performance.

1.1 Motivation

This work has been done in collaboration with Cisco TD&R Data Science team. Their main goal is to provide threat detections based on customer network telemetry in their product Global Threat Alerts [7]. Machine learning (ML) plays a great part in delivering these detections swiftly as an enormous number of threat researchers would be needed to keep up with the network flows without the help of ML algorithms.

A specific retrieval algorithm used in the production pipeline is responsible for retrieving indicators of compromise (IoC) on top of domain names. Part of our effort in this work is to improve its precision and coverage by using additional relations. In order to do that, we have been given access to a subset of Cisco's network telemetry for the purpose of this research.

As those data are confidential and cannot be published, we are also going to attempt to reproduce the results by applying more general ML graph algorithms on publicly available datasets. Nonetheless, that might not be possible in full due to the differences and limitations of the data and algorithms.

The main goal of this work is to improve our understanding of building a graph representation of the data in order to maximize the performance of the task. Such graph building is not thoroughly explored in the scientific literature despite being extremely important in our opinion. One of the main reasons might be the frequent usage of proprietary data and the restrictions it poses. Moreover, any conclusions drawn from specific data might not be applicable across datasets.

1.2 Contribution

This thesis introduces a novel approach for evaluating the importance of structural features on graphs. The proposed method, described in more detail in Section 1.3, uses a regression meta-model on datapoints consisting of graph features and task performance using the studied graph ML agorithm.

In Chapter 2, we provide a comprehensive review of related literature. The main topic of graph structure and feature engineering is supplemented by the topic of state-of-the-art ML algorithms on graphs and their applications in cybersecurity.

Compared to the existing research aiming to improve the algorithm performance by graph rewiring ([8], [6]), this work studies multiple graph features instead of just one. We do so by considering explainability instead of arbitrarily changing the edges as in the mentioned works.

A large number of structural and structure-agnostic graph features are proposed in Section 5.4. Many of them are modified to reflect different class distribution in the same graph structure. In addition to already established graph features, a novel definition of an edge-based confusion matrix for retrieval task on a graph is proposed. In one of the proposed confusion matrix features (Definition 5.32) we show that precision corresponds to attribute homophily from [9], which further validates our new concept.

The experimental section explores feature importance for both the private network telemetry in combination with the custom IoC retrieval algorithm and a popular GNN architecture on public datasets. Graph features that are important in the same manner across datasets were not be found. On public dataset we explore an additional performance metric and discover that despite two performance metrics might have some intersecting important features, they influence the performance in the exact opposite ways.

As an indirect result of the proposed approach, an attempt to use the graph features to accelerate the hyperparameter search was made. Initial trials indicate promising results as we are able to approximate the best model with little deviation from optimum by evaluating as little as 5% of the hyperparameter configurations.

1.3 Method

Because there were no directly applicable solutions for the problem at hand in the related literature, we first attempted to find a proxy metric in the form of a graph property that would be highly correlated with task performance. Had such graph property existed, its optimization would directly lead to building an optimal graph for the task. Unfortunately, no such graph property was found, so we modified the approach to cover the graph structure with multiple features.

Hypothesis 1.1 (Foundational hypothesis). Consider a regression model trained on datapoints consisting of graph feature vectors as the input variables and a performance metric as the ground truth. The goal of the regression model is performance prediction. Assuming the model is able to generalize on new datapoints and does not overfit on the training examples, its feature importance is going to indicate which graph features benefit performance the most. The illustration of this approach is captured in Figure 1.1.

Knowledge of important graph features would enable building the graph in a way to optimize them and therefore maximize the task performance, which is the ultimate goal of this thesis. Graph property optimization could be done by selecting which relations to include in the graph and which not to include, thus satisfying the explainability requirement.

By applying the custom Cisco retrieval algorithm to private network telemetry and GraphSAGE to public datasets modified to retrieval tasks, true

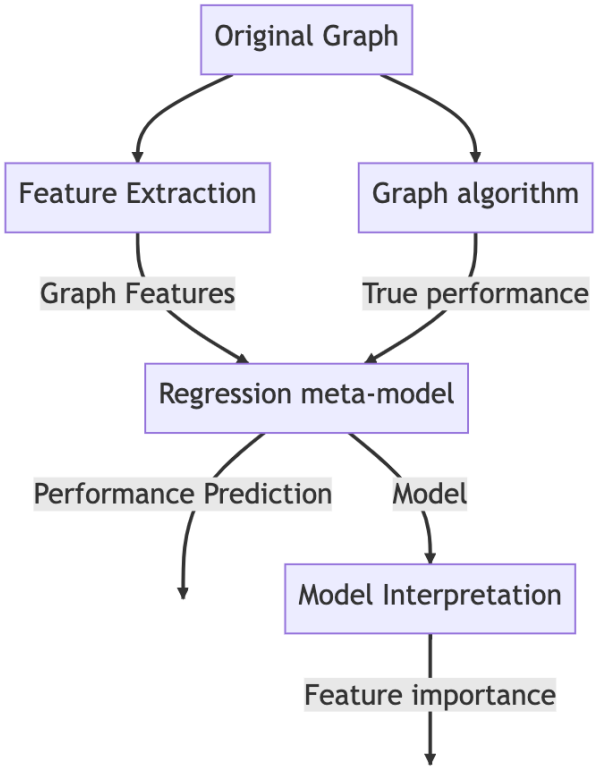


Figure 1.1: Thesis approach illustration. Graph features of the original graph and task performance dependent on the algorithm are used to train the regression meta-model. Meta-model provides the feature importance which can be used to optimize the original graph to maximize task performance. Meta-model's performance prediction is not the main goal and serves only as a validation to verify the model reflects graph structure and task and doesn't just over-fit on the training data.

performance to use the ground truth was obtained. Graph features were also computed for each of the datapoints.

We attempted to utilize the negative curvature from [6] as a graph feature for this work. However, as we are only able to use aggregations and not information for each node, we could not find negative curvature aggregation that would be evaluated as an important feature by the explanation algorithm, therefore it was omitted from further experiments because of its high computational complexity.

In the end, we are going to provide multiple experimental setups to evaluate different aspects of practical applications, such as generalization across datasets or generalization through time. For each setup, a feature explanation algorithm will be used to find the most important graph features.

1.4 Outline

The thesis is organized as follows. Chapter 2 explores current related work and state-of-the-art approaches to outline the possible approaches. Chapter 3 provides the theoretical background on graphs, establishes notation used throughout the thesis and introduces the targeted tasks on graphs. In Chapter 4, theoretical framework for algorithms used to solve targeted tasks is described along with the regression algorithm used for the meta-model and method for interpreting the feature importance. Both public and custom Cisco datasets and specific algorithms used are presented in Chapter 5, along with definitions of utilized graph features. Chapter 6 follows with the experimental evaluation which explores different scenarios for the regression meta-model training and attempts to answer which graph features influence task performance the most. The thesis is concluded in Chapter 7, where Section 7.3 contains summary of accomplishments with respect to the thesis assingment.

Chapter 2

Related work

As a part of this thesis, literature related to this work was studied, such as malicious domain retrieval in cybersecurity, graph structure, feature engineering, feature importance and ML graph algorithms and their explainability in general. To the best of my knowledge, none of them take a similar approach to the problem concerned in this work. In this chapter, a summary of literature that is related and in some cases influenced the direction of our work is going to be provided.

2.1 Graph structure and feature engineering

Graph structure and its impact on graph algorithm performance is gaining more traction in the research community in the last few years. Despite that, graph building is not often addressed, which is understandable as in order for the edges to be interpretable, building strategy is usually given by the nature of the data. Graph rewiring methods are studied instead, which modify the edges in order to increase performance. Grap rewiring is similar to our approach but fails when explainability is an important factor.

There had been research exploring feature selection, such as [10], where authors proposed a feature ranking algorithm. They then use it to select a subset of features from widely adopted datasets while maintaining high accuracy. For example, the Cora dataset with 1433 features was reduced to 225, and the accuracy dropped from 83.5% to 68.2%. They also use the algorithm to extract new features as linear combinations of the original ones. This way, they are able to increase the performance to 73.8% for the Cora dataset by extracting top 225 features.

Authors of [11], of which I am a co-author, approach the graph structure differently: they contract the edges and join nodes based on a node similarity measure computed from a simple base model. In some cases, this enables them to use a distilled graph almost 50% smaller than the original while maintaining approximately the same accuracy. Their goal is to lower the computational resources needed while keeping the same accuracy or given an

accuracy use the least amount of computational resources possible.

In [6], authors explore graph structure more generally with respect to the usual message-passing paradigm, specifically a concept called oversquashing. That occurs when the performance of the task relies on long-distance interactions and the number of neighbors grows rapidly with increasing distance to the origin. They present a potential solution in the form of a graph rewiring algorithm which adds supporting edges for the edge with highest negative curvature and optionally removes the edge with most positive curvature while keeping the degree distribution. Long range dependecies are especially important in heterophilic graphs which will be considered in this work. Earlier work, [8], also introduces a graph rewiring algorithm DIGL, which removes all edges in the beginning and then creates new ones. However, neither of these works are applicable when interpretation is important, as mentioned at the beginning of this sections, as arbitrarily added edges do not have any evidence in the real world data.

2.2 Machine Learning on Graphs

In the past, traditional ML methods were applied to graphs usually using structure-agnostic approach utilizing only the node features. For probability networks, for example Bayesian networks or Markov random fields, bayesian message-passing framework could be used to utilize the graph structure, usually without the features. One such algorithm is Belief Propagation [12]. Random walk based methods, such as node2vec [13], also generate a node embedding based on the graph topology.

With the advent of deep learning, various architectures within the Graph Neural Networks (GNNs) family have been introduced in recent years. Their main advantage is the combination of both the local (node features) and structural information (edges between the nodes), which results in a performance boost in many applications.

A survey of GNN architectures until 2019 is well summarized in [14]. GNN architectures are ordered into new taxonomy consisting of 4 types: Recurrent GNNs [15], Convolutional GNNs [16], Graph Autoencoders and Spatial-temporal GNNs. In addition, different applications are discussed and authors suggest possible future research directions.

One of the suggested research directions is the exploration of new methods for heterogeneous graphs as most of the architectures assume homophilic graphs. In those, edges usually connect similar nodes rather than different ones. This might have been caused by initially using graph networks in the social network scenario. Still, there are many examples of heterophilic graphs, for example, computers (personal devices connect to routers) or (heterosexual) dating networks, as mentioned in Zheng [17]. The homophilic assumption is mitigated by redefining and extending the neighborhood, addressing the

aggregation, and updating functions to boost the expressivity or both.

In contrast to others, in [18], instead of proposing a new GNN architecture, the authors explore different design options such as number of layers or different activation functions. A similarity metric is also proposed and authors use it to transfer the best models across different tasks. This has been a direct inspiration for this work and will be described in more detail in Section 5.3.1.

Authors of [15] are solving the task of link prediction or "recovery of missing facts" and entity classification in knowledge graph. Knowledge graph consists of nodes, which they also call entities, and labeled edges, which means there are multiple edge types to consider. This is similar to the case of network telemetry we are dealing with in this work, as domains are interconnected with for example both server IPs and network devices. In the paper, a new RGCN architecture is built to handle the target tasks. Authors further elaborate, that an important aspect of the performance is the presence of an autoencoder in the network whose addition also noticebaly improves architectures they are comparing themselves with.

2.3 Cybersecurity

Similar to this work, in [19], an automated way of processing network telemetry is proposed with the goal being to identify indicators of compromise (IoCs). The input is a multipartite graph with multiple node types: domains, IP addresses and users. There's an edge between user and domain when user has visited the domain and an edge between IP address and a domain when the domain has been resolved to that IP address. Multiple GNN architectures and simpler algorithms such as Random Forest or SVM are compared at the classification task, all intialized just with an initial set of known malicious domains. In the end, there was little difference in the performance, contrary to the authors' expectations that GNNs should perform better.

In [5], Kahlil et al. propose a DNS-specific algorithm for malicious domain detection. They use definitions established in their earlier work [20] that establish an explicit association between two domains if they share common IPs that do not all come from the same Autonomous Systems (AS). For the classification itself a semi-supervised random forest is used with impressive results (classification accuracy of 98%). They also explore the importance of attributes.

In [4], authors present EvilSeed algorithm for detecting malicious websites. They improve the concept of using web crawlers to suggest new malicious websites based on a set of known website threats called seeds, which are then filtered and finally evaluated using a detection system. The final detection system is sometimes refered to as oracle in the literature. Oracle provides reliable results but it is costly to use it, so the budget of calls is usually

limited. Because of this, the candidates sent to the oracle must be carefully selected to contain as many positive pages as possible. EvilSeed replaces the crawler with a guided search by automatically generating search queries based on the seeds and leveraging search engines to retrieve URLs sharing the desired properties. Queries are generated based on different aspects of the seeding pages such as their word corpus or DNS data. With 604 seeds and 226 140 analyzed URLs (oracle calls), EvilSeed can obtain 3 036 malicious domains while crawler approach (seeded with terms that were trending) was able to find 604 malicious pages out of 437 251 oracle calls.

Similar to the works above, part of this work is a custom algorithm used to retrieve IoCs in the domain name space using a connection map created from network telemetry. In addition, mentioned algorithm also uses an oracle for the non-trivial domain evaluation with limited budget. For more details, see Section 5.3.2.

2.4 Interpretability

For simpler methods such as linear model or decision trees, the intrepretations are easy to understand. With the rise of deep learning and ever increasing number of their parameters, interpretability and explainability have yet to catch up, although being a hot topic for about the past 10 years. In addition to improved debugging of model mistakes, being able to explain the results to customers or making sure the model does not discriminate is slowly becoming a must. Regarding the goals of this work, explainability and feature importance is a crucial part of learning more about what makes algorithms perform good on graphs.

Popular paper regarding explainability of GNNs is [21]. In there, GNNExplainer model-agnostic framework is proposed with the goal of providing interpretable explanations of predictions on any ML task on a graph. The problem of explainability is formulated as "optimization task that maximizes the mutual information between GNN's prediction and distribution of possible subgraph structures" [21, p. 1]. The result is an subgraph and a subset of node features important for the prediction. Both quantitative and qualitative analysis are provided. Ying's GNNExplainer outperforms both of the compared attention and gradient explanation methods on all considered synthetic datasets. From the qualitative side, the explainer is able to identify the mutagenic groups in the MUTAG dataset consisting of molecule graphs with the goal of predicting the mutagenicity on a specific type of bacteria to give an example.

Unrelated to graphs, SHAP is a commonly used framework introduced in [22]. It originates in collaborative game theory from a concept of Shapley value, which solves the issue of gain distribution among cooperating members based on their contribution. Applying this in the ML field, authors present an approach for interpreting results across different models. To deal with

the complexity of exact computation, paper further introduces custom approximation methods with optional assumptions of feature independence and model linearity.

Chapter 3

Theoretical Background

In this chapter, theoretical background and notation are going to be provided for graph theory as well as graph tasks.

3.1 Data Representation Using Graphs

Graphs are a popular method of data representation. Besides being easy to interpret for humans, they can also provide good reasoning based on the neighboring nodes in applications where explainability is important. They often appear in the world around us, spanning from the "Seven Bridges of Königsberg", through the railroad network, electric grid, chemical molecules, and most recently the internet. Many popular graph algorithms were invented to solve a particular issue, such as Borůvka's algorithm for finding the minimum spanning tree, which was used to construct the electricity network for Moravia. In the last half-century, the internet emphasized the importance of network theory, a specialized graph theory field.

Even though graph construction from the data is straightforward for some applications, very often it is not. For data such as network logs, we can build many different graphs based on the definition of what should be considered a node and what should be considered an edge between the nodes. It is non-trivial to construct the graph to optimize the task performance for a specific graph task.

3.2 Graph Theory

As mentioned earlier, graphs are mathematical structures used to represent objects called vertices or nodes and their relations using edges or links. There are multiple types of graphs. The two main types are probably considered to be directed and undirected, whose difference lies in the edges. In the most

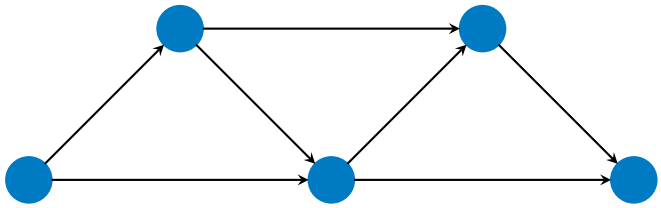


Figure 3.1: Example of a directed graph on five vertices. Undirected graphs are usually drawn with simple lines instead of arrows.

basic definition, a graph G is defined as a tuple

$$G = (V, E) \tag{3.1}$$

where V is a set of nodes and E is the set of edges. In an undirected graph, edges do not have a direction and therefore are defined as an unordered pair of nodes

$$e_i = \{v_j, v_k\} = \{v_k, v_j\}. \tag{3.2}$$

Directed graphs contain directed edges, and thus elements in E are ordered pairs of nodes

$$e_i = (v_j, v_k). \tag{3.3}$$

In the case of a directed graph, we can also write:

$$E \subseteq V \times V \tag{3.4}$$

Both have relevant usage; for example, we might use a directed graph to capture a citation network, where the first (source) node is a paper A and the second (target) node is a paper B being cited by paper A. An undirected graph might be better suited for a friendship network on social media, as the relationship between the users is mutual.

There are several graph modifications: graphs with weighted edges, parallel edges, self-loops, or other specifics but those are irrelevant to this work.

3.2.1 Multipartite Graphs

A special type of graph relevant to this work is a multipartite graph. In a k-partite graph, nodes can be divided into sets $P_i \subseteq V$ where $i \in \{1, ..., k\}$ such that there is no edge between the nodes in the same set:

$$P_i \times P_i \cap E = \emptyset \tag{3.5}$$

For $k = 1$ the graph is called unipartite and in that case $P_1 = V$. For $k = 2$ the graph is called bipartite and it has the following properties:

$$G = (V, E) \tag{3.6}$$
$$V = P_1 \cup P_2 \tag{3.7}$$
$$P_1 \cap P_2 = \emptyset \tag{3.8}$$
$$E = \{\{u, v\} \mid u \in P_1, v \in P_2\} \tag{3.9}$$

Example of a bipartite graph can be graph where P_1 are nodes representing user devices and P_2 are nodes representing websites (domain names). There is an edge between a user device and a website when the website was visited on the device according to a internet service provider's network log. There are no connections between websites or between user devices.

Sometimes it is useful to convert a bipartite graph to unipartite. In that case, only nodes from one of the sets are kept in the graph (let's assume P_1 is kept) and an edge exists between the nodes when they have at least one common neighbor in the second node set $P2$ in the original graph G. Let's first define node neighborhood $\mathcal{N}(u)$ which is a set of nodes connected to node u and degree of a node $d(u)$:

$$\mathcal{N}(u) = \{v \mid \{u, v\} \in E\} \tag{3.10}$$
$$d(u) = |\mathcal{N}(u)| \tag{3.11}$$

Then the constructed unipartite graph G' is

$$G' = (V', E') \tag{3.12}$$
$$V' = P_1 \tag{3.13}$$
$$E' = \{\{u, v\} \mid \mathcal{N}_G(u) \cap \mathcal{N}_G(v) \neq \emptyset\} \tag{3.14}$$

where $\mathcal{N}_G(u)$ is the set of neighboring nodes in the original bipartite graph G.

3.2.2 Relation Types

Relation type, sometimes also reffered to as edge type, is an edge establishing a specific relationship between the two nodes. It encodes a specific shared property into the graph structure. Notation $\mathcal{R}$ will be used for the set of relation types.

In our network telemetry datasets, described in Section 5.2.3, we refer to these relations as modalities and encode it using a bipartite graph where instead of a relation edge (for example device has connected to a specific domain) we encode it by a ordinary edge and a set of property (modality) nodes P_2. Regarding to the example given, there is an edge between P_1 (user) and P_2 (domain) when user has visited that domain.

