This article has been accepted for publication in IEEE Open Journal of the Communications Society. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/OJCOMS.2024.3514774

Received XX Month, XXXX; revised XX Month, XXXX; accepted XX Month, XXXX; Date of publication XX Month, XXXX; date of current version 11 January, 2024.

Digital Object Identifier 10.1109/OJCOMS.2024.011100

Cost-sensitive Hypergraph Learning with Structure Quality Preservation For IoT Software Defect Prediction

Nan Wang[1], Jiqiang Liu[1], Bing Du[2], Qinxin Zhao[3], Yuanlin Sun[1], Tao Zhang[1], Dunqiu Fan[4], Wenjin Li[5], Binyong Li[6]

[1]School of Cyberspace Science and Technology, Beijing Jiaotong University, Beijing 100044, China
[2]Railway Engineering Research Institute, China Academy of Railway Sciences Corporation Limited, Beijing 100081, China
[3]The State Key Laboratory for Novel Software Technology, Department of Computer Science and Technology, Nanjing University, Nanjing 210023, China
[4]The Research Department, NSFOCUS Technologies Group Co., Beijing 100085, China
[5]The Security Capability Center, NSFOCUS Inc., Beijing 100085, China
[6]Advanced Cryptography and System Security Key Laboratory of Sichuan Province, Chengdu 610000, China

CORRESPONDING AUTHOR: Tao Zhang (e-mail: taozh@bjtu.edu.cn).

This work was supported in part by the Supported by the Fundamental Research Funds for the Central Universities under Grant 2024JBMC031; in part by the NSFC Program under Grant 62202042, Grant U20A6003, Grant 62076146, Grant 62021002, Grant U19A2062, Grant 62127803, Grant U1911401 and Grant 6212780016; in part by the Aeronautical Science Foundation of China under Grant ASFC-2024Z0710M5002; in part by the Fundamental Research Funds for the CentralUniversities, JLU; in part by the Fundamental Research Funds for the CentralUniversities, JLU; in part by the by the OpenFund of Advanced Cryptography and System Security Key Laboratory of Sichuan Province(Grant No. SKLACSS-202312); in part by Industrial Technology Infrastructure Public Service Platform Project ”Public Service Platform for Urban Rail Transit Equipment Signal System Testing and Safety Evaluation” (No. 2022-233-225), Ministry of Industry and Information Technology of China.

ABSTRACT Generative AI is revolutionizing Software Engineering (SE), as both engineers and academics embrace this technology in their work. To better leverage this technology for software generation, it is essential to propose effective IoT software defect prediction methods. However, this task is challenging due to the unclear high-order correlation underlying the data. Moreover, in real-world IoT software defect prediction applications, different types of misclassifications generally lead to distinct losses and associated costs. However, accurately determining these specific costs is often not feasible. Under such circumstances, we propose a cost-sensitive hypergraph learning method with structure quality preservation (csHL^Q) to optimize the cost information and preserve the graph quality in a principled way. Due to the representational ability on high-order relationship exploring, we employ hypergraph structure instead of graph structure to model the complex correlations among the datasets. We note that if a cost-sensitive hypergraph has a high quality, its classification results may exhibit a large margin separation. Thus, csHL^Q exploits the large margin cost-sensitive hypergraph while avoiding using of a cost-sensitive hypergraph with a small margin. To measure the performance of our proposed method, we performed experiments on three distinct groups of datasets, *i.e.*, the NASA Metrics Data Program (NASA) dataset, CK metric dataset and UCI Machine Learning Repository (UCI) dataset. Experimental results and comparisons with state-of-the-art methods demonstrate the superiority of our method.

INDEX TERMS Cost-sensitive hypergraph learning, Hypergraph structure optimization, IoT software defect prediction.

I. INTRODUCTION

DISCUSSIONS on the use of generative AI range from claims of it marking the “end of programming” [1], [2] to perspectives on its potential to enhance software engineering [3], [4]. With the application of GAI in the field of software generation [5]–[7] and the rising frequency of software attacks [8], [9], the development of IoT software defect prediction techniques has gained increasing attention from researchers. Targeting the identification of software modules as defect-prone or not, IoT software defect prediction [10]–[12] has become one important research topic in the IoT software engineering field. Effective IoT software defect prediction methods can reduce the cost during the software development phase and help maintain the quality of software systems. Before releasing a reliable software system, it is essential to identify as many defects as possible. It is worth

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License. For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

This article has been accepted for publication in IEEE Open Journal of the Communications Society. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/OJCOMS.2024.3514774

noting that the conventional IoT software defect prediction process requires much human effort and is challenging to cover all possible conditions, as software defects may appear in varied ways. Therefore, how to predict software defects has attracted significant research attention [13]–[15] in recent decades.

In data-driven IoT software defect prediction, a software module is the smallest unit of functionality [16], which is represented by a group of software metric values. IoT Software defect prediction task can be viewed as a binary classification task, where a testing software module is categorized as either defect-prone or defect-free.

In recent years, many machine learning methods have been applied to this task [11], [17], [18]. Despite extensive research on IoT software defect prediction, this task continues to pose significant challenges. Firstly, the relationships between software metrics and their associated labels remain ambiguous [19]. Many methods designed to capture the complex relationships within datasets, such as graph-based approaches, fail to leverage the high-order correlations present in the data [20]. For example, in a traditional graph structure, each edge can only connect two vertices at a time, which limits the model's ability to capture high-order relationships within data [21]. To build a robust classifier, it is crucial to utilize a more effective data formulation that captures the intricate relationships within the data. Second, the misclassification of a defective module incurs a far greater cost than that of a non-defective sample, potentially causing the software to crash. Therefore, in the field of prediction, minimizing the total cost is more significant. However, increasing IoT software defect prediction works mainly focus on minimizing the number of errors rather than the total cost.

Therefore, to solve the above challenges, we propose a cost-sensitive hypergraph learning method with structure quality preservation for IoT software defect prediction. This method simultaneously optimizes cost information and preserves the quality of the hypergraph structure. The framework of our method is illustrated in Figure 1. By leveraging the hypergraph structure, we incorporate misclassification costs and perform cost-sensitive hypergraph learning to minimize the overall total cost. Unlike conventional software defect prediction methods that rely on simple graphs, where each edge represents a pairwise connection between two vertices, a hypergraph structure allows for the representation of high-order correlations among vertices. This enables the exploration of complex relationships between software modules, as hyperedges provide a flexible, degree-free connection among data points. Moreover, considering that the quality of hypergraph structure may affect the robustness of the hypergraph-based classifier, we employ large-margin criterion to evaluate the quality of the hypergraph structure [22]. More specifically, since the predictive results of a high-quality graph have a large margin separation (small hinge loss) [23] and the large margin separation has the ability to evaluate the quality of the graph structure, we set the predictive labels of multiple cost-sensitive hypergraphs as new features and construct a large margin classifier based on these features to preserve the quality of cost-sensitive hypergraph structure. Experimental results and comparisons with state-of-the-art methods show the superiority of the proposed csHL^{Q} method.

The remainder of this paper is organized as follows. Section II briefly reviews related work. Section III presents the cost-sensitive hypergraph learning method with structure quality preservation. Experimental results and comparisons with state-of-the-art methods are provided in Section IV. We conclude this paper in Section V.

II. Related Works

We're at a stage where generative AI in IoT software development is expected to accelerate progress by enabling large-scale changes with less effort [24]–[26]. However, as the initial enthusiasm has faded, there's an increasing recognition of the various risks and challenges involved, such as security concerns, unexpected failures, and trust issues. Thus, many researchers are focusing on the field of IoT software defect prediction.

A. GAI for Software Generation

Generative AI (GAI) has made significant strides in a variety of domains, including natural language processing [27], image generation [28], and software engineering [29]. In the context of software generation, GAI techniques are gaining attention for their potential to automate and accelerate the development process [30], [31]. For instance, Sauvola *et al.* [5] proposed four scenarios, outlined model trajectories to represent transitions between them, and examined them in the context of relevant software development activities. Russo *et al.* [6] developed a theoretical model for AI adoption in software engineering, termed the Human-AI Collaboration and Adaptation Framework. This model was subsequently validated through Partial Least Squares–Structural Equation Modeling, using data collected from 183 software engineers. Ebert *et al.* [7] provided guidance for both creating GAI software and developing software with the help of GAI. Practical insights are provided based on experiences in industrial environments. By employing scenario-based design and question-driven XAI design methodologies, Jiao *et al.* [32] examined users' explainability requirements for GenAI across three software engineering use cases: natural language to code, code translation, and code auto-completion. Despite the existence of several approaches that focus on using GAI for software generation, many challenges remain in this area. One of the key research hotspots is how to effectively predict software errors in GAI-generated software to improve its robustness [33]. In this paper, we investigate software defect prediction for GAI-generated software.