3.3 Tasks on Graph

In addition to the graph itself, ML algorithms on graphs usually take advantage of node features[1]. For this reason, let's consider a graph

$$G = (V, E, \mathbf{X}, \mathcal{Y}) \tag{3.15}$$

where V, E have been defined earlier, $\mathbf{X} \in \mathbb{R}^{n \times m}$ is a feature matrix with $n = |V|$ and m being the number of features. Note, that rows of the matrix X are feature vectors $\mathbf{x}_v$ of specific nodes.

$\mathcal{Y}$ is a function assigning a true class label to each node. C is a set of class labels.

$$\mathcal{Y} : V \to C, u \mapsto y_u \tag{3.16}$$

For node u this functions assigns the true label $y_u \in C$. True labels for each node together form a vector $\mathbf{y} \in \mathbb{R}^n$ containing a true label for all of the nodes in V.

3.3.1 Node Classification

A node classification algorithm $\mathcal{A}$ aims to predict node labels based on the input graph G. Label predictions vector will be denoted $\hat{\mathbf{y}} \in \mathbb{R}^n$. Predicted labels $\hat{\mathbf{y}}$ and true labels $\mathbf{y}$ are then compared using a performance metric $\mathcal{P}$, such as accuracy. The result is a real number $p \in \mathbb{R}$.

$$\mathcal{P} : (\hat{\mathbf{y}}, \mathbf{y}) \mapsto p \tag{3.17}$$

Usually, we also divide the nodes into a training set V_{train} and a testing set V_{test}. As the name suggests, the training set is used for algorithm training and testing set for evaluation on never before seen nodes. This implies transductive learning setup, where all nodes were seen by the algorithm during the training phase but only a subset (V_{train}) had the labels available. Inductive learning, on the other hand, is shown never-before seen observations during the testing phase. Although our main focus is on transductive setup, there is no reason why this method should not work in the inductive setup.

One more notation I am going to use from now on is the subset of nodes belonging to class k:

$$V^{(k)} = \{u \in V \mid y_u = k\} \tag{3.18}$$

Neighbors of node u belonging to class k will be denoted $\mathcal{N}(u)^{(k)}$ and node u's class degree $d(u)^{(k)}$:

$$\mathcal{N}(u)^{(k)} = \mathcal{N}(u) \cap V^{(k)} \tag{3.19}$$

$$d(u)^{(k)} = |\mathcal{N}(u)^{(k)}| \tag{3.20}$$

[1] Edge features are also possible but will not be considered in this work.

3.3.2 Information Retrieval

As defined in [23]: "Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)." The primary goal of IR is to obtain relevant information to a particular search query and rank the results based on their relevance.

Due to the varying capacity of users to consume information, the relevant results can be further reduced to a certain maximum number, also known as "window size". This is where ranking of the relevant results comes into play: it aims to sort the obtained results from the most relevant to the least relevant. A common example of an IR system with ranking is a search engine, which strives to return the most relevant websites based on the user's search query, with the most pertinent results appearing at the beginning of the list.

It is important to note that IR is typically an imbalanced problem: there are often significantly more irrelevant results than relevant ones. This poses a challenge to developing effective retrieval systems that accurately identify and prioritize relevant information.

Let us now consider the IR task in the context of graphs. This task is different from the original concept in many ways, but I will use the same terminology when possible. In this scenario, the objective is to retrieve nodes with specific properties not directly present in the input data. This can be thought of as a special case of node classification.

The main distinction between IR and node classification is obtaining the top k relevant results in the retrieval context. Here, k represents the window size, i.e. the number of relevant results to be retrieved. To evaluate the performance of such a retrieval-based node classification system, we can employ the precision at k metric, also referred to as $prec@k$, which measures the proportion of relevant results in the top k retrieved items. Suppose we rank the nodes based on the class probability for the target class to be retrieved t and number them from 1 to n with v_1 having the highest probability for class t. Then given the true class labels y_i for each node, the equation for calculating $prec@k$ is as follows:

$$prec@k = \frac{\sum_{i=1}^{k}[y_i = t]}{k} \tag{3.21}$$

where $[P]$ is the Iverson bracket, which for statement P is defined as:

$$[P] = \begin{cases} 1 & \text{if } P \text{ is true,} \\ 0 & \text{otherwise.} \end{cases} \tag{3.22}$$

Chapter 4

Algorithmic Foundations and Theoretical Framework

In this chapter, the theory behind the algorithms used for the experimental part of this work is going to be introduced. This includes the general working principles of GNNs, decision trees and how they are used together in a Random Forest algorithm, and how a concept of Shapley values is used in machine learning for model interpretability. In the thesis flowchart (Figure 4.1), this corresponds to the graph algorithm, regression meta-model, and model interpretation blocks.

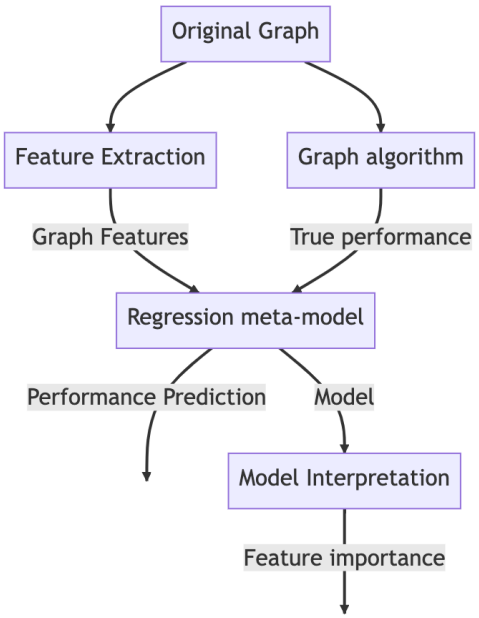


Figure 4.1: Thesis approach illustration. Graph features of the original graph and task performance dependant on the algorithm are used to train the regression meta-model. Meta-model provides the feature importance which can be used to optimize the original graph to maximize task performance. Meta-model's performance prediction is not the main goal and serves only as a validation to verify the model reflects graph structure and task and doesn't just over-fit on the training data. (Repeated Figure 1.1.)

4.1 Machine Learning Graph Algorithms

Machine learning graph algorithms operate on an input graph. An algorithm can be limited to a specific graph type, such as directed, undirected, weighted, and more. In some cases, graphs can be converted to another type under some assumptions; for example, an unweighted graph to weighted with the same weight.

In the past, traditional ML methods were applied to graphs usually using structure-agnostic approach utilizing only the node features. For probability networks, for example Bayesian networks or Markov random fields, bayesian message-passing framework could be used to utilize the graph structure. One such algorithm is Belief Propagation [12].

With the growing demand for network applications such as social network analysis or recommendation systems, new methods appeared such as node2vec [13]. Node2vec is an algorithm generating an embedding based on random walks through the graph, which can then be used for downstream tasks such as node classification.

With the rise of deep learning, different architectures in the family of Graph Neural Networks (GNNs) were introduced in the past few years. GNNs are

designed to combine both the graph topology and node features. This has made them popular choice for graph-structured data applications.

4.1.1 Graph Neural Networks

The evolution of GNNs is well documented and explained in [24] which has also been used as the main source for this section. For the node classification task the main goal is to calculate a node embedding $\mathbf{h}_u$ for all nodes $u \in V$ dependent on both the node feature information $\mathbf{x}_u$ and the graph topology captured by edges $e \in E$.

The main difference between graph deep learning and more classical deep learning such as Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs) is that the classical approach performs the computations on euclidean data in the CNN case or sequences in the RNN's while GNNs have to work in non-euclidean space. This also inspired one of the two works where graph networks initially emerged: authors of [25] were attempting to generalize convolutional networks for graph data. The other work was [26] which proposed a belief propagation algorithm with differentiation.

The solve the problem of non-euclidean space, GNN architectures utilize a form of message passing to exchange information between the nodes in the graph $u \in V$. This can be divided into two component functions: UPDATE and AGGREGATE. In interation k, the AGGREGATE function distills the information from node u's neighborhood into a message vector $\mathbf{m}_{\mathcal{N}(u)}^{(k)}$. Then the UPDATE function updates the embedding using the acquired message, creating $h_u^{(k+1)}$ to be used in the next iteration $k+1$.

$$\mathbf{m}_{\mathcal{N}(u)}^{(k)} = \text{AGGREGATE}^{(k)}\left(\{\mathbf{h}_v^{(k)}, \forall v \in \mathcal{N}(u)\}\right) \tag{4.1}$$

$$\mathbf{h}_u^{(k+1)} = \text{UPDATE}^{(k)}\left(\mathbf{h}_u^{(k)}, \mathbf{m}_{\mathcal{N}(u)}^{(k)}\right) \tag{4.2}$$

These functions are usually separate differentiable and learnable functions such as an instance of multi-layer perceptron (MLP). The initial embedding have to be initialized by using the original node feature vectors $\mathbf{x}_u$, therefore:

$$\mathbf{h}_u^{(0)} = \mathbf{x}_u \tag{4.3}$$

After the first message passing iteration at $k = 1$, each node's embedding is updated using the aggregated information from 1-hop neighborhood.

There are many possible improvements for the AGGREGATE function. [27] proposes a Graph Convolutional Network (GCN) using an aggregation function with symmetric normalization. It normalizes the embedding vector of each node v in the u's neighborhood $\mathbf{h}_v$ based on the size of $\mathcal{N}_u$ and $\mathcal{N}_v$ before the aggregation. In [28]'s GraphSAGE, normalization in conjunction with a learnable node pooling function to sample the neighboring nodes achieves comparable results while significantly reducing the computational

complexity. Authors of [29] propose using learnable neighborhood attention in the aggregation function resulting in Graph Attention Network (GAT).

Modifying the UPDATE function is also a viable research direction. Concatenation and skip-connections aim to elevate the embedding from the previous iteration by using one function to combine the embedding with the update message and then a second function to combine the new embedding with the one from previous step. Alternative approach of Jumping Knowledge Connections produces the final embedding by combining the node's embedding from each step instead of simply using the last one. Gated updates inspired by RNNs such as Gated Recurrent Unit (GRU) or Long Short-Term Memory (LSTM) networks have also been used to improve learnability by using the node embedding instead of the RNNs hidden state.

All of these modifications aim to solve a common issue of GNNs when after multiple iterations the node embeddings become very similar to each other. This is reffered to as oversmoothing. It has been a persistent issue preventing deeper GNNs models.

Up until this point we have been describing message passing concerned only with node features. Approaches considering multiple edge types or edge features also exist. RGCN architecture from [15] is able to deal with multiple edge types by using different weight matrices. Simple solution for considering edge features is to concatenate with the node features them during the neighborhood aggregation phase.

Once the final node embedding $\mathbf{z}_u$ is calculated, we can use it to solve the desired task. For node classification, each embedding is used to calculate a label using another learnable function:

$$y_u = f_1(\mathbf{z}_u) \tag{4.4}$$

For edge classification, edge source and target nodes along with the edge feature (if available) are taken into account:

$$y_{uv} = f_2(\mathbf{z}_u, \mathbf{z}_v, e_{uv}) \tag{4.5}$$

Graph classification can be solved by either pooling all of the nodes to receive a graph classification or can use graph coarsening, which leverages the graph topology once more for the final judgement. During the coarsening process, edges are collapsed and neighboring nodes joined together until a final node is produced distilling the information of the whole graph and producing the final graph label. Other tasks on graphs exist such as relation prediction or many forms of regression which build upon the described concepts but are not relevant to this work.

Each of the described approaches can be considered a separate GNN architecture. Making a fair comparison between multiple architectures would require using the same computational resources including memory and using optimal hyperparameters for each one which is itself a difficult (and unsolved) task. We are instead going to explore how hyperparameters affect performance

of a specific architecture, GraphSAGE. Direct inspirations was [18], where a design space for GNNs is proposed.

4.2 Regression Algorithms

This section describes the meta-model block in the flowchart in Figure 4.1. It will be used to predict the performance based on the graph features (defined in Section 5.4 alone without the access to the original graph. Multiple algorithms were considered but because of the relatively small number of datapoints in the case of the Risk Map Graph (Section 5.3.2, 56 datapoints[1]), the model needs to be a rather simple one. Any gradient based method, such as multi-layer perceptron, would fail to generalize on such small dataset and more samples were not possible to obtain within the limited time frame.

Simple linear regression or decision trees usually perform well even on smaller datasets. However, some non-linear dependencies of the graph features are to be expected. For example, high value of feature 1 can be an indicator of good performance if feature 2 is low, however high feature 1 can indicate the exact opposite when feature 2 also has high value. This rules out linear regression but can be captuterd by decision trees. Furthermore, a random forest model can perform even better than a decision tree, especially in the case of many variables, because it utilizes multiple decision trees each time on a subset of variables to avoid overfitting.

4.2.1 Decision Trees

Decision trees are a supervised ML algorithm supporting both classification and regression [30]. It can deal with both continuous and categorical variables and is computationally inexpensive in today's standards. When given a set of training samples, samples are repeatedly splitted into subgroups by a selected variable. These splits are represented as tree nodes. Once there are no more variables on which to split the subgroups, a leaf node containing one or more samples is created.

The variable to split is selected to minimize the impurity measure such as Gini impurity in the classification scenario, which bascially expresses the error given a specific classification label. In the regression scenario, variable is chosen to minimize the loss function such as mean squared error as much as possible.

For a decision tree to be able to generalize, the maximum depth has to be limited. If each leaf node of the tree contains one sample in the end, the tree becomes a lookup table and the model is necessarily overfitted.

[1] 14 days times 4 modality types

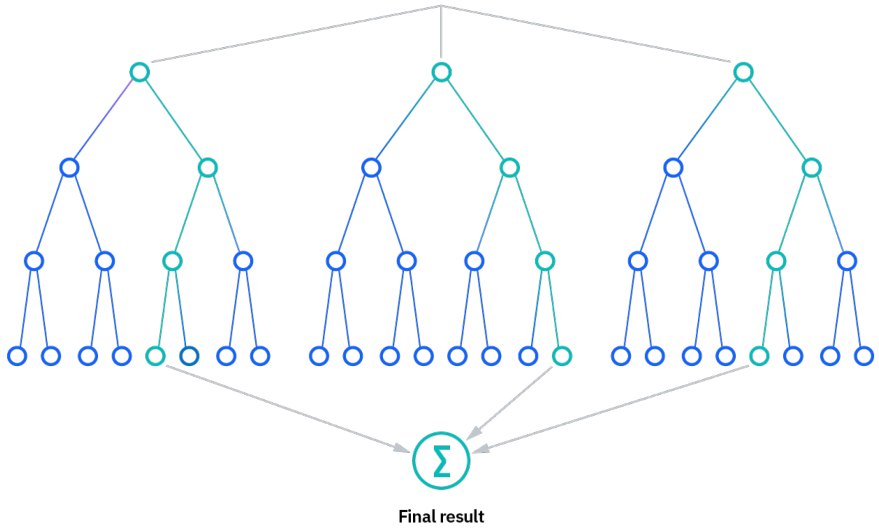


Figure 4.2: Illustration of Random Forest and its underlying decision trees [33].

4.2.2 Random Forests

A Random Forest ([31], [32]), as the name suggests, consists of multiple decision trees. Each of the trees operates on a subset of variables a tries to minimize the impurity (classification) or loss (regression). The results of each tree are then recorded and either the majority vote (classification) or an average result (regression) is given as an output.

The main benefit of combining multiple trees together is increased resistance to overfitting as the trees are uncorrelated[2] since they are using different subset of features. According to [31], "using a random selection of features to split each node yields error rates that compare favorably to Adaboost".

Random Forest also provides a feature importance information based on the splitting. However, this information can be biased towards high cardinality features ([34], [35]), which give the algorithm option to obtain better split and form small sized leaf groups. To avoid this issue, permutation based feature importance approach, specifically Shapley Additive Explanations, is going to be used.

4.3 Model Interpretation

In this section I am going to provide a tool to describe how to interpret the regression meta-model and analyze the importance of variables used to train it. As described in the previous section, the model used for regression is not directly interpretable as a linear regression or a single decision tree would be, therefore some tool needs to be used. It is worth noting that other methods

[2] If the features are also not correalted.

exist, such as Local interpretable model-agnostic explanations (LIME) [36]. However, the presented method is easy to implement, has a strong theoretical foundation, which is also presented in the next section, and has an efficient calculation method for tree-based algorithms.

4.3.1 Shapley Value

The Shapley value, introduced in [37], is a concept originally developed in coalitional game theory that provides a solution to the problem of distributing the reward among a coalition of players based on their contribution to the goal. This concept is based on the idea that the total value generated by a group of players is a function of the individual contributions of each player.

It is computed as the marginal contribution of each player to each possible coalition meaning each player's contribution is calculated based on the reward of each possible coalition with and without them.

Mathematically speaking, Shapley value $\varphi^S(v)$ is a mapping from the set of all games over a set of players Γ to a real values vector.

$$\varphi^S : \Gamma \to \mathbb{R}^n, \varphi^S = (\varphi_1^S, ..., \varphi_n^S) \tag{4.6}$$

where $\varphi_i^S(v)$ is the Shapley value of player i. Given a set of players $N, |N| = n$ and value of the game for coalition A as $v(A)$, Shapley value for player i can be computed as:

$$\varphi_i^S(v) = \sum_{A \subseteq N \setminus i} \frac{|A|!(n - |A| - 1)!}{n!} (v(A \cup i) - v(A)) \tag{4.7}$$

One of the key benefits of the Shapley value is that it is additive, efficient, symmetric and satisfies the null player property. Let Γ be the set of all games over the set of players N. Additivity axiom means that the total value of the coalition is equal to the sum of the values of each individual player. Therefore, for games $u, v \in \Gamma$

$$\varphi^S(u + v) = \varphi^S(u) + \varphi^S(v) \tag{4.8}$$

Efficiency states that the cumulative reward received by individual player of the coalition must be exactly equal to the total reward obtained, so that no reward is undistributed. Mathematically:

$$\varphi_1(v) + ... + \varphi_n(v) = v(N) \tag{4.9}$$

Additionally, the Shapley value is symmetric, which means that the value assigned to each player is the same regardless of the order in which the players are added to the coalition. Consider players $i, j \in N$, a coalition of players $A \subseteq N \setminus i, j$ and a game v. Then:

$$v(A \cup i) = v(A \cup j) \implies \varphi_i(v) = \varphi_j(v) \tag{4.10}$$

Finally, the Shapley value satisfies the null player property, which means that a player who does not contribute anything to the coalition is assigned a value of zero.

$$v(A \cup i) = v(A) \implies \varphi_i(v) = 0 \tag{4.11}$$

This notation was taken from [38].

Overall, the Shapley value is a powerful tool for solving cooperative games and is widely used in various fields, including economics, political science, and computer science.

In the computer science field, Shapley value can be used as is for feature importance computation. However, as all marginal contributions have to be computed, it becomes rather computationally expensive with 2^n possible combinations where n is the number of players (or features in feature explanation scenario). Thankfully, polynomial approximation methods for Shapley value computation exist as well as methods building on top of this concept, such as Shapley Additive Explanations.

4.3.2 SHAP

Shapley Additive Explanations (SHAP) [22] is a method for interpreting machine learning models that is based on the Shapley value. It is able to explain individual predictions or combine them together to provide a global explanation, therefore the *additive* explanation. Interpretation of which values of the features (high/low) benefit the final prediction and which decrease it is also available. That is not possible with simpler interpretations such as Random Forest built-in feature importance. Combination of these informations allows us to gain insights into how the model works and why it makes certain decisions. For thorough explanation please refer to the original paper [22] or [39, ch. 9] which has also been used to improve our understanding.

The basic idea behind SHAP is that the feature values of a data instance act as players in a coalition. The Shapley value tells us how to fairly distribute the payout (i.e., the prediction) among the features. In other words, it provides a way to measure the contribution of each feature to the final prediction, taking into account the interactions between features.