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License. For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

This article has been accepted for publication in IEEE Open Journal of the Communications Society. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/OJCOMS.2024.3514774

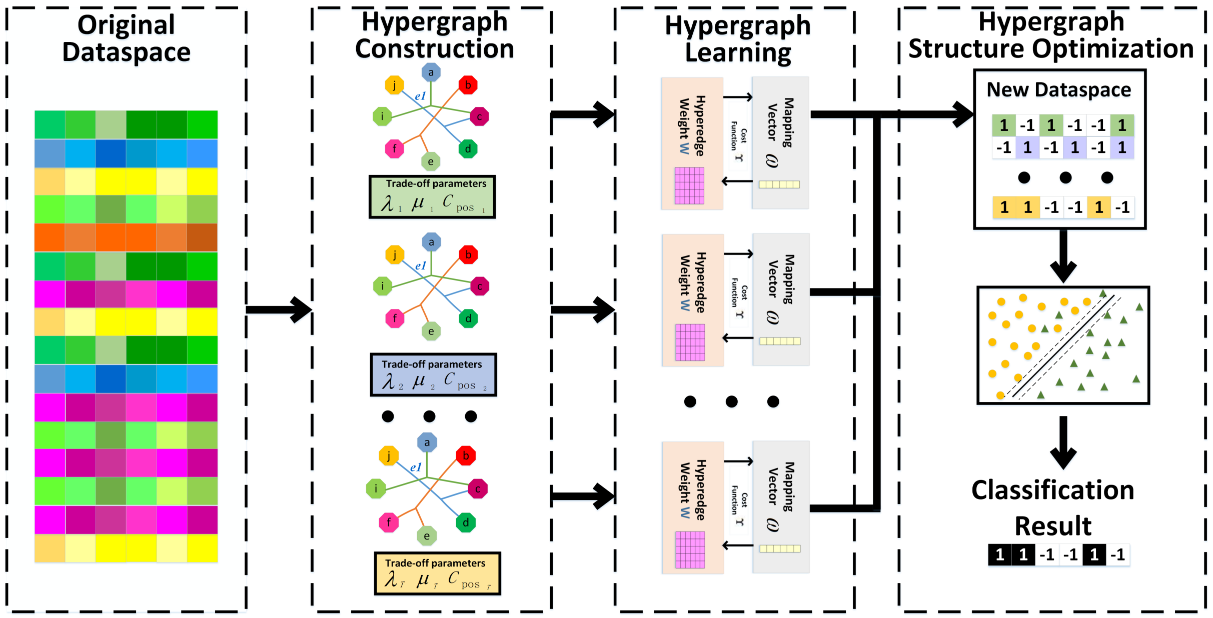


FIGURE 1. The framework of our proposed method.

B. Software Defect Prediction

The task of data-driven IoT software defect prediction can be formulated as a binary classification problem, and thus many machine learning methods have been investigated, such as parametric and non-parametric approaches. To establish the connection between software metric values and the occurrence of faults, neural network [34], contrastive learning [17], and ensemble learning [13], [35] have been integrated into this field.

To develop more effective methods, many researchers have adapted traditional machine learning techniques to address the unique challenges in IoT software defect prediction. Liapis *et al.* [13] focused on the application of active learning methods in code defect prediction and emphasizing the efficacy of combining active learning with ensemble methods, leveraging the dynamic selection and labeling of training instances to increase model performance. Chen *et al.* [36] proposed a deep hierarchical convolutional neural network (DH-CNN) based on multiple source code representations. Tong *et al.* [10] proposed a multi-source transfer weighted ensemble learning (MASTER) method for software defect prediction, which measured the weight of each source dataset based on feature importance and distribution difference and then extracted the transferable knowledge based on the proposed feature-weighted transfer learning algorithm. Yang *et al.* [37] proposed a learning-to-rank algorithm for software defect prediction by directly optimizing ranking performance. Although these methods advance the field of software defect prediction, the complex relationships among software modules and the markedly different misclassification costs for defect-free and defect-prone classes necessitate the development of more robust classifier learning algorithms.

Recognizing that the misclassification consequences for defect-free and defect-prone classes differ, many software defect prediction methods focus on cost-sensitive learning [38]–[41], which incorporates misclassification costs for different categories into the learning process. For instance, Ali *et al.* [39] introduced a cost-sensitive logistic regression and decision tree ensemble model aimed at accurately predicting defects in software components. Siers *et al.* [40] developed a framework that generates cost-sensitive predictions while addressing class imbalance, utilizing a decision forest classifier from which knowledge can be extracted through manual inspection of the individual decision trees. Liu *et al.* [42] suggested a two-phase cost-sensitive classification approach that integrated cost data during both feature selection and classification process. Wang *et al.* [43] proposed a multiple kernel ensemble learning (MKEL) method utilizing a sample weight vector updating strategy. During the training process, the weights of defect-prone samples can be increased and the weights of defect-free samples can be reduced based on the classification results. Accordingly, in this paper, we examine the high-order interactions among software modules and integrate misclassification costs into the detection process, resulting in the development of an efficient software error prediction approach.

C. Graph-based Semi-supervised Learning Method

In order to exploit the information from abundant unlabeled samples, many graph-based semi-supervised methods have been proposed [11], [44]. Zhang *et al.* [45] proposed a graph-based method, which employed the label propagation algorithm to iteratively label the testing samples. Zhang *et al.* [46] addressed the computational challenge of traditional semi-supervised methods through graph sparsification.

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License. For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

This article has been accepted for publication in IEEE Open Journal of the Communications Society. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/OJCOMS.2024.3514774

Jiang *et al.* [47] extended the Bayesian method, which constructed a graph-based sparseness-generating prior to exploiting the data manifold. In order to further improve the effectiveness of graph-based methods, many research works have been proposed to improve the performance of graph-based methods both in the graph construction process and the label propagation process [48], [49]. Considering that the construction of graphs has a critical impact on the performance of the graph-based classifier, Jebara *et al.* [50] employed a maximum weight b-matching method for graph construction to ensure the graph is exactly regular. Xu *et al.* [20] introduced a defect prediction method based on an Augmented-Code Property Graph(CPG), which utilizes a unique graph encoding format.

To evaluate the effectiveness of graph structure, li *et al.* [23] utilized margin separation to assess graph quality, enhancing the impact of graphs with large margins while seldom employing those with small margins. Since each connection of graph structure can only link two vertices at once, the pairwise connections may limit the models to investigate the high-order relationships within the datasets. A more robust framework is required to establish the relationships within the dataset. Due to its strong capacity to represent high-order relationships, hypergraph structures have become widely utilized in the field of semi-supervised learning [51], [52]. For example, Zhou *et al.* [53] employed a hypergraph structure to represent the complex relationship among datasets and extended the spectral hypergraph clustering method to hypergraph embedding and transductive classification. Zhao *et al.* [54] proposed a multi-hypergraph joint learning method to find the relevance among multiple features. Luo *et al.* [55] proposed a feature learning method named spatial-spectral hypergraph discriminant analysis (SSHGDA) to extract the spatial-spectral features of hyperspectral images. While the hypergraph structure is capable of investigating high-order relationships among data by representing complex interdependencies between multiple entities simultaneously, the quality of the hypergraph itself plays a crucial role in determining the effectiveness of hypergraph-based methods. Thus, in this paper, we focus on the characteristics of software defect prediction and perform a detailed evaluation of hypergraph detection structures. Based on this analysis, we propose an effective software defect prediction model.

III. Cost-sensitive Hypergraph Learning with Quality Preservation

A. Introduction of Hypergraph Learning

In this section, we briefly outline the key definitions of hypergraphs, the learning methods related to them, and their applications. Given a hypergraph structure $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{W})$, it generally has three components, *i.e.*, the vertices set $\mathcal{V}$, the hyperedges set $\mathcal{E}$ and the weights of hyperedges $\mathbf{W}$. We regard each sample as a vertex in a hypergraph, and the relationship among different samples is represented by hyperedges. A hypergraph structure can be defined as a $|\mathcal{V}| \times |\mathcal{E}|$ incidence matrix $\mathbf{H}$ based on the vertices set and hyperedges set, and the entry of $\mathbf{H}$ is defined as

$$h(v,e) = \begin{cases} 1 & \texttt{if } v \in e \\ 0 & \texttt{if } v \notin e \end{cases}, \tag{1}$$

which indicates whether vertex v is connected by hyperedge e or not.