The SHAP method introduces two computation methods: KernelSHAP and TreeSHAP. KernelSHAP is model agnostic and can be applied to any ML model, while TreeSHAP is specific to tree-based models. TreeSHAP is particularly efficient as it computes in polynomial time instead of exponential thanks to the tree structure. However, there are also drawbacks to the TreeSHAP approximation, such that it "can assign non-zero attributions to features that are not even referenced by the model" as shown in [40].

One of the strengths of SHAP is that because of its theoretical Shapley value foundation the four axioms of additivity, efficiency, symetry and the null

player property hold even for this method. This means that the attribution of the prediction to each feature is consistent and fair.

Overall, SHAP provides a way to gain insights into how the model works and why it makes certain decisions, which is particularly important in applications where transparency and accountability are crucial. By introducing TreeSHAP, it also makes it possible to compute exact Shapley values for a wide variety of tree-based models in polynomial time.

We will be using the TreeSHAP computation method for the explanation of the meta-model and interpreting the contribution of each feature. Additionally, we will be able to find out whether each feature should increase or decrease to improve the performance.

Chapter 5

Datasets, Algorithms and Graph Features

This chapter starts with a formal problem formulations. Next section will focus on the graph data used for the experimental evaluation and the specifics of machine learning graph algorithms used to evaluate performance. Graph features to capture the graph data reprerentation are going to be defined and last section will describe how all of these information build together the distilled retrieval task representation for the regression meta-model.

5.1 Formal Problem Description

The goal of this thesis is to propose a systematical way of selecting or creating relations in a way that maximizes downstream task performance. On the input, we have the network telemetry containing multiple relations $\mathcal{R}$.

Because the construction of the graph G is given by the nature of the telemetry, we focused on selecting an optimal subset of relation types $\mathcal{R}^* \subseteq \mathcal{R}$ in order to maximize performance metric $\mathcal{P}$ of graph ML algorithm $\mathcal{A}$. This subset of realations $\mathcal{R}^*$ implicitly defines a subgraph G^* on the original graph G.

Our approach is to create a feature vector $f(G)$ and use the algorithm $\mathcal{A}$ performance $\mathcal{P}$ on graph G as a label for regression meta-model $\mathcal{M}$. The goal of the model $\mathcal{M}$ is to predict the performance $\mathcal{P}$. This prediction will be reffered to as $\hat{\mathcal{P}}$.

Regression meta-model $\mathcal{M}$ will be analyzed to find out which variables of the feature vector $f(G)$ are most influential to the prediction $\hat{\mathcal{P}}$. We will also use the available tools to figure out whether the feature should be increased or decreased to optimize the performance metric. This is captured in the foundational Hypothesis 1.1.

Once the important features are selected, we will be able to evaluate different subgraphs purely based on the feature vector $f(G)$. As each subset of relations $\mathcal{R}' \subseteq \mathcal{R}$ can be assigned this feature vectors, the relation type selection is then reduced to finding the optimal relations subset $\mathcal{R}^*$.

5.2 Datasets

As mentioned earlier, this work is focusing on a retrieval task performed with custom algorithm on private data and a more general approach utilizing GNNs on public datasets.

5.2.1 Public Datasets

GNNs are often benchmarked on citation networks. In this work I adopted the Cora [41] and DBLP [42] datasets in the form introduced in [43]. In citation networks, nodes are specific papers and edges stand for citation between the papers. Cora's node features represent presence or absence (zero or one) of selected words in the text. PubMed [44] is another citation network. Node feature vectors are bag-of-words representations of the papers.

The largest citation network used is ArXiv from the Open Graph Benchmark collection [45]. It contains only computer science related papers with directed edges indicating that one paper cites the other which will be modified to be undirected. Features are 128-dimensional averaged embeddings of the words from the abstract and title.

In order not be limited only to citation networks, we will also include a few more datasets. Squirrel is a Wikipedia dataset initially created for a regression task of predicting average monthly traffic from [46]. Nodes of the graph are webpages from English Wikipedia as of December 2018 containing the topic of squirrels and edges as links between them. Node fetaures correspond to selected words occuring in the text. We will be using a modification of this dataset introduced in [16], where the nodes are divided into five classes based on the average monthly traffic of the page.

Amazon Computers from [47] is a graph where nodes are products and edges link two products when they are frequently bought together. Features are bag-of-words vectors based on the product reviews and class is the product category.

In the Flickr dataset [48] images are represented by nodes. Edge exists between two images if they share common properties such as geographical location, same user interaction or occur in the same gallery. Node features are based on a 500-dimensional bag-of-words representation of image descriptions.

Table 5.1 shows some basic statistical properties of each of the datasets. All of the datasets consist of a single graph.

5.2.2 Adapting Public Datasets for Retrieval Task

As the original task on used public datasets is multiclass node, they have to be modified for the node retrieval task (Section 3.3.2), which is a special case

Dataset	PubMed	DBLP	Cora	Squirrel
Number of nodes	19 717	17 716	19 793	5 201
Number of edges	88 648	105 734	126 842	217 073
Number of features	500	1 639	8 710	2 089
Number of classes	3	4	70	5
Homophily	0.79	0.81	0.59	0.09
Dataset	Computers	Flickr	ArXiv	
Number of nodes	13 752	89 250	169 343	
Number of edges	491 722	899 756	1 166 243	
Number of features	767	500	128	
Number of classes	10	7	40	
Homophily	0.79	0.32	0.43	

Table 5.1: Statistics of the used public datasets. The homophily value represents node homophily as defined by [16].

of binary node classification. We are going to create multiple retrieval tasks from each graph, each time retrieving a specific class (target class) denoted t.

The modification strategy is described below.

1. Select target class $t \in C$ as the class to be retrieved.
2. Relabel other classes $C \setminus t$ to class 0 (negative).
3. Relabel target class t to class 1 (positive).

This conversion effectively increases the number of retrieval datasets to the total number of classes from all datasets (139).

This approach comes with both its drawback and an advantage. The drawback is that graph features, defined in the next Section 5.4, have to be defined in a way to give different results based on the node class labels in order to be differentiable across the same dataset. However, this can be also considered an advantage as such graph features should reflect the distribution and structural relations between positive and negative nodes more accurately.

5.2.3 Private Datasets

As menitoned previously, the second part of this work is focused on Cisco use-case of detecting malicious domains (IoC retrieval). This requires specific data to work with, in our case we are going to use Stealthwatch flows.

Stealthwatch is a security analytics solution by Cisco. It uses either dedicated network probes, routers and switches to monitor network communication or flows. As defined in RFC 7011 [49]: "A Flow is defined as a set of packets or frames passing an Observation Point in the network during a certain time interval. All packets belonging to a particular Flow have a set of common

`secondleveldomain`	Second level domain URL.
`serverip`	IP to which the second level domain was resolved using DNS.
`networkdeviceid`	Unique identifier of the user network device.
`autonomoussystem`	Autonomous system of the server.
`tlsrequestfingerprintja3`	Client fingerprint computed from the client hello packet.

Table 5.2: Description of available network flow properties.

properties." Those common properties are for example source and target IP address or port.

Based on the analysed traffic, a baseline of normal network activity is created mainly so that anomalous events can be more easily distinguished. In this work, only the anomalous subset of the Stealthwatch telemetry from the anomaly detection layer will be used.

The telemetry flows contain multiple properties of which we are going to use a subset. Selected properties are listed in Table 5.2 along with their brief descriptions. Each of those properties can be an indicator of a malicious activity when shared with a known malicious domain. Domains (SLDs) resolving to the same IP address (`serverip`) as a known threat is suspicious or when a device tries to connect to a blocked known botnet command and control (C&C) server and so it tries an alternative C&C right after that (`networkdeviceid`). Note that there are no additional features included in the data besides these properties.

All of these flows together create a multipartite graph with multiple edge and node types (five to be exact). The graph is 5-partite as there are no edges between the same type of flow property. However, IoC retrieval is only interested in retrieving nodes from the SLD node set and the other node sets are used only as features. We are also going to refer those node and edge types as **modalities**. To evaluate each modality (e.g. "networkdeviceid" modality) on its own, only bipartite subgraph of the whole flow graph will be considered at once. This results in four bipartite graphs per day, therefore total of 56 datapoints in total from the 14-day network telemetry data.

Because the original data cannot be shared as a part of the thesis, we will provide at least some basic statistical information about the data in Table 5.3. The reason days 18, 19, 25, 26 have around half of the edges (flows) than the others is because they are weekends when less traffic is commonly observed than on week days.

Date	14	15	16	17	18
Edges	19 005 567	18 059 254	18 207 499	18 060 270	9 242 554
SLDs	365 364	329 932	325 634	322 409	157 867
Date	19	20	21	22	23
Edges	8 484 966	19 109 058	19 776 480	20 049 115	19 915 321
SLDs	148 728	320 565	314 912	313 055	319 887
Date	24	25	26	27	
Edges	18 628 743	9 838 469	8 952 773	18 918 494	
SLDs	308 940	166 405	150 366	315 348	

Table 5.3: Statistics of the used private datasets.

5.3 Specific Machine Learning Graph Algorithms

5.3.1 GraphSAGE

GraphSAGE is one of the very popular GNN architectures proposed in [28]. It is much faster compared to other popular choices such as GCN [16] or GAT [29] thanks to utilizing uniform sampling while constructing the message to be passed in the AGGREGATE function while keeping comparable performance.

The AGGREGATE and UPDATE functions from Equation 4.1 take the following form:

$$\mathbf{m}_{\mathcal{N}_s(u)}^{(k)} = \text{AGGREGATE}^{(k)}\left(\{\mathbf{h}_v^{(k-1)}, \forall v \in \mathcal{N}_s(u)\}\right) \tag{5.1}$$

$$\mathbf{h}_u^{(k+1)} = \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}\left(\mathbf{h}_u^{(k-1)}, \mathbf{m}_{\mathcal{N}_s(u)}^{(k)}\right)\right) \tag{5.2}$$

The uniform sampling of neighboring nodes is captured by $\mathcal{N}_s(u)$ which is a modified neighborhood nodes set (s for sampling). Other than that the first equation keeps it's ability to use arbitrary differentiable function to aggregate the sampled embeddings $h_v^{(k-1)}$. UPDATE function is defined with σ acting as the activation function and a trainable weight matrix $\mathbf{W}^k$ controlling the information propagation between different layers of the model.

However, as hyperparameter tuning is an unsolved problem and can have an enormous impact on performance, I decided to experiment with different hyperparameters of this GNN architecture. This approach was directly inspired by [18] already mentioned in related literature in Section 2.2. Design space parameters used in this work are described in Table 5.4.

5.3.2 Risk Map Graph

Risk Map Graph is a custom Cisco algorithm used for malicious domain retrieval, also called Indicators of Compromise (IoC) retrieval. It is based on the label propagation algorithm (LPA) [50], which propagates labels of

Hyperparameter	Values
Layers	1, 2, 3
Hidden channels	16, 32, 64
Dropout probability	0, 0.3, 0.6
Activation function	ReLU, PReLU
Aggregation function	mean, max

Table 5.4: Design space considered for the GraphSAGE architecture. Inspired by [18].

known labeled nodes to unknown nodes through the graph edges. It can be used in scenarios with few labeled nodes and has guaranteed convergence.

LPA was modified to operate with multiple modalities at once. Thanks to the algorithm's simplicity it scales very well on the large network telemetry data. Because of the confidentiality restrictions I am not at liberty to provide the implementation of the algorithm. Nevertheless, the implementation is not the main focus of this thesis and for the feature importance explanations information about the LPA algorithm should be sufficient. Even a black-box approach would be enough for studying the effects of graph features on the resulting performance using presented methods as long as there is a way of obtaining the performance.

5.4 Graph Features

This section describes the graph features, whose goal is to capture properties of the graph which are important for the task performance. In the thesis flowchart 5.1 this corresponds to the feature extraction block. Some well defined concepts, such as homophily, already exist. I am going to employ both graph features used in the literature and attempt to suggest new ones.

The idea behind graph features is that nodes belonging to the same class should:

1. have similar nodes features if available
2. be connected by an edge

This esentially describes the concept of homophily, which is thoroughly explored in [51]. The main idea is that nodes "soak up" the features from their neighborhood making them more alike and thus more likely to be assigned to the same class. Because of this phenomenon, homophily is usually an assumption for the graph but not necessarily true. This is why the two concepts mentioned above should summarize the properties of a good performing graph fairly well.

However, in a real-world scenario, only a subset of true labels (training set) is available to us and therefore there's no way of evaluating these properties.

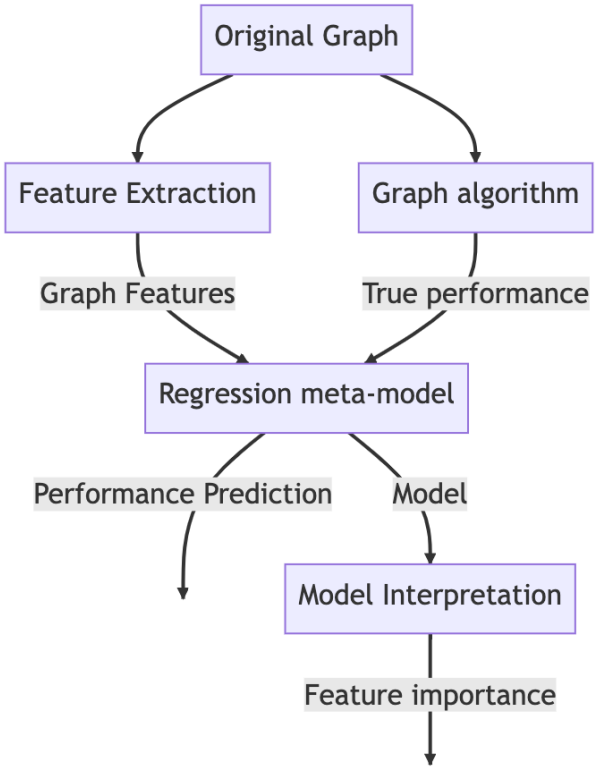


Figure 5.1: Thesis approach illustration. Graph features of the original graph and task performance dependant on the algorithm are used to train the regression meta-model. Meta-model provides the feature importance which can be used to optimize the original graph to maximize task performance. Meta-model's performance prediction is not the main goal and serves only as a validation to verify the model reflects graph structure and task and doesn't just over-fit on the training data. (Repeated Figure 1.1.)

We are only able to compute their value based on the training subset which provides some form of estimation.

In the following sections, I am going to present graph features based on these concepts. First I am going to explore the graph features computed on the unipartite graph of public datasets, then on the bipartite graph for network telemetries and finally define graph features that can be used for both types of graphs.

5.4.1 Unipartite Graph Features

For the public datasets, the graph is unipartite meaning each node can be connected to any other node. Datasets also come with a training and testing masks which divide the nodes into two groups. Therefore, the graph feature computation can use both of the positive and negative class but only on the training nodes.

Structure Agnostic

Definition 5.1 (Number of nodes)**.** For graph $G = (V, E)$ the number of nodes is equal to

$$\texttt{nodes} = |V| \tag{5.3}$$

Definition 5.2 (Number of positive training nodes - `num_positive`)**.**

$$\texttt{num\_positive} = |\{u_i \mid u_i \in V_{train} \wedge u_i \in V^{(1)}\}| \tag{5.4}$$

Definition 5.3 (Number of negative training nodes)**.**

$$\texttt{num\_negative} = |\{u_i \mid u_i \in V_{train} \wedge u_i \in V^{(0)}\}| \tag{5.5}$$

Definition 5.4 (Class feature MSE)**.** This feature was designed specifically for this work to capture the feature vector difference between the classes. It is a four element matrix computing the class MSE from average feature vector. Consider $\bar{\mathbf{x}}_k$ as an average feature vector of class k and $V_{train}^{(k)}$ set of training nodes in its class. Let's call $\bar{\mathbf{x}}_k$ the average feature vector of class k:

$$\bar{\mathbf{x}}_k = \frac{1}{|V_{train}^{(k)}|} \sum_{v \in V_{train}^{(k)}} \mathbf{x}_v \tag{5.6}$$

Then the class feature MSE matrix is:

$$M = \begin{bmatrix} m_{1,1} & m_{1,2} \\ m_{2,1} & m_{2,2} \end{bmatrix} = \begin{bmatrix} \sum_{v \in V_{train}^{(0)}} (\mathbf{x}_v - \bar{\mathbf{x}}_0)^2 & \sum_{v \in V_{train}^{(0)}} (\mathbf{x}_v - \bar{\mathbf{x}}_1)^2 \\ \sum_{v \in V_{train}^{(1)}} (\mathbf{x}_v - \bar{\mathbf{x}}_0)^2 & \sum_{v \in V_{train}^{(1)}} (\mathbf{x}_v - \bar{\mathbf{x}}_1)^2 \end{bmatrix} \tag{5.7}$$

$$\texttt{feat\_dist\_XY} = m_{X,Y} \tag{5.8}$$

Structure Aware

Definition 5.5 (Number of connected components)**.** Component of a graph is a maximal set of nodes between which there exists a path. Path is an order of nodes such that there exists an edge between each two consecutive nodes. Number of connected components is the number of such node sets. The minimum number of connected components is one, meaning there exists a path between all of the nodes.

$$\texttt{num\_components} \in \mathbb{N} \tag{5.9}$$

Definition 5.6 (Number of edges)**.**

$$\texttt{edges} = |E| \tag{5.10}$$

Definition 5.7 (Positive node ratio from nodes of degree $> N$)**.**

$$\texttt{pos\_node\_ratio\_N} = \frac{|\{u_i \mid u_i \in V_{train} \wedge u_i \in V^{(1)} \wedge d(u_i) > N\}|}{|\{deg(u_i) > N\}|} \tag{5.11}$$

Definition 5.8 (Average node degree)**.**

$$\texttt{degree\_avg} = \frac{\sum_{u \in V} d(u)}{|V|} \tag{5.12}$$

Definition 5.9 (Average positive node degree)**.**

$$\texttt{degree\_pos\_avg} = \frac{\sum_{u \in V^{(1)}} d(u)}{|V^{(1)}|} \tag{5.13}$$

Definition 5.10 (Assortativity)**.** Assortativity indicates how likely are nodes in the graph connected to similar nodes based on node degree. More details can be found in [52]. Result is a real number:

$$\texttt{assortativity} \in \langle -1; 1 \rangle \tag{5.14}$$

Definition 5.11 (Conductance)**.** Conductance is a measure of how many edges lead between two disjunctive node sets normalized by the number of edges inside the larger set. In this work, the conductance is calculated on a cut given by positive training nodes $V^{(1)}_{train}$ and the other nodes.

$$E_A = \{\{u, v\} \in E \mid u, v \in V^{(1)}_{train}\} \tag{5.15}$$

$$E_B = \{\{u, v\} \in E \mid u, v \in V \setminus V^{(1)}_{train}\} \tag{5.16}$$

$$\texttt{conductance} = \frac{|\{\{u, v\} \in E \mid u \in V^{(1)}_{train} \wedge v \in V \setminus V^{(1)}_{train}\}|}{max(|E_A|, |E_B|)} \tag{5.17}$$

Definition 5.12 (Neighborhood cosine similarity)**.** Second feature designed specifically for this work. Neighborhood cosine similarity represents average cosine similarity between the node's feature vector $\mathbf{x}_v$ and its neighborhood's average feature vector $\bar{\mathbf{x}}_{\mathcal{N}(v)}$. As expected, this only considers the feature vector's direction and not magnitude. I am expecting magnitude to be captured by one of the other feature based metrics.

$$\bar{\mathbf{x}}_{\mathcal{N}(v)} = \frac{1}{|\mathcal{N}(v)|} \sum_{u \in \mathcal{N}(v)} \mathbf{x}_u \tag{5.18}$$

$$\texttt{cos\_similarity} = \frac{1}{|V|} \sum_{v \in V} \frac{\mathbf{x}_v \cdot \bar{\mathbf{x}}_{\mathcal{N}(v)}}{||\mathbf{x}_v||\ ||\bar{\mathbf{x}}_{\mathcal{N}(v)}||} \tag{5.19}$$

Definition 5.13 (Attribute homophily)**.** Homophily in general is a measure of node similarity based on class labels. Attribute homophily rate proposed in [9] is attempting to give similar information but based on node features. The attribute homophily rate is formulated for each feature f independently as:

$$\beta_f = \frac{1}{\sum_{v \in V} x_{vf}} \sum_{v \in V} \beta_{vf} = \frac{1}{\sum_{v \in V} x_{vf}} \sum_{v \in V} \left(x_{vf} \frac{\sum_{u \in \mathcal{N}(v)} x_{uf}}{d(v)} \right) \tag{5.20}$$

We can see the first fraction is a feature normalization as it adds up all of the feature f's values over all nodes in the graph. β_{vf} is the feature value of

node v times average feature value of the nodes in its neighborhood. I am going to take an average attribute homophily rate computed as follows:

$$\beta = \frac{1}{|F|} \sum_{f \in F} \beta_f \tag{5.21}$$

where F is the set of all features. I am going to calculate attribute homophily for nodes $u \in V_{train}^{(1)}$.

Definition 5.14 (Edge homophily)**.** Edge homophily from [51] is ratio between edges connecting the same class and total number of edges. I am going to adjust the calculation by dividing the node sets into $V_{train}^{(1)}$ and $V \setminus V_{train}^{(1)}$ to receive different values based on the retrieved class $k = 1$ from the original class labels.

$$\texttt{homophily\_edge} \tag{5.22}$$

Definition 5.15 (Node homophily)**.** Node homophily [16] is edge homophily normalized by the size of the neighborhood for each node. Similarly to edge homophily, I am going to divide the nodes to $V_{train}^{(1)}$ and $V \setminus V_{train}^{(1)}$ and calculate the value based on them.

$$\texttt{homophily\_node} \tag{5.23}$$

5.4.2 Bipartite Graph Features

Network flows are bipartite graphs. In the node retrieval task, only a starting set of positive nodes (seeds, class 1) is specified, all other nodes are unknown (class 0). I am going to use unknown and negative interchangably in the case of network telemetry graphs. Because of the absence of training nodes for the negative (unknown) class and extremely large number of edges, some unipartite metric could not be computed at all. Therefore, I am going to suggest graph features specifically for bipartite graphs.

Structure Agnostic

Definition 5.16 (Number of nodes)**.** Number of nodes for bipartite graphs will be given but the number of SLD nodes in the graph. This corresponds to the number of nodes in unipartite graph induced by the bipartite graph, as described in Section 3.2.1.

$$\texttt{nodes} = |V_{domains}| \tag{5.24}$$

Definition 5.17 (Number of modality nodes)**.** Number of modality nodes can be computed by subtracting the number of unique SLDs (number of nodes in the first part of the graph) form the total number of nodes in the graph.

$$\texttt{nodes\_mod} = |V_{modality}| = |V| - |V_{domains}| \tag{5.25}$$

Definition 5.18 (Number of positive nodes)**.** Number of positive nodes (SLDs).

$$\texttt{nodes\_pos} \tag{5.26}$$

Definition 5.19 (Number of negative nodes)**.** Number of negative nodes (SLDs).

$$\texttt{nodes\_neg} \tag{5.27}$$

Structure Aware

Definition 5.20 (Clique number)**.** Clique number of a graph is another property used in graph theory. Clique in a graph G is a subset of nodes which induce a complete subgraph on G. Clique number of a graph is then equal to the number of node in the largest clique.

$$\texttt{clique} \tag{5.28}$$

Definition 5.21 (Expansion ratio)**.** Expansion ratio describes how many nodes are in the neighborhood of positive modalities compared to the number of seeds. This gives an idea how many nodes can the information propagation reach in one step.

$$\texttt{expansion} = \frac{\sum_{(u,v)\in E^1} d(v)^{(1)}}{|S|} \tag{5.29}$$

Definition 5.22 (Mean modality positivity)**.** The first one, mean modality positivity, indicates how many seeds are on average connected to the modalities with at least one positive node (positive modalities) in their neighborhood. To define it mathematically:

$$\texttt{mod\_positivity} = \frac{1}{|E^1|} \sum_{(u,v)\in E^1} \frac{d(v)^{(1)}}{d(v)} \tag{5.30}$$

$$E^1 = \{(u,v) \mid u \in S, v \in V_B\} \tag{5.31}$$

where S is the seed set, V_B is the node set of the second part of the graph (modalities), E^1 are the edges originating in one of the seeds, $d(v)$ is the number of neighbors of v and $d(v)^{(1)}$ is the number of neighbors of v belonging to class 1 (seeds). This number will be lower for graphs where seeds are unique to the modalities and higher when multiple seeds share modalities often.

Definition 5.23 (Number of positive edges)**.** Number of positive edges, e.g. leading from seed nodes (SLDs).

$$\texttt{edges\_pos} \tag{5.32}$$

Definition 5.24 (Number of bipartite edges)**.** Number of edges from positive modalities, e.g. modalities connected to at least one positive seed node (SLD).

$$\texttt{edges\_bip\_pos} \tag{5.33}$$

5.4.3 Common Graph Features

Structure Agnostic

Definition 5.25 (Positive to negative class ratio)**.**

$$\texttt{class\_ratio} = \frac{|V_{train}^{(1)}|}{|V \setminus V_{train}^{(1)}|} \tag{5.34}$$

Structure Aware

Definition 5.26 (Class insensitive homophily)**.** One of the well established graph properties is homophily, which expresses how intertwined are the nodes of the same class. There are many variations of homophily such as edge homophily ratio [51], node homophily ratio [16] or class insensitive edge homophily ratio [53]. Because in the retrieval scenario only a small portion of the nodes is of known class, class insensitive homophily $\hat{h}$ is best because of the ability to evaluate each class individually. Definition is as follows:

$$\hat{h} = \frac{1}{C-1} \sum_{k=0}^{C-1} \left(h_k - \frac{|C_k|}{n} \right) \tag{5.35}$$

$$h_k = \frac{\sum_{u \in C_k} d(u)^{(k_u)}}{\sum_{u \in C_k} d(u)} \tag{5.36}$$

where C is the number of classes, h_k is the class homophily, C_k is the set of all nodes belonging to the class k, d_u is the degree of node u which equals the number of neighbours of node u and $d_u^{(k_u)}$ is the number of node u's neighbors that belong to the same class. I am going to compute this only for the seed class

$$\hat{h} = \left(h_k - \frac{|C_k|}{n} \right) \tag{5.37}$$

and refer to this metric shortly as `class_homophily`.

Confusion matrix

The motivation behind establishing an edge-based confusion matrix is that a graph can be considered to be a classificator whose goal is to connect nodes of the same class using an edge. The better the model (graph), the better it is suited for the considered task.

Therefore, I decided to create an edge centric graph features based on the notion of which nodes should be connected and which not. An edge is considered to be a true positive (TP) when it connects two nodes positive for the retrieval task. False positive (FP) edge connects positive and a negative nodes. Negatives are concerned with an edge absence, thus true negative (TN)

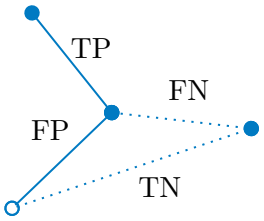


Figure 5.2: Line expresses an edge between two nodes, dotted line stands for absence of an edge. Filled an emtpy nodes are considered to belong into different classes.

is when there's an absence of edge between a positive an a negative node, while false negative (FN) means there isn't an edge between two positive nodes. A visual representation is diplayed in Figure 5.2. Using these concepts I am able to define a confusion matrix and with that all the derived concepts, such as precision or recall, while interpreting them in the new retrieval-oriented graph environment.

The calculation will be the same for both private a public datasets. Positive training nodes (seeds) of the retrieved class are considered to be class 1, all other nodes class 0.

Definition 5.27 (True positive)**.** Number of edges connecting two nodes of class 1.

$$\texttt{cm\_tp} \tag{5.38}$$

Definition 5.28 (True negative)**.** Number of absent edges between node of class 1 and node of class 0.

$$\texttt{cm\_tn} \tag{5.39}$$

Definition 5.29 (False positive)**.** Number of edges between node of class 1 and node of class 0.

$$\texttt{cm\_fp} \tag{5.40}$$

Definition 5.30 (False negative)**.** Number of absent edges between two nodes of class 1.

$$\texttt{cm\_fn} \tag{5.41}$$

Definition 5.31 (Number of edges between negative nodes)**.** Number of edges between two nodes of class 0. This was defined to capture the last combination of nodes.

$$\texttt{edges\_00} \tag{5.42}$$

Definition 5.32 (Precision)**.** Precision expresses how exclusively is the positive class interconnected. It is equal to 1 when there are only edges between nodes of class 1 and no edges to nodes of class 0 from class 1. As there would be no information propagation to the class 0 (unknown) with $FP = 0$, our expectation is for the performance to increase with increasing precision up until some threshold when the information propagation is so low that the

algorithm is not able to find any new positives in the unkown. We found that this definition of edge precision is the same as class homophily h_k from Equation 5.35. Because class homophily is used in the related literature, We are going to use it further in this work instead of the precision. However, we consider this to be a small confirmation that the definition of the confusion matrix is sensible.

$$precision = \frac{TP}{TP + FP} = \frac{\sum_{u \in C_1} d(u)^{(1)}}{\sum_{u \in C_1} d(u)} = h_1 \tag{5.43}$$

Definition 5.33 (Recall)**.** Recall states how interconnected is the class 1.

$$recall = \frac{TP}{TP + FN} \tag{5.44}$$

Definition 5.34 (Prevalence)**.** Prevalence is a concept used in epidemiology where it describes how many specimens are affected by a medical condition from the total population. With the confusion matrix defined on edges, the interpretation is how many edges could lead between the class 1 nodes over total possible edges where at least one node is from class 1.

$$prevalence = \frac{TP + FN}{TP + FP + TN + FN} \tag{5.45}$$

Prevalence feature correlates highly with class ratio defined earlier for both private and public datasets. Possible explanation is that the size of class 1 is very small so the number of possible edges between class 0 nodes and class 1 nodes is much greater than possible edges between nodes of class 1. Therefore, prevalence cannot be used in unbalanced dataset which we unfortunately deal with.

Definition 5.35 (False positive ratio)**.** False positive rate (FPR) is a ratio between how many edges between class 1 and class 0 nodes exist out of all the possible combinations. In other words, it gives an approximate notion of how well the information can be propagated from the class 1 nodes to the class 0. It is also a complement of specificity, also called true negative rate (TNR).

$$\texttt{fpr} = \frac{FP}{FP + TN} = 1 - TNR \tag{5.46}$$

Definition 5.36 (Negative predictive value)**.** Negative predictive value (NPV):

$$\texttt{npv} = \frac{TN}{TN + FN} \tag{5.47}$$

5.5 Datapoint Construction

The construction of datapoints for the regression meta-model goes as follows. Graph features are considered to be the features or variables for the method.

Measured performance is the ground truth, the true label. These datapoints are divided into training and testing sets based on the experiment's goal. The model is trained using the training samples and evaluated using the test samples which it has never seen before.

For the public datasets, hyperparameters from Table 5.4 are also used as variables. The goal is to capture the hyperparameter as another variable, as some graphs might perform better with different parameters than others based on their structure. There are total of 108 hyperparameter configurations and total of 139 binary classification tasks for the public datasets. This results in 15 012 datapoints for public datasets in total.

For network telemetry there are just 14 days of data for 4 bipartite graph each day, which sums up to 56 datapoints.

Chapter 6

Experimental Evaluation

In this chapter, experimental evaluation is provided to validate the effectiveness of proposed approach. Proposed method is training a random forest regression meta-model (described in Section 4.2.2) on a vector of graph feature variables introduced in Section 5.4 and the true performance of a graph algorithm as the ground truth. The goal of the regression meta-model is to predict the performance, based on which the importance of individual features will be evaluated using the SHAP method described in Section 4.3. The method illustration is in Figure 6.1.

The experiments are designed to cover different aspects of the problem at hand and will be divided into two sections. The first one will focus on the Cisco use-case of applying Risk Map Graph algorithm (Section 5.3.2) to network telemetry (Section 5.2.3). Second one will be focusing on the generalized problem of applying GNNs (Section 5.3.1) to retrieval task on public graph datasets (Section 5.2.1).

In each section, we start by presenting a correlation analysis of the proposed graph features to the performance metric. We will be using the Spearman correlation, which does not evaluate the linear dependacy of the two variables but rather the monocity of the resuting function by ranking the observations. Spearman correlation coefficient will be refered to as ρ.

Information about datapoints can be found in Section 5.5.

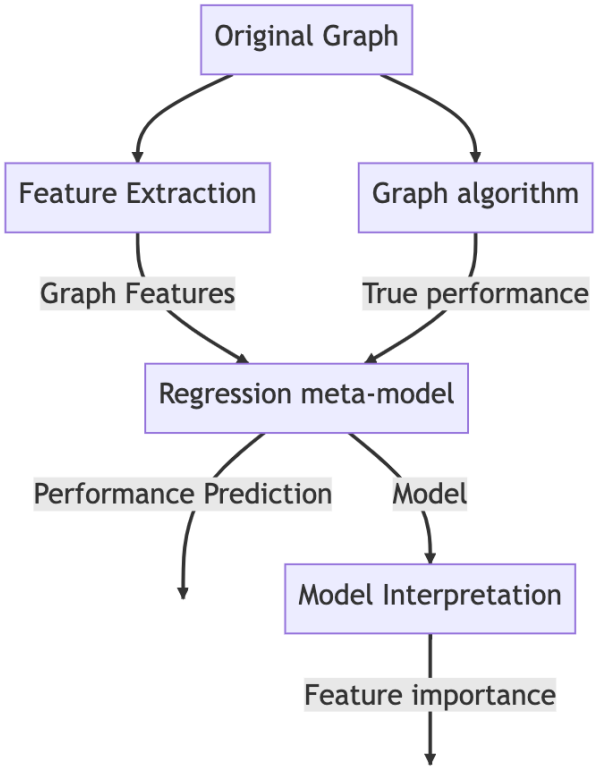


Figure 6.1: Thesis approach illustration. Graph features of the original graph and task performance dependant on the algorithm are used to train the regression meta-model. Meta-model provides the feature importance which can be used to optimize the original graph to maximize task performance. Meta-model's performance prediction is not the main goal and serves only as a validation to verify the model reflects graph structure and task and doesn't just over-fit on the training data. (Repeated Figure 1.1.)

6.1 Network Telemetry IoC Retrieval

The task of IoC retrieval is going to be performed using the Risk Map algorithm (Section 5.3.2) on 14 days of network telemetry data, described in Section 5.2.3. Each modality was evaluated separately on each day, which means there are total of 56 datapoints, each corresponding to a single graph. Not all of the graph features could be computed due to the large size of the data, therefore only the following will be considered:

- Number of edges between negative nodes (`edge_00`)
- Number of bipartite edges (`edges_bip_pos`)
- Class homophily (`class_homophily`)
- Class ratio (`class_ratio`)
- Clique number (`clique`)

- Expansion ratio (`expansion`)
- Number of true positive edges (`cm_tp`)
- Number of true negative edges (`cm_tn`)
- Number of false positive edges (`cm_fp`)
- Number of false negative edges (`cm_fn`)
- False positive ratio (`fpr`)
- Number of modality nodes (`nodes_mod`)
- Modality positivity (`mod_positivity`)
- Number of negative domains (`nodes_negs`)
- Number of nodes (`nodes`)
- Negative predictive value (`npv`)
- Number of positive domains (`nodes_poss`)
- Precision (`precision`)
- Prevalence (`prevalence`)
- Recall (`recall`)
- Number of seed edges (`edges_pos`)

6.1.1 Metric Correlation

As the initial goal was to find a proxy-metric which could be used to capture the performance of the task on the graph, all of the graph features were plotted with the number of IoCs performance metric in Figure A.1. Unfortunately, none of them correlate with the performance enough across the modalities to consider them to capture the performance alone. Combination of graph features might still be able to approximate the performance at least to some degree, which is why I will be training the meta-model regressor.

The RF regressor should not be given multiple highly correlated variables. In order to statisfy this assumption, pairwise correlation for graph features is visualized in Figure 6.2. In the left plot, regressor's prediction for training set are presented, while the right plot displays results for the test set. There are four highly correlated groups of features:

- Class homophily & precision
- Number of nodes & number of negative domains
- Class ratio & prevalence

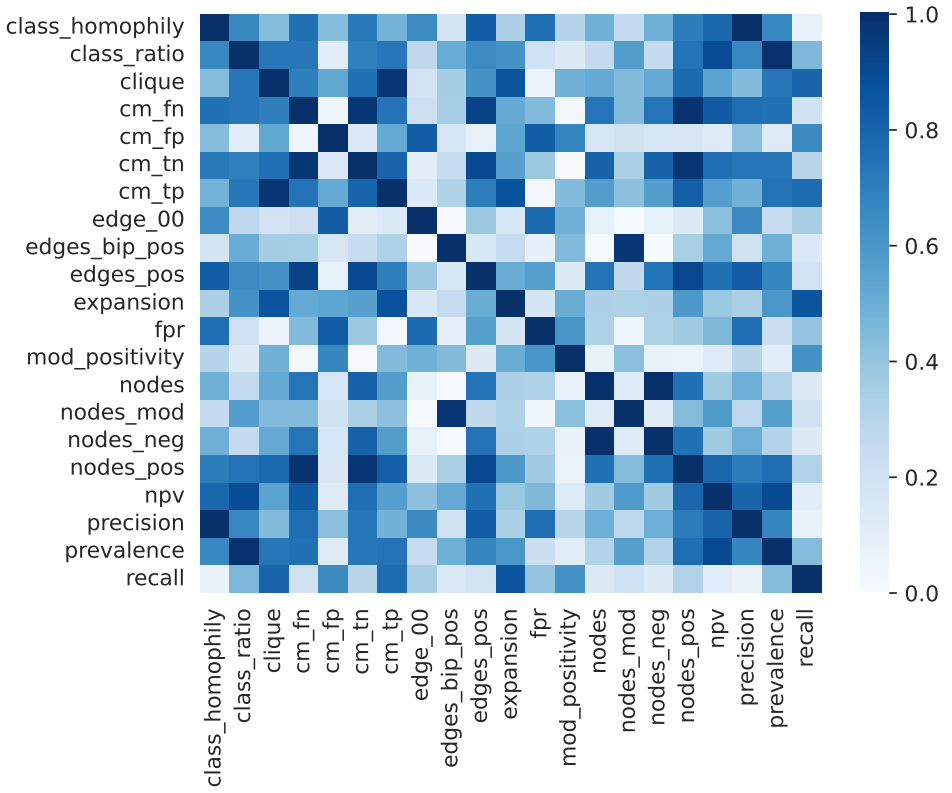


Figure 6.2: Correlation of graph features used to describe the network traffic graphs.

- TN & FN & number of positive domains

I am going to keep the first graph feature from each group. The decision was made according to interpretability of each metric with the most interpretable kept.

6.1.2 Cross Modality Validation

To study the generalization ability across the modalities, I employed a leave-one-out strategy on the modalities, meaning the training of the RF regressor considered three out of four modalities and the one left out was considered for testing. Results can be seen in Figure 6.3. In all cases but one, both the Spearman correlation coefficients and the root mean squared error (RMSE) indicate poor results. Only for the server IP as testing modality, Figure 6.3d, the correlation coefficient is $\rho = 0.83$ but that alone does not prove good results as RMSE is still high at $RMSE = 114.63$.