The degrees of different vertices and hyperedges are calculated by

$$d(v) = \sum\nolimits_{e \in \mathcal{E}} \mathbf{W}(e)\, h(v,e) \tag{2}$$

and

$$\delta(e) = \sum\nolimits_{v \in \mathcal{V}} h(v,e). \tag{3}$$

According to the degrees of vertices and hyperedges, two diagonal matrices $\mathbf{D_v}$ and $\mathbf{D_e}$ are employed to represent the degrees of vertices and hyperedges, respectively.

Hypergraphs have been widely employed in learning methods, *e.g.*, classification, embedding, and ranking. Semi-supervised hypergraph learning method [53] is based on the assumption that the two stronger connected vertices on hypergraph structure are more likely to have similar labels. Thus, the objective formulation is defined as

$$\arg\min_{\mathbf{L}} \left\{ \Omega(\mathbf{L}) + \lambda R_{emp}(\mathbf{L}) \right\}. \tag{4}$$

In this function, $\Omega(\mathbf{L})$ is the structure regularization term that controls the connections among vertices. $R_{emp}(\mathbf{L})$ is the empirical loss and λ is the trade-off parameter to balance the influences of $\Omega(\mathbf{L})$ and $R_{emp}(\mathbf{L})$. More specifically, in order to smooth the relationships among samples on the hypergraph, the regularizer $\Omega(\mathbf{L})$ can be defined as

$$\begin{aligned}\Omega(\mathbf{L}) &= \frac{1}{2} \sum_{e \in \mathcal{E}} \sum_{u,v \in \mathcal{V}} \sum_{k=1}^{C} \frac{w(e)\mathbf{H}(u,e)\mathbf{H}(v,e)}{\delta(e)} \left(\frac{\mathbf{L}(u,k)}{\sqrt{d(u)}} - \frac{\mathbf{L}(v,k)}{\sqrt{d(v)}} \right)^2 \\ &= tr(\mathbf{L}^{\mathrm{T}} \Delta \mathbf{L}). \end{aligned} \tag{5}$$

Moreover, Δ is defined as $\Delta = \mathbf{I} - \mathbf{\Theta} = \mathbf{I} - \mathbf{D}_v{}^{-\frac{1}{2}} \mathbf{H} \mathbf{W} \mathbf{D}_e{}^{-1} \mathbf{H}^{\mathrm{T}} \mathbf{D}_v{}^{-\frac{1}{2}}$, which is the hypergraph Laplacian. Matrix $\mathbf{L}$ denotes the to-be-estimated labels for total samples. C is the number of classes in the classification tasks. In this definition, the more hyperedges connect between two vertices, the more similar their labels are.

The empirical loss $R_{emp}(\mathbf{L})$ is usually defined as

$$R_{emp}(\mathbf{L}) = \|\mathbf{L} - \mathbf{Y}\|_{\mathrm{F}}^2, \tag{6}$$

where $\mathbf{Y}$ represents the label matrix derived from the labeled samples.

Thus, the objective function can be expressed as

$$\arg\min_{\mathbf{L}} \left\{ \operatorname{tr}\left(\mathbf{L}^T \mathbf{\Delta} \mathbf{L}\right) + \lambda \|\mathbf{L} - \mathbf{Y}\|_F^2 \right\}, \tag{7}$$

which can be addressed directly using $\mathbf{L} = \left(\mathbf{I} + \frac{1}{\lambda}\mathbf{\Delta}\right)^{-1}\mathbf{Y}$.

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License. For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

This article has been accepted for publication in IEEE Open Journal of the Communications Society. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/OJCOMS.2024.3514774

Due to its strengths in modeling intricate relationships, hypergraphs are frequently applied across a range of fields, such as multi-object tracking [56], image retrieval [57], anomaly detection [58], [59], and so on.

B. Cost-Sensitive Hypergraph Learning

In this section, we include cost information into hypergraph learning to minimize the total cost for IoT software defect prediction. Given a set of training samples $\{\mathbf{A}_i, y_i\}_{i=1}^{l}$ and testing samples $\{\mathbf{A}_j\}_{j=l+1}^{n}$, We create connections between these samples using a hypergraph structure and integrate cost information into the learning process to capture the features of the test samples $\{\mathbf{A}_j\}_{j=l+1}^{n}$ to labels $\{\hat{y_j}\}_{j=l+1}^{n}$. In this section, in order to describe the proposed method more clearly, we employ boldface upper-case letters to represent matrices, boldface lower-case letters to represent vectors and normal italic letters to represent scalar.

In a hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{W})$, each vertex from $\mathcal{V}$ denotes one sample from $\{\mathbf{A}_1, \ldots, \mathbf{A}_l, \mathbf{A}_{l+1} \ldots, \mathbf{A}_n\}$. Thus, we have n vertices in total. The entries of the diagonal matrix $\mathbf{W}$ denote the weights of hyperedges. The set of hyperedges $\mathcal{E}$ connects vertices according to the distances among samples, which are generated by the common-used distance-based hyperedges generation method. Distance-based hypergraph generation methods leverage the relationships between vertices by utilizing the distance in the feature space, such as Euclidean distance. More specifically, we select one vertex from $\{\mathbf{A}_1, \ldots, \mathbf{A}_l, \mathbf{A}_{l+1} \ldots, \mathbf{A}_n\}$ as centroid vertex, and calculate the distance between the centroid vertex with the other vertices. According to the distances, the K nearest neighbors are selected to construct the hyperedge for this centroid vertex. Then a hyperedge can associate with itself and its nearest neighbors in the feature space. This process is repeated n times until all the vertices from $\{\mathbf{A}_1, \ldots, \mathbf{A}_l, \mathbf{A}_{l+1} \ldots, \mathbf{A}_n\}$ have been chosen as centroid vertices. Ultimately, this results in a total of n hyperedges.

In this method, in order to take the distances between each pair of samples into consideration, we utilize a probabilistic incidence matrix $\mathbf{H}$ to represent the relationship between hyperedges and vertices. More specifically, the (p, q)-th entry of the matrix $\mathbf{H}$ represents the connection between vertex v_p and hyperedge e_q and is defined as follows

$$\mathbf{H}(v_p, e_q) = \begin{cases} \exp\left(-\frac{d(v_p, v_{centroid})^2}{\alpha \bar{d}^2}\right) & \text{if } v_p \in e_q \\ 0 & \text{if } v_p \notin e_q \end{cases}. \quad (8)$$

Here, $v_{centroid}$ denotes the centroid vertex of hyperedge e_q, while $\bar{d}$ represents the average distance between all pairs of vertices in the hypergraph. Moreover, $d(v_p, v_{centroid})$ is the distance between v_p and $v_{centroid}$.

The definitions of degrees for vertices and hyperedges are similar to the traditional hypergraph, *i.e.*, $d(v_p) = \sum_{e \in \mathcal{E}} \mathbf{W}(e) h(v_p, e)$ for vertex v_p and $\delta(e_q) = \sum_{v \in \mathcal{V}} h(v, e_q)$ for hyperedge e_q. Then, we can get two diagonal matrices $\mathbf{D}_v$ and $\mathbf{D}_e$ for the vertices degrees and hyperedges degrees, respectively.

In order to minimize the total cost instead of accuracy in the learning process, misclassification costs have been introduced into the hypergraph learning process. The regularization formulation of cost-sensitive hypergraph learning includes three components, *i.e.*, the hypergraph Laplacian regularization term $\Omega(\boldsymbol{\omega})$, the empirical loss function $R_{emp}(\boldsymbol{\omega})$, and the optimal regularization for hypergraph structure $Q(\mathbf{W})$. Here, $\boldsymbol{\omega}$ is the mapping vector that needs to be learned, which embeds the features of samples to the labels.

The hypergraph Laplacian regularization term $\Omega(\boldsymbol{\omega})$ is similar to the traditional hypergraph structure, which is defined as follows