These results suggest that each of the modalities is quite different from the rest and there probably is no single indicator between the proposed graph metrics. However, the fact that the regressor seems to be able to learn well

among the modalities presented during the training indicates there cound be a systematic way of combining multiple variables (graph features) in a non-linear way. This corresponds with the conclusion drawn from the correlations alone.

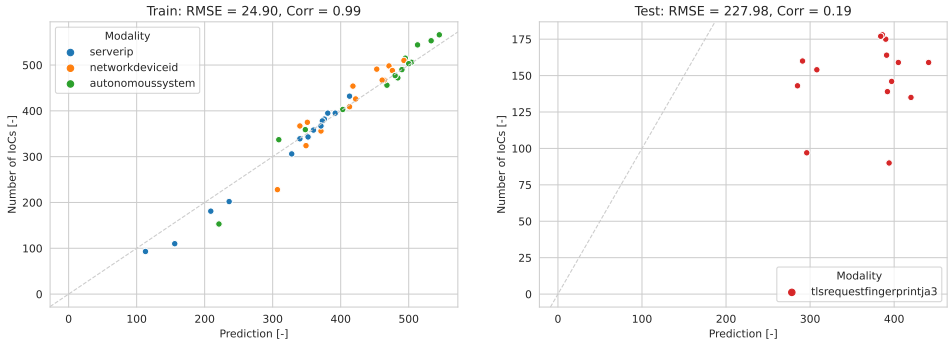
6.1.3 Generalization in Time

To uncover the non-linear combination of graph metrics, the classificatior first has to provide reasonable predictions for the test examples. As the network telemetries are gathered across 14 consecutive days, next experiment is aiming to generalize from first 9 days onto the last 5. The results are shown in Figure 6.4. Most of test datapoint's predictions are close to the ground truth except for the three ones in lower right corner of the plot. The maximum absolute error is 250.03. However, without these 3 points, which account for 12% of the test set, the maximum error decreases to 95.99, $RMSE = 43.22$ and correlation increases to $\rho = 0.95$.

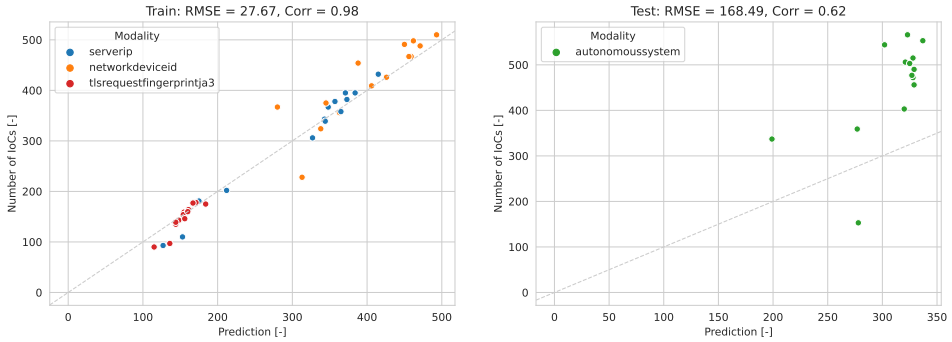
I tried looking into the reasons these 3 datapoints get classified incorrectly as they even come from different modalities. One common property they share is that they are a weekend datapoints, which contain generally about half of the connections of a normal weekday. However, there are total of 8 weekend datapoints meaning the other 5 get relatively good predictions. Despite this, the model might still provide some valuable insight into the general behavior of the graph metrics and further analysis could reveal them.

Figure 6.5 presents a beeswarm plot of SHAP feature importance. The features are ordered from the most important to the least important. X-axis displays the SHAP values which represent the impact on the prediction. Color scales from highest feature value in red to the lowest in blue. Notice we are able to interpret which values (higher or lower) increase or decrease the prediction based on the dot's color. When the colors are mixed across the x-axis, we can assume the feature is a pair-wise (or group-wise) feature meaning the influence is dependant on other (one or more) features. This is thanks to the expressive ability of the RF method, as it can build decision trees and consider variable values linked to anothers.

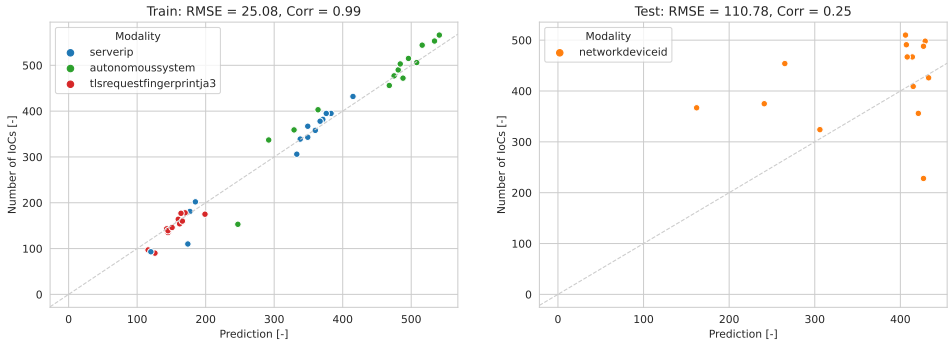
Such group-wise features appear to be the most important number of true positive edges. On the other hand, clique number and class ratio both benefit the model when their values are high. Higher class ratio means higher number of initial positive seeds, which intuitively enables discovery of more nodes than when the ratio is lower. Lower number of modality nodes appears to increase the prediction. We can interpret this that graphs with lower number of modality entities probably dilute the information less and algorithm is therefore able to discover more positive nodes. Well divided are the red and blue points for number of true negative edges. True negative edge stands for an absence of edge between different classes, which could propagate the information in wrong direction, therefore less of these edges (higher true negative count) increases the performance.



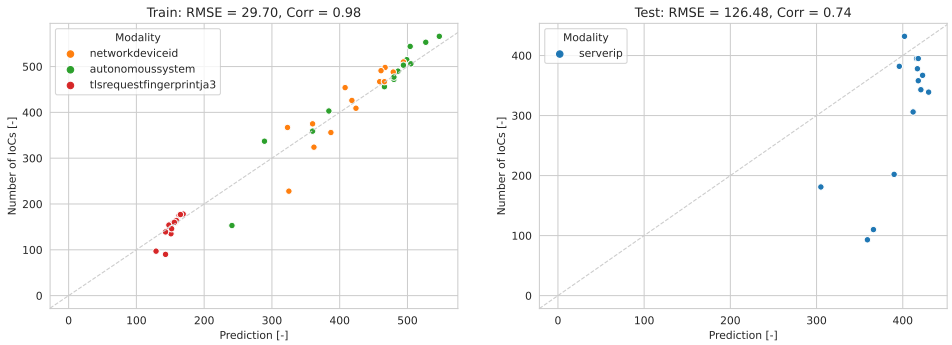
(a) : Testing on TLS fingerprint, training on the rest.



(b) : Testing on autonomous system, training on the rest.



(c) : Testing on network device, training on the rest.



(d) : Testing on server IP, training on the rest.

Figure 6.3: Meta-model predictions on network telemetry. Results of generalization experiments from three modalities to the missing one.

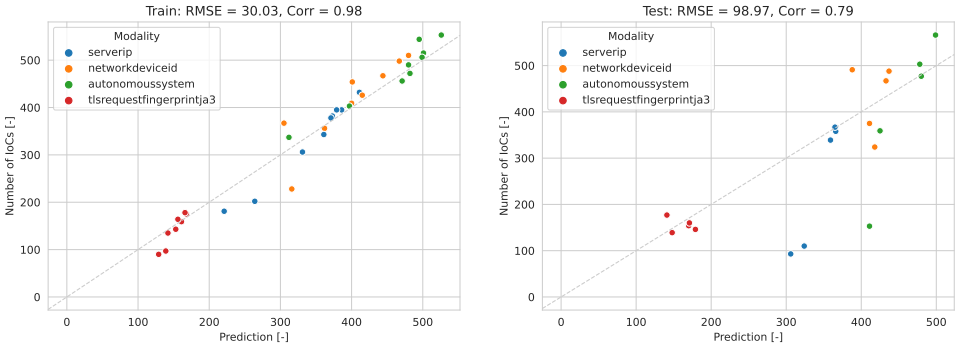


Figure 6.4: Meta-model predictions on network telemetry. Results of generalization through time, training examples were taken from the first 9 days, testing examples from the remaining 5.

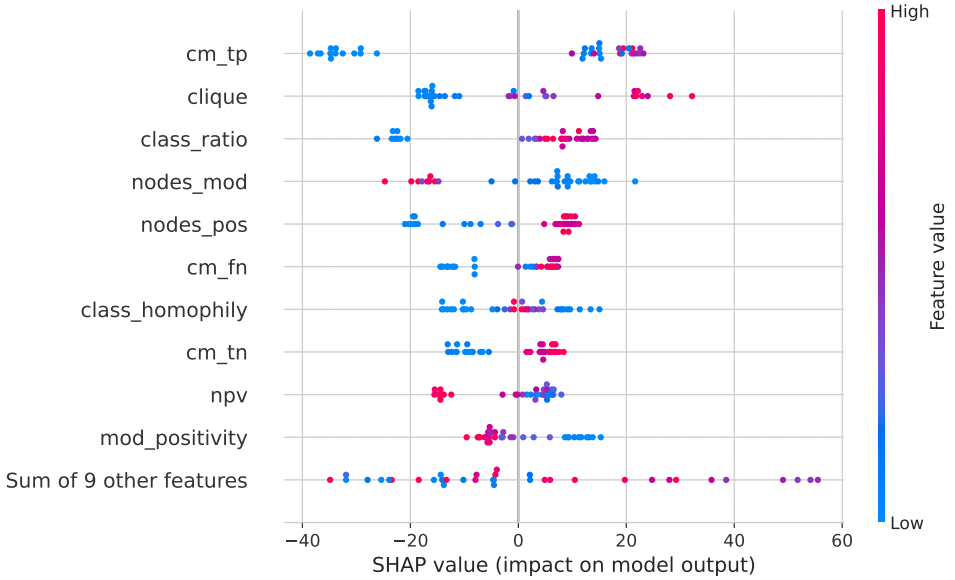


Figure 6.5: Summarization of SHAP values in the training examples. Features are ordered with the most influential on the top. Interpretation for time generalization scenario, predictor displayed in Figure 6.4. Magnitue bar plot available in Figure A.6.

6.2 Node Retrieval on Public Datasets

For node retrieval on public datasets, total of seven graph datasets with various number of nodes, number of edges and different type of source data were used. As the considered task is node retrieval, the nodes have to be redistributed into a positive and a negative class. Considering original classes $C = \{1, 2, ..., c\}$ and a selected target positive class t, the nodes are re-labeled so that nodes of class t in the original graph are positive (1) and all other classes $C \setminus t$ are negative (0). That means for one dataset with c classes there are c binary classification tasks. For more details about the public datasets, please refer to Section 5.2.1. For more detail about the datapoints, see Section 5.5.

As mentioned in 5.3.1, hyperparameter search is alone a difficult field. Therefore, I decided to experiment with different hyperparameters in a defined design space. Those will also be included as variables for the datapoints in addition to the graph features. This will also make the datapoints more representative of the graph compared to when only one set of hyperparameters would be used for all of them.

The performance of the task will be evaulated using the precision at k metric with $k = 100$. I am also going to look into the F1 score which will have better representation of the overall performance and will not be limited to the top 100 nodes for the positive class.

The following graph features have been used:

- Assortativity (`assortativity`)
- Class homophily (`class_homophily`)
- Positive to negative class ratio (`class_ratio`)
- Conductance (`conductance`)
- Neighborhood feature vector cosine similarity (`cos_similarity`)
- Average degree (`degree_avg`)
- Average degree of positive nodes (`degree_pos_avg`)
- Number of edges between negative nodes (`edge_00`)
- Number of edges (`edges`)
- MSE eucledian distance of class X feature vector to mean class Y feature vector (`feat_dist_XY`)
- Number of true positive edges (`cm_tp`)
- Number of true negative edges (`cm_tn`)

- Number of false positive edges (`cm_fp`)
- Number of false negative edges (`cm_fn`)
- False prediction ratio (`fpr`)
- Attribute homophily (`homophily_attr`)
- Edge homophily (`homophily_edge`)
- Node homophily (`homophily_node`)
- Number of nodes (`nodes`)
- Negative predictive value (`npv`)
- Number of connected components (`num_components`)
- Number of negative trainig nodes (`num_negative`)
- Number of positive training nodes (`num_positive`)
- Ratio of positive nodes among nodes of degree > 1 (`pos_node_ratio_1`)
- Ratio of positive nodes among nodes of degree > 2 (`pos_node_ratio_2`)
- Ratio of positive nodes among nodes of degree > 3 (`pos_node_ratio_3`)
- Precision (`precision`)
- Prevalence (`prevalence`)
- Recall (`recall`)

6.2.1 Graph Feature Correlation

Figure 6.6 displays pair-wise correlation between all of the graph features. For graph features correlated with $\rho = 1$ only one feature from the group will be kept. There are some other highly correlated groups, which provide complementary information, such as `feat_dist_AB`. For this reason, they have been kept despite the correlation.

Groups correlated with $\rho = 1$, the first graph feature in the group was kept:

- Class ratio & NPV & prevalence
- Number of positive nodes & FN

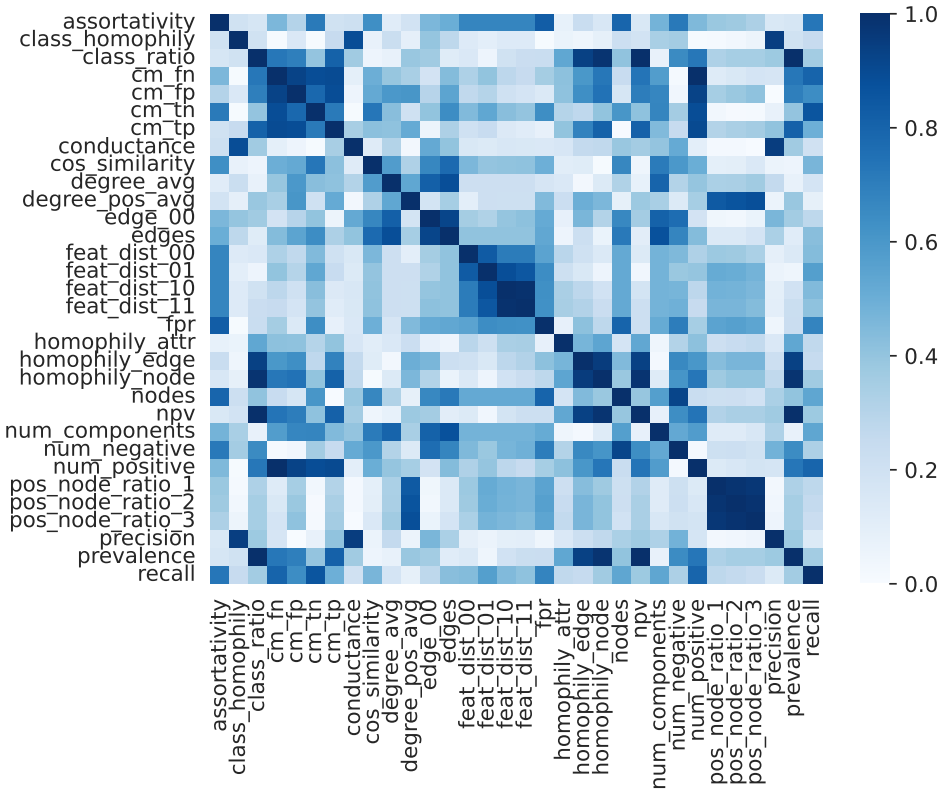


Figure 6.6: Absolute correlation of graph metrics between each other.

6.2.2 Plausability Validation

The fist two experiments were performed as a means of validation to ensure the consistency and plausibility of the findings. In Figure 6.7, results of training on the graph features alone are shown. In Figure 6.8, conversely, only hyperparameters were used to distinguish the observations. The training and testing examples were randomly divided with two-thirds allocated for training and the remaining one-third for testing.

In the hyperparameter only scenario, the predictions align with the ideal trendline a little and correlation $\rho = 0.79$ is achieved for training, $\rho = 0.80$ for testing. This can be explained by the fact that number of layers influence the performance (in this case precision at 100) greatly and these two are therefore highly correlated as shown in Figure 6.6. Despite that, the model still predicts the same value for multiple observations. Predictions are even more off in the case of graph feature only scenario. The model in incapable of learning on the presented data.

All of that is expected, as there are insufficient information to distinguish the datapoints from each other: the graph features are shared between

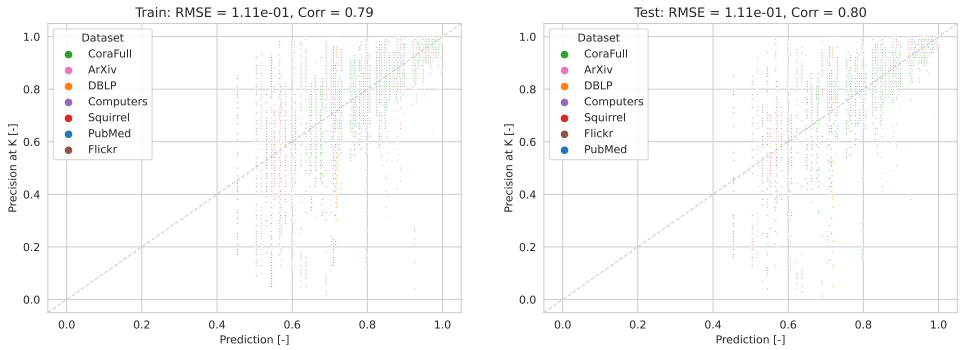


Figure 6.7: Meta-model predictions on public datasets. Only graph feautres were used. Random split with $\frac{2}{3}$ used for training, $\frac{1}{3}$ for testing.

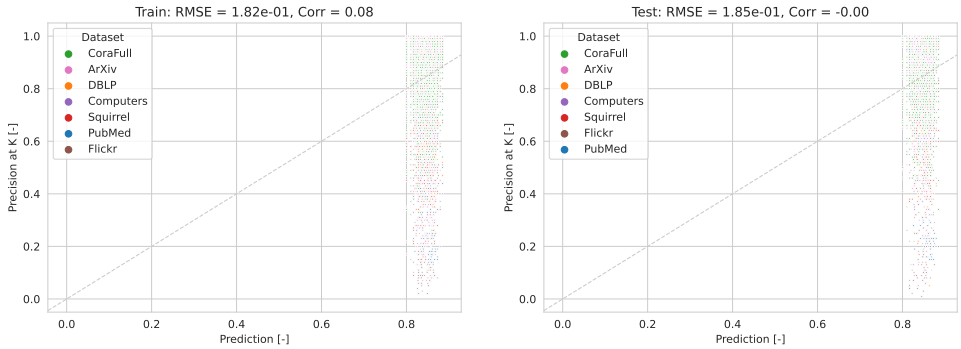


Figure 6.8: Meta-model predictions on public datasets. Only GNN hyperparameters were used. Random split with $\frac{2}{3}$ used for training, $\frac{1}{3}$ for testing.

datapoints with different parameters[1] and vice versa[2]. As a result, the RF regressor merely returs the expected value of all observations sharing the same variable values.

6.2.3 Random Split

Let's now apply the meta-model to all available variables (both the hyperparameters and graph features). With two-thirds of total number of datapoints used for training and the remaining one-third for testing, predictions of the RF meta-model regressor are displayed in Figure 6.9. Correlation with precision at k for the test predictions is above $\rho = 0.9$ and visually the predictions generally follow the trendline.

Feature importance inspection using SHAP is shown in Figure 6.10. The first positive outcome is that the most important features are not just the hyperparameters in combination with one or two graph features as that

[1]The same graph features are shared between 108 points as the are only 139 different graph feature value sets.