$$\begin{aligned}\Omega(\boldsymbol{\omega}) &= \frac{1}{2}\sum_{e \in \mathcal{E}} \sum_{v_i, v_j \in \mathcal{V}} \frac{\mathbf{W}(e)\mathbf{H}(v_i,e)\mathbf{H}(v_j,e)}{\delta(e)} \left(\frac{\boldsymbol{\omega}\mathbf{A}_i}{\sqrt{d(v_i)}} - \frac{\boldsymbol{\omega}\mathbf{A}_j}{\sqrt{d(v_j)}}\right)^2 \\ &= \sum_{v_i \in \mathcal{V}} (\boldsymbol{\omega}\mathbf{x}_i)^2 - \sum_{e \in \mathcal{E}} \sum_{v_i, v_j \in \mathcal{V}} \frac{(\boldsymbol{\omega}\mathbf{x}_i)\mathbf{H}(v_i,e)\mathbf{W}(e)\mathbf{H}(v_j,e)(\boldsymbol{\omega}\mathbf{x}_j)}{\sqrt{d(v_i)d(v_j)}\delta(e)} \\ &= (\mathbf{X}\boldsymbol{\omega})^\top \left(\mathbf{I} - \mathbf{D}_v^{-1/2}\mathbf{H}\mathbf{W}\mathbf{D}_e^{-1}\mathbf{H}^\top\mathbf{D}_v^{-1/2}\right)(\mathbf{X}\boldsymbol{\omega}) \\ &= (\mathbf{A}\boldsymbol{\omega})^\top \Delta(\mathbf{A}\boldsymbol{\omega}).\end{aligned} \quad (9)$$

As for the empirical loss $R_{emp}(\boldsymbol{\omega})$, it incorporates the cost information and is defined as

$$\mathcal{R}_{emp}(\boldsymbol{\omega}) = \|\boldsymbol{\Upsilon}(\mathbf{A}\boldsymbol{\omega} - \mathbf{y})\|_2^2 = \sum_{i=1}^{n} (\boldsymbol{\Upsilon}_{i,i}(\mathbf{A}_i\boldsymbol{\omega} - \mathbf{y}_i))^2. \quad (10)$$

Here, the diagonal matrix $\boldsymbol{\Upsilon}$ represents the misclassification cost and $\boldsymbol{\Upsilon}_{i,i}$ denotes the cost associated with the i-th sample. $\mathbf{A}\boldsymbol{\omega}$ is the classification result.

Although hypergraph has been employed to exploit high-order relationship among datasets, some hyperedges may not effectively model these correlations and the weights of different hyperedges need to be optimized in the learning process. To address this, the optimal hypergraph structure regularization $\mathbf{Q}(\mathbf{W})$ is employed to re-weight the influences of hyperedges and improve the effectiveness of hypergraphs, which is defined as $\mathbf{Q}(\mathbf{W}) = \|\mathbf{W}\|_{\mathrm{F}}^2$.

Then the objective framework of cost-sensitive hypergraph learning is summarized as follows.

$$\begin{aligned}\arg\min_{\boldsymbol{\omega}, \mathbf{W}} \quad & \left\{(\mathbf{A}\boldsymbol{\omega})^\top \Delta(\mathbf{A}\boldsymbol{\omega}) + \mu\|\mathbf{C}(\mathbf{A}\boldsymbol{\omega}) - \mathbf{y}\|_2^2 + \lambda\|\mathbf{W}\|_{\mathrm{F}}^2\right\} \\ \text{s.t.} \quad & \sum_{i=1}^{n} \mathbf{W}_{i,i} = 1, \ \forall\ \mathbf{W}_{i,i} \geq 0.\end{aligned} \quad (11)$$

Here, λ and μ are trade-off parameters to adjust the influences of the above three components.

The optimization task in Eq.(11) is convex. Then we can optimize Eq.(11) by an alternating optimization scheme. First, we fix $\mathbf{W}$ and compute the value of $\boldsymbol{\omega}$. The partial derivative of the objective function with respect to $\boldsymbol{\omega}$ is expressed as follows:

$$\frac{\partial}{\partial \boldsymbol{\omega}} \left\{\|\mathbf{C}(\mathbf{A}\boldsymbol{\omega}) - \mathbf{y}\|_2^2 + \mu(\mathbf{A}\boldsymbol{\omega})^\top \Delta(\mathbf{A}\boldsymbol{\omega})\right\} = 0, \quad (12)$$

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License. For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

This article has been accepted for publication in IEEE Open Journal of the Communications Society. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/OJCOMS.2024.3514774

and the solution of $\boldsymbol{\omega}$ is as follows

$$\boldsymbol{\omega} = \left(\mathbf{A}^\top \mathbf{C}^2 \mathbf{A} + \mu \mathbf{A}^\top \Delta \mathbf{A}\right)^{-1} \left(\mathbf{A}^\top \mathbf{C}\mathbf{y}\right). \tag{13}$$