[2]The same hyperparameters are shared between 139 points as the are only 108 different hyperparameter cofigurations

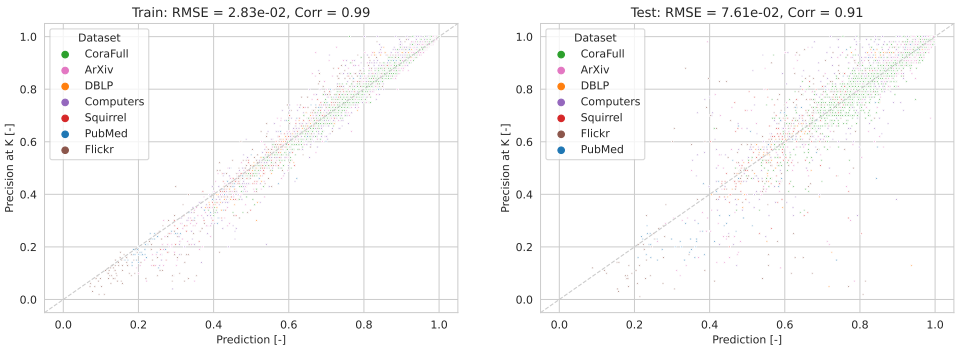


Figure 6.9: Meta-model predictions on public datasets. Both hyperparameters and graph features were used. Random split with $\frac{2}{3}$ used for training, $\frac{1}{3}$ for testing.

would be enough to distinguish all of the observations. Instead, only 3 hyperparameters (number of layers, aggregation function and number of hidden channels) are present in the top 10 features and only one of them in the top 5.

There are only two graph features I would consider to provide a good split based on the feature value. The first one is number of edges between negative nodes, where higher value results in higher prediction. Second one is the ratio of positive nodes among nodes of degree 3 where higher value decreases the prediction. That means it is more important for the performance to have less neighbors to propagate the information than more, which corresponds with the precision at k metric which wants the results to be as confident as possible.

The most important features of class ratio and node homophily seem to be pair variables as there is a mixture of high and low feature values that decrease the prediction. For class ratio, low value increases the prediction more often than not, for node homophily the same is true for high feature values. Third most important feature, the number of negative nodes, almost exclusively increases the prediction for high values.

The hyperparameter variables are very noisy and must be dependant upon other variables. I would expect for example higher number of layers clearly correlate with higher predictions. However, this can be caused by the used performance metric precision at 100, which (especially for small windows k) prefers confident predictions over the total number. Higher model expressability is able to propagate the information through the graph over longer distances, which can sometimes be a drawback.

6.2.4 Model Generalization Across Datasets

In this study, a random forest regressor was employed to predict precision at 100, taking into account both hyperparameters and graph features. The

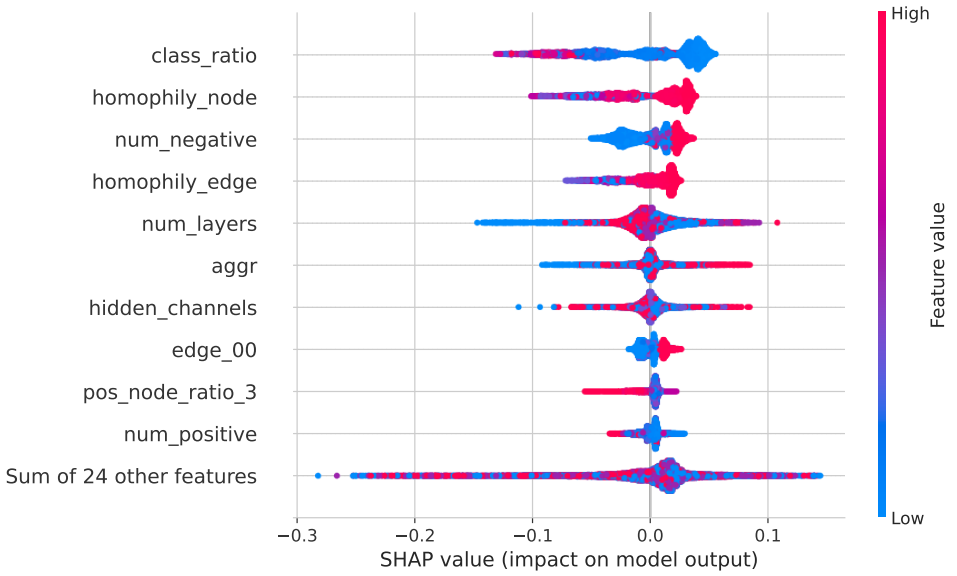


Figure 6.10: SHAP feature importance analysis of model from 6.9, random split on public datasets with prec@k as the performance metric. Magnitue bar plot available in Figure A.7.

primary focus of this investigation was to explore the potential generalization of the meta-model across datasets, wherein the model was initially trained on a selection of datasets and subsequently applied to a novel dataset without any further training.

Unfortunately, the results obtained from this approach (shown in Figure 6.11) were unsatisfactory. Even when each testing dataset is explored separately, there's no noticable trendline, the meta-model predicts all the eamples into some range different for each dataset. For Pubmed the approximate range is 0.7 to 0.9 and the around 0.98. Squirrel predictions are mostly between 0.6 and 0.8. A potential explanation for this outcome could be attributed to the significant differences between the datasets used for training and the novel dataset. This disparity may have hindered the model's ability to generalize and accurately predict precision at 100, thus emphasizing the need for further research and optimization in the realm of cross dataset generalization.

To explore this more in depth, I also conducted an experiment utilizing only the citation network datasets PubMed, DBLP, Cora and ArXiv. All combinations were performing approximately the same with part of the testing samples indicating the trendline but others completely wrong. Example is presented in Figure 6.12 with DBLP dataset used for testing.

To conclude this experiment, meta-model generalization does not seem to be viable even for datasets based on similar types of data.

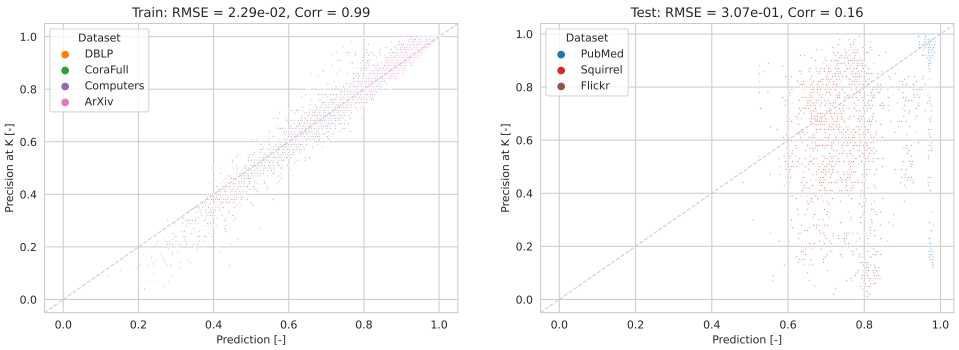


Figure 6.11: Meta-model predictions on public datasets. Both hyperparameters and graph features were used. Datasets DBLP, CoraFull, Computers and ArXiv were used for the training, PubMed, Squirrel and Flickr for the testing.

Figure 6.12: Meta-model predictions on public datasets. Both hyperparameters and graph features were used. Datasets PubMed, CoraFull and ArXiv were used for the training, DBLP for the testing. All of these datasets are citation network data.

6.2.5 Hyperparameter Search

In the subsequent experiment, the focus shifted to conducting a hyperparameter search on a specific dataset. Consider a situation when only a handful of GNN runs with different hyperparameters in combination with the graph features could train a good enough meta-model predicting the performance for the rest of the hyperparameters. If that is the case, we could effectively limit the hyperparameter search to just those runs used to train the meta-model.

Our experiments indicate that this indeed could be true. See Figure 6.13, where the meta-model regressor is trained for the Flickr dataset on just 38 (5% out of the total 756) observations. The predictions for the rest of the observations may seem fairly scattered upon first glance. However, for this scenario we are only interested in configurations that perfom the best. Suppose we are interested in node retrieval of arbitrary class t. After specifying a GNN design space, running approximately those 5% of total configurations with different target classes, we will be able to get very close to the optimal GNN configuration for our target class t. Note, that this could not be done by

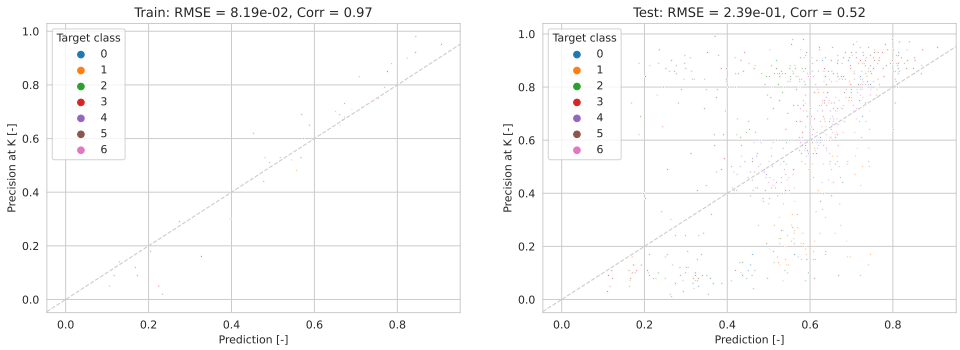


Figure 6.13: Meta-model predictions on public datasets. Both hyperparameters and graph features were used. Thirty randomly selected observations from ArXiv dataset were used for training, the rest for testing.

	Train		Test		Optimal	Deviation	
Class	prec@k	Prediction	prec@k	Prediction	prec@k	Train	Test
0	0.90	0.82	0.97	0.82	0.98	0.08	**0.01**
1	0.88	0.79	0.81	0.77	0.92	**0.04**	0.11
2	0.65	0.59	0.86	0.62	0.92	0.27	**0.06**
3	0.95	0.91	0.95	0.91	0.99	**0.04**	**0.04**
4	0.73	0.67	0.28	0.67	0.78	**0.05**	0.50
5	0.98	0.84	0.88	0.87	0.98	**0.00**	0.10
6	0.79	0.77	0.77	0.76	0.86	**0.07**	0.09

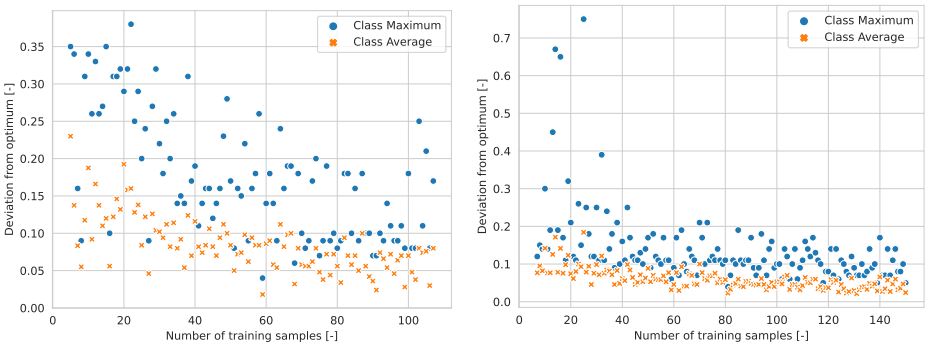
Table 6.1: Overview of predictions and prec@k performance for each class in the Flickr dataset corresponding to results in Figure 6.13. Lowest deviation from the optimal performance for each class is in bold.

training on the hyperparameters alone as the meta-model would not be able to distinguish two target classes with the same hyperparameter configuration.

Table 6.1 pesents the meta-model performance predictions and true performances measured by precision at 100 for each class for the training presented in Figure 6.13. The approach is following. In the training set, we can choose the best configuration according to the real performance (prec@k) as we have computed those samples. In test set, we choose the best predicted configuration for each class and evaluate it. Best performing configuration from the two chosen is then compared to the optimum.

Compared to the best performing datapoints, the worst performance is found for class 6 retrieval where the difference is 7% compared to the best performing configuration. For classes 0 and 5 we are able to obtain best possible hyperparameters for class 5 and almost best, losing 1% to the optimal configuration, for class 0.

In Figure 6.14, the training sample size is further explored for Flickr and Squirrel datasets. Both the maximum class deviation and average deviation from optimum decrease with growing sample size, however, the values stabilize



(a) : Squirrel dataset **(b) :** Flickr dataset

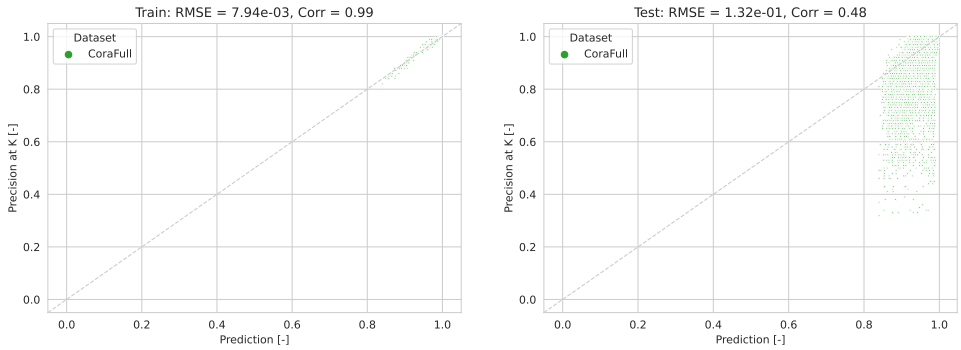
Figure 6.14: Dependance of deviation from optimum on training sample size for two datasets. Maximum and average values from all binary classification tasks are shown for each training sample size. Training sample sizes were chosen to be from range between number of classes to 20% of observations. Note that both the x and y axes have different ranges for each plot.

fairly quickly and little to no change is visible beyond the 10% mark.

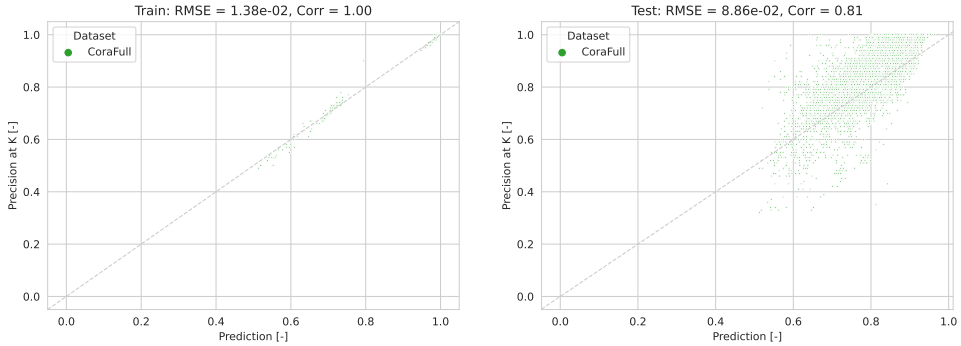
It is important to state, that when the training observations exhibited low variance, the predictions on the test set deviated considerably from the true values and the results were not as promising. Illustrative examples are displayed in Figure 6.15. This outcome is not unexpected, as it is common phenomenon in ML methods when dealing with datasets characterized by minimal variation. The model's low performance in such cases highlights the importance of having diverse and representative training data to ensure accurate predictions. In the real-worl usage however, we would be able to detect this case and simply add more training samples to cover different hyperparameter configurations and various performance.

Despite these challenges, the results appear promising, as the optimal hyperparameters could potentially be predicted by integrating structural information. Consequently, this would enable the execution of only a few trials, as opposed to conducting an extensive grid search.

Upon inspection of the SHAP analysis in Figure 6.16, three of the hyperparameter variables appear on top of the importance list and even provide a fairly good segregation based on their values. Aggregation mean (1) performs better than maximum (0) and ReLU activation (1) better than PReLU (0). Higher number of layers increase the prediction as expected. Suprisingly, higher number of hidden channels decrease the prediction and further research is needed to explain this. Unforutnately, none of the top 10 features are the same as the ones from random split experiment presented in Figure 6.10.



(a) : Training examples consisted from observations of classes 1 and 2, which are very similar. Despite the 216 training samples, model fails to learn properly.



(b) : Training examples consisted from observations of classes 1 and 13, which are diverse. Despite the same number of training samples as in the figure above, model outputs reasonable predictions for the other classes.

Figure 6.15: Comparison of two meta-models trained on Cora dataset with different training examples.

6.2.6 F1 score

As mentioned above, precision at k metric, although important for the retrieval task, might be making the feature importance harder to interpret, especially for $k = 100$. To provide an alternative view, I am going to investigate alternative performance metric F1 score, which is also suitable for imbalanced datasets. F1 score is calculated as the harmonic mean of precision and recall:

$$F_1 = 2\frac{precision \cdot recall}{precision + recall} \tag{6.1}$$

This is ideal for the meta-model training, as it needs to be trained on one metric as ground truth and neither precision nor recall could express the whole information, while F1 combines both of them.

The random split training and testing results can be seen in Figure 6.17. Samples in the training set are more uniformly distributed along the first quadrant axis than in the case of precision at k, which seems to benefit the training, reaching correlation $\rho = 0.93$ for testing set.

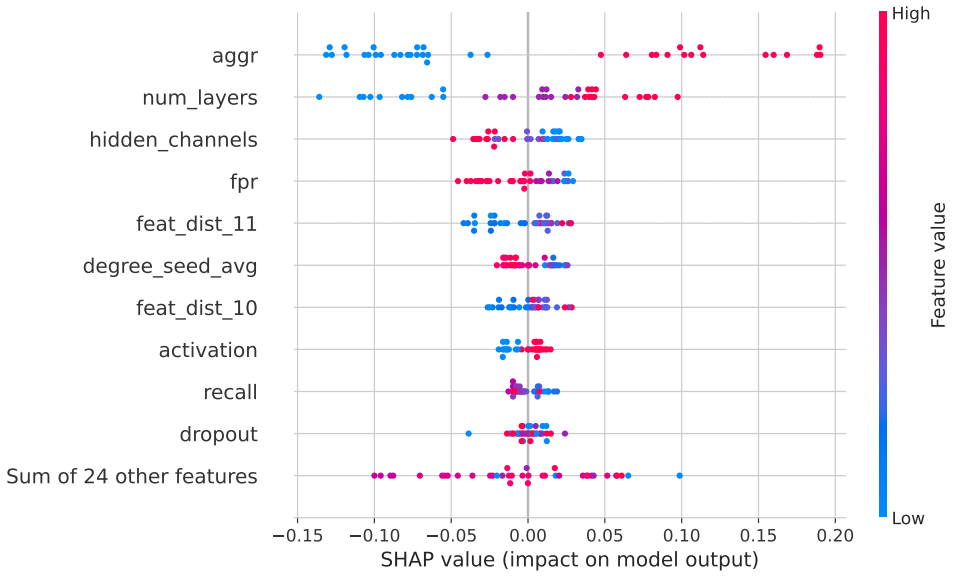


Figure 6.16: SHAP feature importance analysis of model from 6.13, hyperparameter search. Magnitue bar plot available in Figure A.8.

Top 10 important features for F1 score, shown in Figure 6.18, include many of the features seen in top 10 for precision at k in Figure 6.10. However, each one of them has exactly the opposite influence on the final prediction, this time class ratio increases prediction for higher values, number of edges between negative nodes for low values and number of negative nodes also increases the prediction for lower values. This is unexpected and further research will be needed to fully understand this outcome.

The hyperparameter features, on the other hand, are very well segregated this time compared to the prec@k case, as F1 score is more capable to capture the learning on all of the nodes, not only the k nodes included in the window. Both hidden channels and number of layers raise the prediction for higher values as expected. However, aggregation and activation importance is swapped compared to hyperparameter feature importance scenario (Figure 6.16).

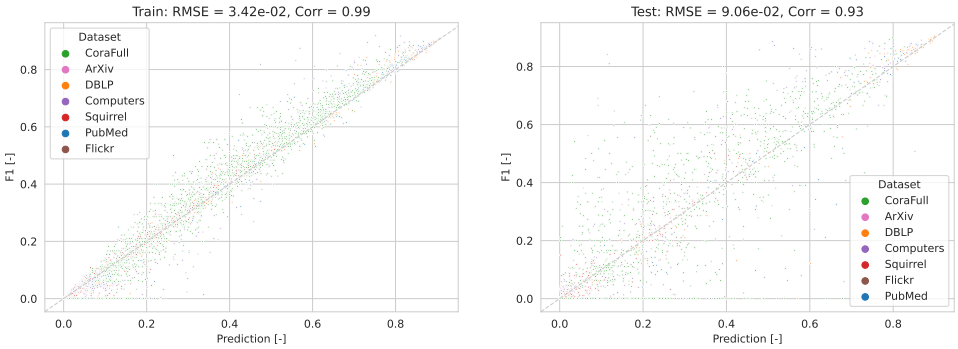


Figure 6.17: Meta-model predictions on public datasets. F1 score was used as the ground truth performance metric. Random split with $\frac{2}{3}$ used for training, $\frac{1}{3}$ for testing.

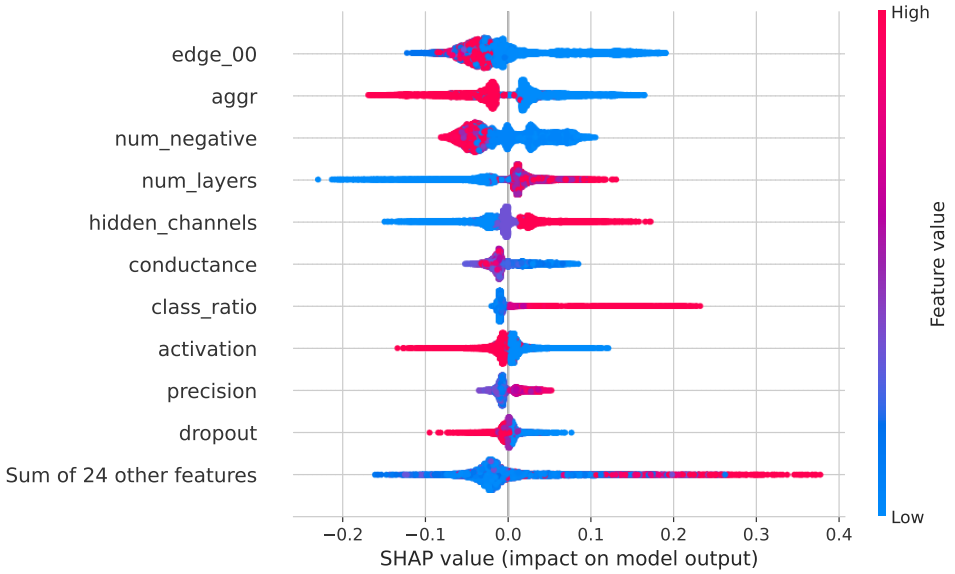


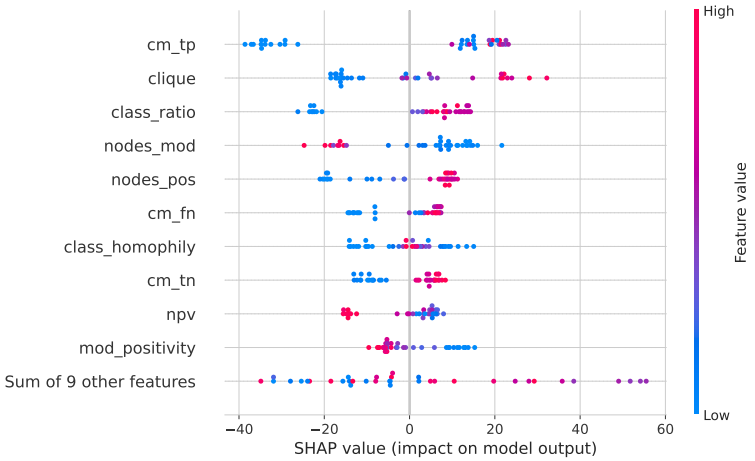
Figure 6.18: SHAP feature importance analysis of model from 6.17, random split on public datasets with F1 score as the performance metric. Magnitue bar plot available in Figure A.9.

6.3 Experiment's Conclusion

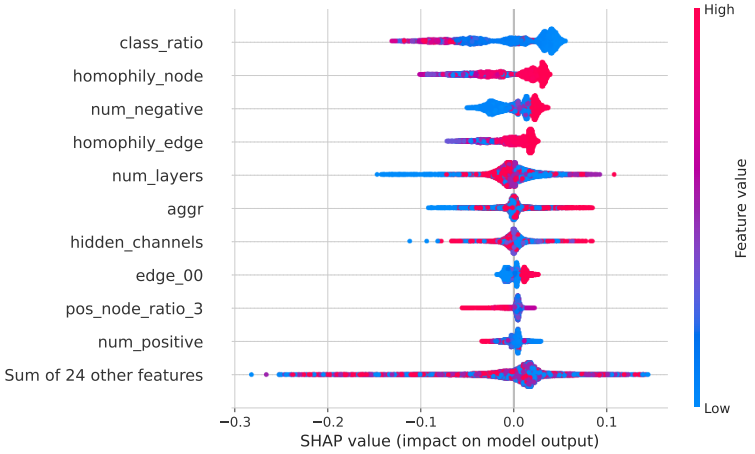
In conclusion, there exist graph features which appear repeatedly among the top 10 important, such as `edge_00`, `class_ratio` or `num_negative`. However, they may influence the predicted value differently based on the task or the performance metric, so there do not seem to be any universal graph features. A comparison of three main SHAP beeswarm plots is presented in Figure 6.19.

The differences in influence might have been caused by the use of improper graph feartures. For example, mentioned `edge_00` feature was imprudently defined as an absolute number instead of normalizing the value by the total number of edges. Similar issues arise for multiple graph features defined in Section 5.4 and have to be fixed during continued research effort.

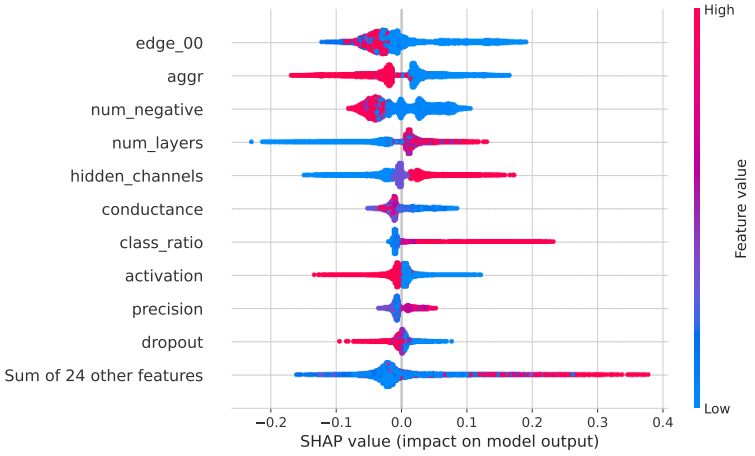
Promising results were shown in the hyperparameter search scenario. Further research is still needed to fully describe the scope of application and provide theoretical boundaries for the results. Nevertheless, the idea of effective hyperparameter search for graph algorithms would benefit the research community not to mention the possible applications and possible time savings.



(a) : SHAP beeswarm for Risk Map time generalization scenario, model from Figure 6.4. (Repeated)



(b) : SHAP beeswarm for GNN random split with prec@100, model from Figure 6.9. (Repeated)



(c) : SHAP beeswarm for GNN random split with F1 score, model from Figure 6.17. (Repeated)

Figure 6.19: Comparison of SHAP feature importance for different setups.

Chapter 7

Conclusion

7.1 Discussion

The work provides a comprehensive review of related literature on graph structure and feature engineering, as well as state-of-the-art machine learning algorithms on graphs and their applications in cybersecurity.

A novel approach for evaluating the importance of graph structural features using a regression meta-model on datapoints consisting of graph features and task performance is proposed. The approach explores a large number of structural and structure-agnostic graph features, including a novel definition of an edge-based confusion matrix for retrieval tasks on a graph.

Multiple experimental scenarios were examined such as generalization across datasets or temporal generalization on the network telemetry datasets. Experimental results indicate that important graph features vary across datasets and performance metrics.

Additionally, the proposed approach shows promise in accelerating hyper-parameter search. Despite strong assumptions and a narrow application area for the time being, continued research in this area has the potential to yield significant advancements in graph machine learning.

Overall, this work contributes to the field of graph machine learning by providing a more comprehensive and explainable approach to evaluating the importance of graph features.

7.2 Limitations & Future Work

While providing valuable insights, the research presented in this thesis is not without its limitations and areas for improvement.

There are some absolute number graph features, such as the number of edges between negative class, which should have been defined in a more "graph

size resistant" way. A way of normalizing these graph features could prevent distinguishing the datapoints based on graph size and help generalization across datasets.

We also realized, that one way to provide more insight into the feature importance analysis might have been to select the most important features in one of the experiment scenario and apply only this subset of features into the meta-model for another experiment. This would also result in a more comparable output.

The hyperparameter search, which was an unexpected side effect of our approach, requires a thorough theoretical investigation. Furthermore, the average deviation from the optimum for each class across configurations should have been provided to give context to the presented results.

To fully verify the main Hypothesis 1.1, further experiments and more data need to be evaulated in the future.

Finally, the approach using the regression meta-model is clearly limited by how well the graph features describe the input graph. All the features provide aggregated information about the graph. In the future, a more sophisticated approach could be designed to first divide the graph into related subgraphs based on a criterion and compute the graph features for each of them separately. This way, the expressive power of the aggregation features would increase dramatically and better results could be obtained.

All of these limitations can be considered a future research direction for this topic. To enhance the effectiveness of our approach, future work should focus on normalizing graph features to ensure better generalization across datasets, refining feature importance analysis for more comparable results, and conducting a thorough investigation of the hyperparameter search application.

By addressing these limitations and exploring the proposed future research directions, we believe that significant progress can be made in the understanding and application of graph features for relation selection on graphs.

7.3 Conclusion

This section summarizes how different parts of the thesis assignment were accomplished.

A comprehensive survey of the related literature and exploration of possible directions has been provided in Chapter 2.

A formal description of the problem at hand was formulated in Section 5.1.

Method for suitable relation types selection was proposed using the regression meta-model operating on graph structural features and an evaluated set of samples providing the true performance used as a meta-model label. An explanation of how the proposed graph features relate to graph performance

was given in Section 5.4. Section 5.1 describes the relation selection process once the graph feature importance is established. This approach could be applied to an arbitrary graph algorithm but requires a set of evaluated data.

Experiments were carried out in Chapter 6 in order to find the graph features to use for the relation selection. However, performed experiments suggest that the graph feature importance is not universal across graph algorithms and performance metrics. Separate sets of graph features important for the performance of the custom Cisco algorithm and GNN on public datasets were found. The final relation type selection has not been performed because of the complexity of the previous parts of the assignment. Still, it is planned to be concluded in an oncoming paper based on this thesis.

Bibliography

[1] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2018/file/53f0d7c537d99b3824f0f99d62ea2428-Paper.pdf

[2] T. Nguyen, H. Le, T. P. Quinn, T. Nguyen, T. D. Le, and S. Venkatesh, "GraphDTA: Predicting drug-target binding affinity with graph neural networks," *Bioinformatics*, vol. 37, no. 8, pp. 1140–1147, 10 2020. [Online]. Available: https://doi.org/10.1093/bioinformatics/btaa921

[3] Y. Zhou, S. Liu, J. Siow, X. Du, and Y. Liu, "Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/49265d2447bc3bbfe9e76306ce40a31f-Paper.pdf

[4] L. Invernizzi, P. M. Comparetti, S. Benvenuti, C. Kruegel, M. Cova, and G. Vigna, "EvilSeed: A guided approach to finding malicious web pages," in *2012 IEEE Symposium on Security and Privacy*, 2012, pp. 428–442.

[5] I. Khalil, B. Guan, M. Nabeel, and T. Yu, "Killing two birds with one stone: Malicious domain detection with high accuracy and coverage," *arXiv preprint arXiv:1711.00300*, 2017.

[6] J. Topping, F. Di Giovanni, B. P. Chamberlain, X. Dong, and M. M. Bronstein, "Understanding over-squashing and bottlenecks on graphs via curvature," *arXiv preprint arXiv:2111.14522*, 2021.

[7] Cisco, "Global Threat Alerts in Secure Endpoint - Dashboard [Cisco Secure Endpoint] - Cisco," 2023, last accessed April 22 2023. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/security/amp/endpoints/global-threat-alerts-in-secure-endpoint/m_dashboard.html

[8] J. Gasteiger, S. Weißenberger, and S. Günnemann, "Diffusion improves graph learning," 2022.

[9] L. Yang, M. Li, L. Liu, bingxin niu, C. Wang, X. Cao, and Y. Guo, "Diverse message passing for attribute with heterophily," in *Advances in Neural Information Processing Systems*, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., 2021. [Online]. Available: https://openreview.net/forum?id=4jPVcKEYpSZ

[10] D. B. Acharya and D. H. Zhang, "Feature selection and extraction for graph neural networks," 2019.

[11] P. Procházka, M. Mareš, and M. Dědič, "Scalable graph size reduction for efficient GNN application," *Information Technologies – Applications and Theory 2022*, 2022.

[12] J. Pearl, "Reverend bayes on inference engines: A distributed hierarchical approach," in *Proceedings of the Second AAAI Conference on Artificial Intelligence*, ser. AAAI'82. AAAI Press, 1982, pp. 133–136.

[13] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," 2016.

[14] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[15] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *The Semantic Web: 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, Proceedings 15*. Springer, 2018, pp. 593–607.

[16] H. Pei, B. Wei, K. C.-C. Chang, Y. Lei, and B. Yang, "Geom-GCN: Geometric graph convolutional networks," 2020.

[17] X. Zheng, Y. Liu, S. Pan, M. Zhang, D. Jin, and P. S. Yu, "Graph neural networks for graphs with heterophily: A survey," 2022.

[18] J. You, R. Ying, and J. Leskovec, "Design space for graph neural networks," *CoRR*, vol. abs/2011.08843, 2020. [Online]. Available: https://arxiv.org/abs/2011.08843

[19] Š. Dvořák, P. Procházka, and L. Bajer, "GNN-based malicious network entities identification in large-scale network data," in *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2022, pp. 1–4.

[20] I. Khalil, T. Yu, and B. Guan, "Discovering malicious domains through passive DNS data graph analysis," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications*

Security, ser. ASIA CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 663–674. [Online]. Available: https://doi.org/10.1145/2897845.2897877

[21] R. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, "GNNExplainer: Generating explanations for graph neural networks," 2019.

[22] S. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," 2017.

[23] C. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*, ser. An Introduction to Information Retrieval. Cambridge University Press, 2008.

[24] W. L. Hamilton, "Graph representation learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 14, no. 3, pp. 1–159, 2020.

[25] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," 2014.

[26] H. Dai, B. Dai, and L. Song, "Discriminative embeddings of latent variable models for structured data," 2020.

[27] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2017.

[28] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," 2018.

[29] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2018.

[30] J. Drchal, "Lecture 12: Ensembling," 2020, last accessed May 12 2023. [Online]. Available: https://cw.fel.cvut.cz/b201/_media/courses/be4m33ssu/ensembling_w2020.pdf

[31] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, Octorber 2001. [Online]. Available: https://doi.org/10.1023/A:1010933404324

[32] T. K. Ho, "Random decision forests," in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, vol. 1, 1995, pp. 278–282 vol.1.

[33] "What is Random Forest? | IBM — ibm.com," https://www.ibm.com/topics/random-forest, last accessed May 12 2023.

[34] [Online]. Available: https://scikit-learn.org/stable/auto_examples/inspection/plot_permutation_importance.html

[35] M. Saad, "Detailed Explanation of Random Forests Features importance Bias — eng.mohammed.saad.18," https://medium.com/@eng.mohammed.saad.18/detailed-explanation-of-random-forests-features-importance-bias-8755d26ac3bc, last accessed May 12 2023.

[36] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should i trust you?": Explaining the predictions of any classifier," 2016.

[37] L. S. Shapley, "Notes on the n-person game—ii: The value of an n-person game.(1951)," *Lloyd S Shapley*, 1951.

[38] T. Kroupa, "The shapley value," 2020, last accessed May 12 2023. [Online]. Available: https://cw.fel.cvut.cz/b201/_media/courses/be4m36mas/cg02_lectures.pdf

[39] C. Molnar, *Interpretable Machine Learning*, 2nd ed. Independently published, 2022. [Online]. Available: https://christophm.github.io/interpretable-ml-book

[40] M. Sundararajan and A. Najmi, "The many shapley values for model explanation," 2020.

[41] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore, "Automating the construction of internet portals with machine learning," *Information Retrieval*, vol. 3, pp. 127–163, 2000.

[42] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su, "ArnetMiner: Extraction and mining of academic social networks," in *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '08. New York, NY, USA: Association for Computing Machinery, 2008, pp. 990–998. [Online]. Available: https://doi.org/10.1145/1401890.1402008

[43] A. Bojchevski and S. Günnemann, "Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking," 2018.

[44] Z. Yang, W. W. Cohen, and R. Salakhutdinov, "Revisiting semi-supervised learning with graph embeddings," 2016.

[45] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," *arXiv preprint arXiv:2005.00687*, 2020.

[46] B. Rozemberczki, C. Allen, and R. Sarkar, "Multi-scale attributed node embedding," 2021.

[47] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," 2019.

[48] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "Graph-SAINT: Graph sampling based inductive learning method," 2020.

[49] P. Aitken, B. Claise, and B. Trammell, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information," RFC 7011, Sep. 2013. [Online]. Available: https://www.rfc-editor.org/info/rfc7011

[50] X. Zhu and Z. Ghahramani, "Learning from labeled and unlabeled data with label propagation," *Carnegie Mellon School of Computer Science*, 2002.

[51] J. Zhu, Y. Yan, L. Zhao, M. Heimann, L. Akoglu, and D. Koutra, "Beyond homophily in graph neural networks: Current limitations and effective designs," 2020.

[52] M. E. J. Newman, "Mixing patterns in networks," *Physical Review E*, vol. 67, no. 2, feb 2003. [Online]. Available: https://doi.org/10.1103%2Fphysreve.67.026126

[53] D. Lim, F. Hohne, X. Li, S. L. Huang, V. Gupta, O. Bhalerao, and S.-N. Lim, "Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods," 2021.

[54] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[55] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[56] T. pandas development team, "pandas-dev/pandas: Pandas," Jan. 2023, if you use this software, please cite it as below. [Online]. Available: https://doi.org/10.5281/zenodo.7549438

[57] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," 2019.

[58] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

Appendix A

Additional Material

A.1 Correlation Plots

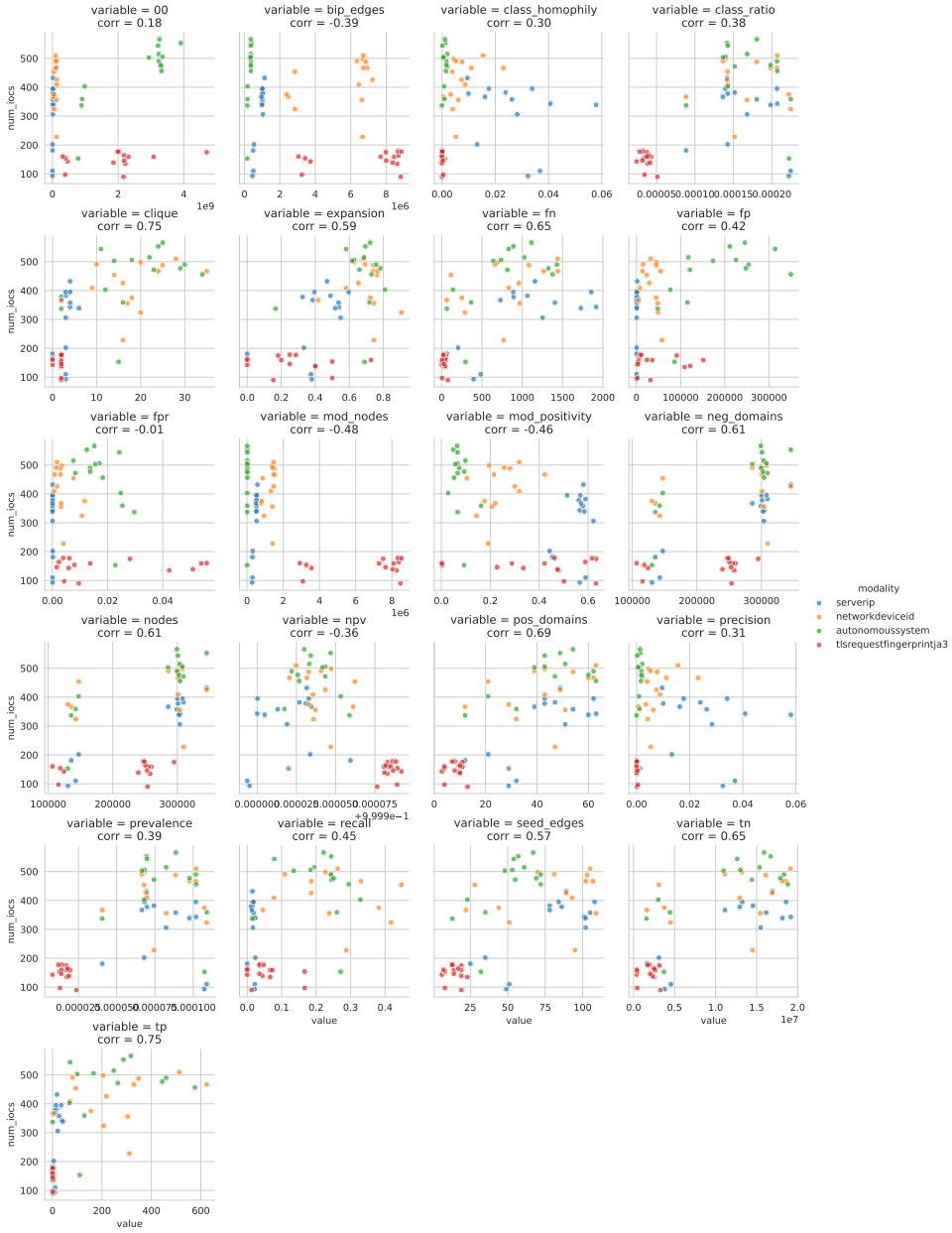


Figure A.1: Scatter plots of dependance of graph structure metrics (x-axis) with number of retrieved IoCs (y-axis) on private datasets. Correlation value is computed using Spearman correlation.

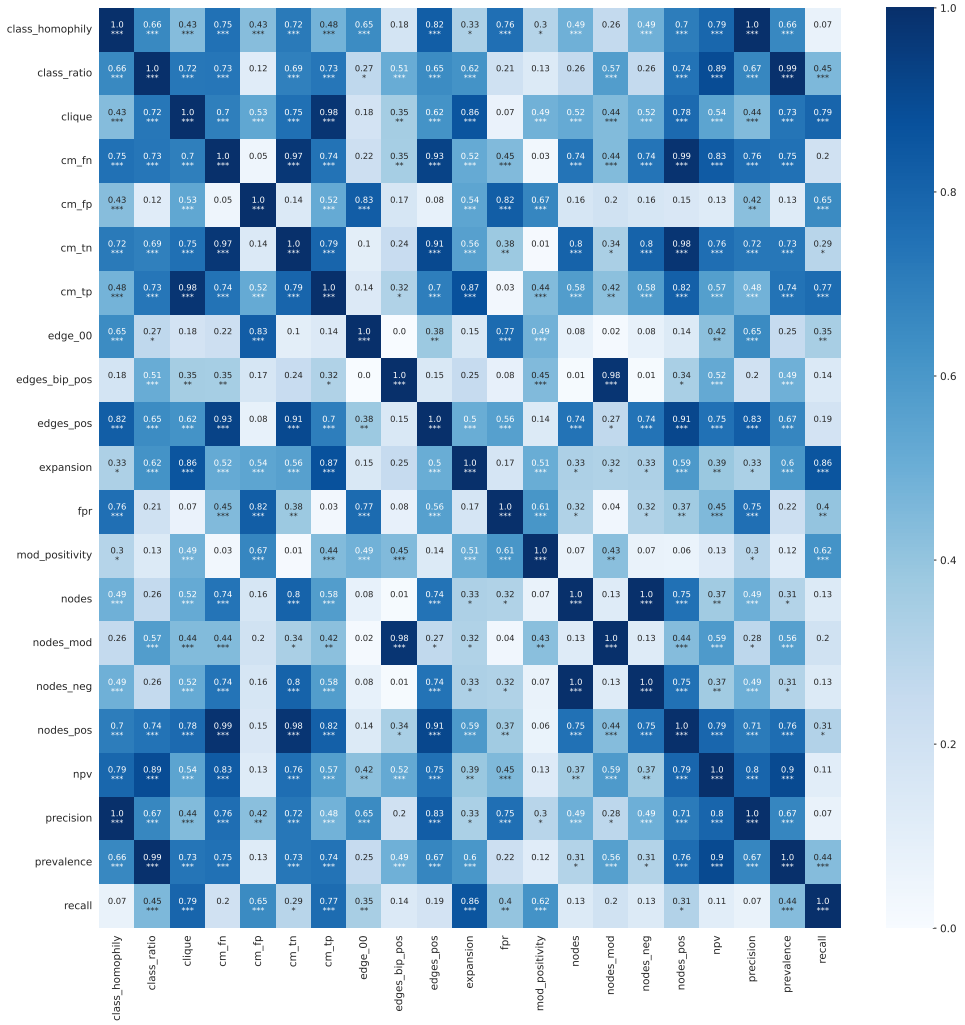


Figure A.2: Pair-wise Spearman correlation absolute values of graph structure metrics on private datasets. Asterisks represent the statistical significance level at 1% (***), 5% (**) and 10% (*).

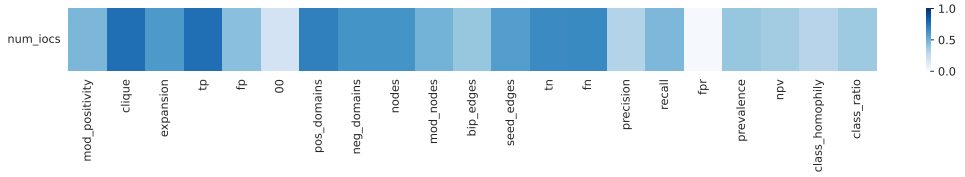


Figure A.3: Spearman correlation absolute values of graph structure metrics to performance metric of number of IoCs on private datasets.

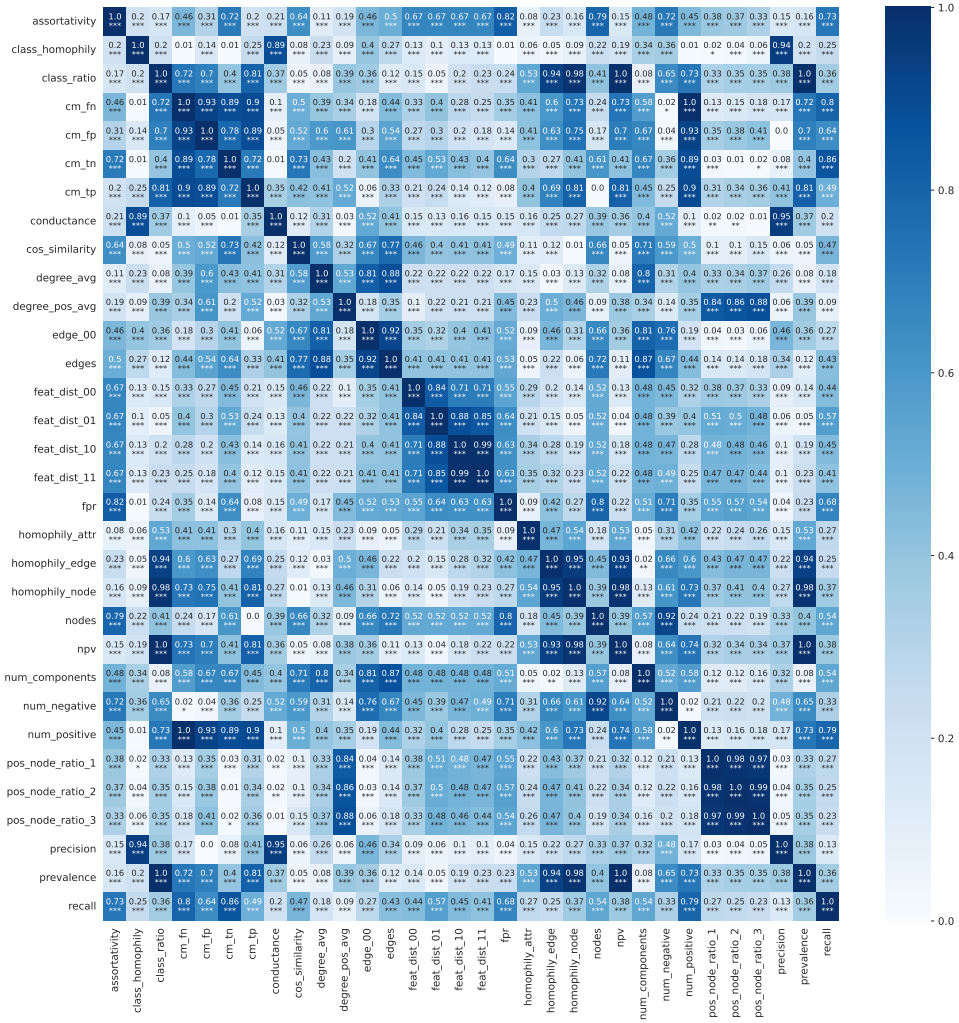


Figure A.4: Pair-wise Spearman correlation absolute values of graph structure metrics on public datasets. Asterisks represent the statistical significance level at 1% (***), 5% (**) and 10% (*).

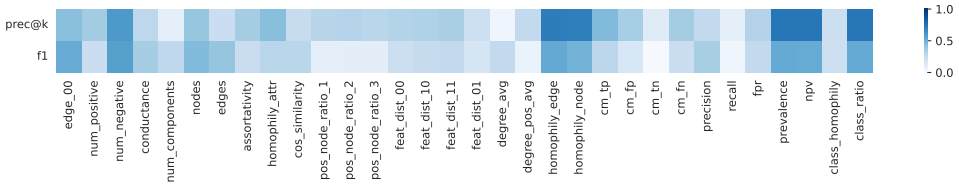


Figure A.5: Spearman correlation absolute values of graph structure metrics to performance metric of number of IoCs on public datasets.

A.2 SHAP Bar Plots

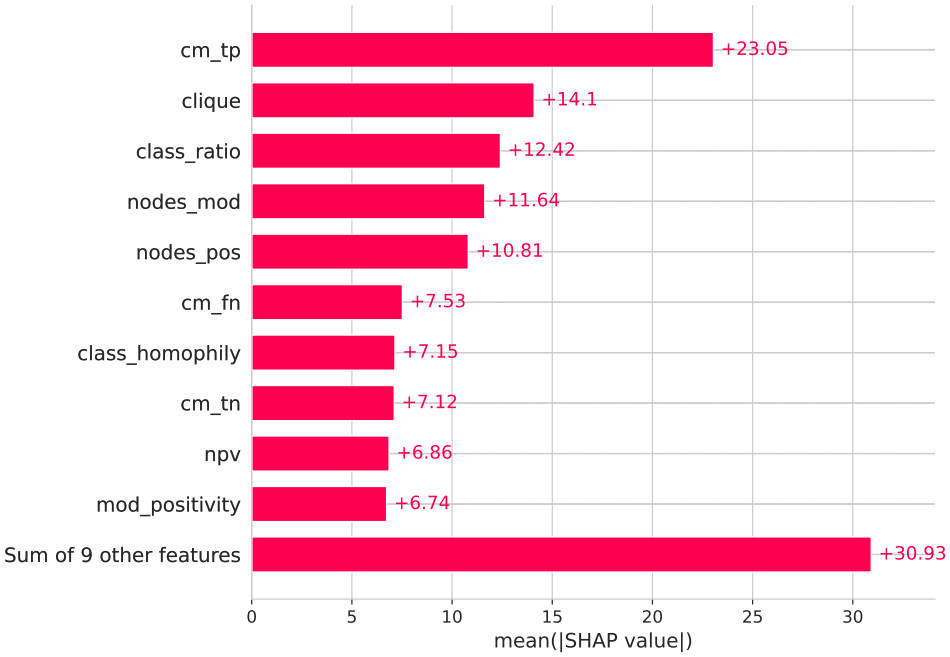


Figure A.6: SHAP feature importance magnitude plot for time generalization experiment on private dataset. Model in Figure 6.4

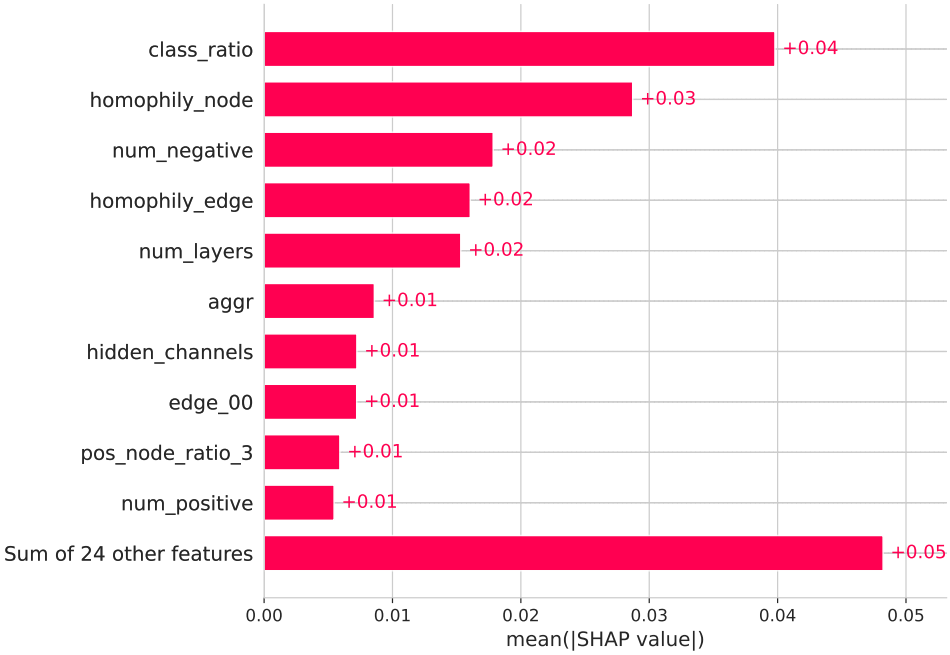


Figure A.7: SHAP feature importance magnitude plot for prec@k random split generalization experiment on public dataset. Model in Figure 6.9

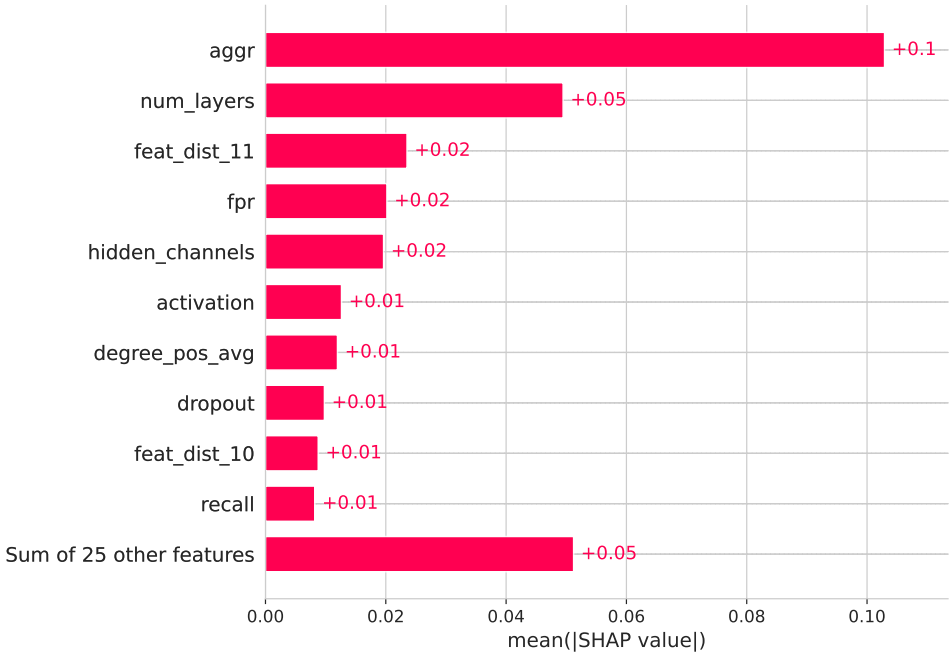


Figure A.8: SHAP feature importance magnitude plot for prec@k hyperparameter search experiment on Flickr dataset. Model in Figure 6.13

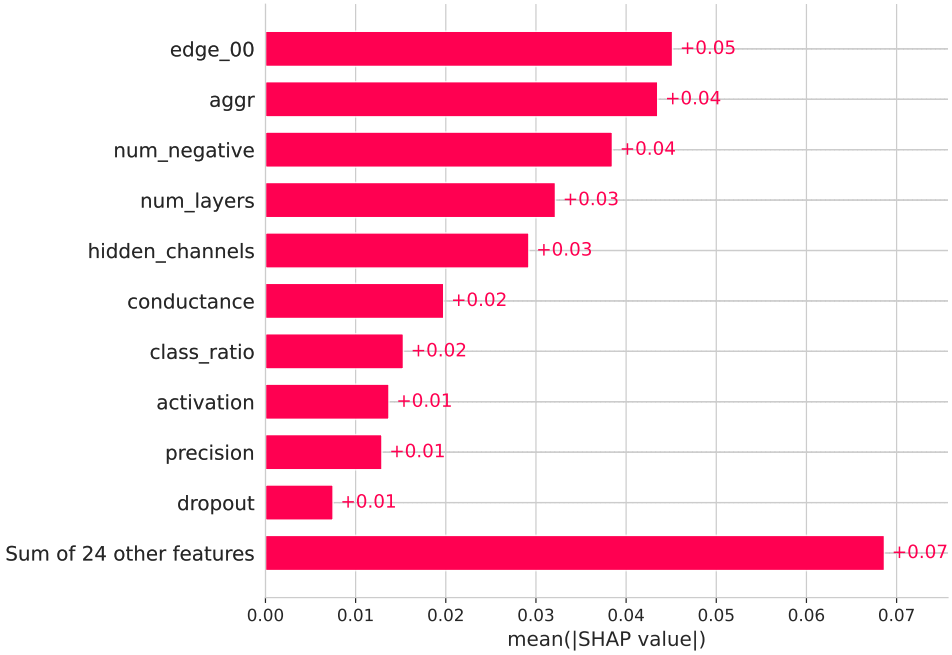


Figure A.9: SHAP feature importance magnitude plot for F1 score random split generalization experiment on public dataset. Model in Figure 6.17

A.3 Implementation Details

Algorithms in this work were implemented mainly using the PyTorch [57] and PyTorch Geometric [54] frameworks. Other libraries used include NumPy [55], Pandas [56], OGB [45] and scikit-learn [58].

Code for the public dataset evaluation is provided with this work along with the precomputed Jupyter notebook containing all the results. For the private dataset part, only the precomputed Jupyter notebook can be provided.