Then we fixed $\boldsymbol{\omega}$ and calculate $\mathbf{W}$. The partial derivative of the objective function with respect to $\mathbf{W}$ is as follows

$$\frac{\partial}{\partial \mathbf{W}} \left\{ \mu (\mathbf{A}\boldsymbol{\omega})^\top \Delta (\mathbf{A}\boldsymbol{\omega}) + \lambda \|\mathbf{W}\|_F^2 + \eta (\sum_{i=1}^{N_e} \mathbf{W}_{i,i} - 1) \right\} = 0. \tag{14}$$

and the solution of $\mathbf{W}$ is as follows

$$\mathbf{W} = \frac{\mu \boldsymbol{\Phi}^\top \boldsymbol{\Phi} (\mathbf{D}_e)^{-1} - \alpha \mathbf{I}}{2\lambda} \tag{15}$$

and

$$\alpha = \frac{\mu \boldsymbol{\Phi} (\mathbf{D}_e)^{-1} \boldsymbol{\Phi}^\top - 2\lambda}{N_e}. \tag{16}$$

Here, $\boldsymbol{\Phi} = (\mathbf{A}\boldsymbol{\omega})^\top (\mathbf{D}_v)^{-\frac{1}{2}} \mathbf{H}$ and $\mathbf{I}$ is an identity matrix. According to the mapping vector $\boldsymbol{\omega}$, we can calculate the predictive labels with the feature of testing samples as $\mathbf{L} = \mathbf{A}\boldsymbol{\omega}$.

C. Quality Evaluation on Hypergraph Structure

In order to optimize the hypergraph structure, we evaluate the quality of the hypergraph structure in the learning process. Inspired by the effectiveness of the large margin criterion in graph-based quality evaluation [22], [23], we use the large margin criterion to maintain the integrity of the cost-sensitive hypergraph structure. More specifically, the predictive results with a small margin may cause a higher risk of the cost-sensitive hypergraph classifier. Thus, in the learning process, we emphasize cost-sensitive hypergraph with a large margin and avoid utilizing cost-sensitive hypergraph with a small margin. Here, we denote the category with a higher misclassification cost C_{pos} as positive class, and the category with a lower misclassification cost C_{neg} as negative class. We fix C_{neg} as 1, and we only consider the cost of the more important class. In the above section, cost-sensitive hypergraph learning method assumes that the precise cost value, *i.e.*, C_{pos} is known in advance. However, it is crucial to recognize that establishing a fixed, exact cost value is not feasible, which renders the direct use of cost-sensitive hypergraphs ineffective. In this section, we optimize the cost value C_{pos} and hyperparameters λ and μ according to the relationships among testing samples in the learning process automatically. With the candidate cost information and trade-off parameters, we construct a set of cost-sensitive hypergraphs and obtain a set of predictive labels for the testing samples. Then we treat the predictive results as new features for the testing samples and further judge the quality of the set of cost-sensitive hypergraphs.

More specifically, after constructing cost-sensitive hypergraphs with different cost information and trade-off parameters, we obtain a set of cost-sensitive hypergraph classifiers $\{HG_t\}_{t=1}^T$. Then let $\mathbf{L}^{(t)} = \{l_1^{(t)}, l_2^{(t)}, \ldots, l_n^{(t)}\}$ represents the predictive results of n testing samples from t-th cost-sensitive hypergraph classifier and we get the predictive results of all hypergraph classifiers $\mathbf{L} = \{\mathbf{L}^{(1)}, \mathbf{L}^{(2)}, \ldots, \mathbf{L}^{(T)}\}$. Although we obtain the experimental results of these cost-sensitive hypergraphs, it is still challenging to identify the quality of these solutions. Thus, we employ the large margin principle to evaluate the quality of these cost-sensitive hypergraphs. According to the predictive labels, we generate a new feature dataset for training and testing samples, denoted as $\{\mathbf{X}_i, y_i\}_{i=1}^n$ and the new feature vector of i-th sample is denoted as $\mathbf{X}_i = \{l_i^{(1)}, l_i^{(2)}, \ldots, l_i^{(T)}\}$, where each entry represents the result of corresponding hypergraph. With the new feature dataset, we employ the large margin principle to evaluate the quality of the corresponding cost-sensitive hypergraph.

According to [23], we construct a large margin linear classifier to distinguish the predictive results with a small margin. The basic principle for labeling is to increase the utilization of cost-sensitive hypergraphs with a large margin and decrease the utilization of those with a small margin. Then unstable and ineffective performance can be reduced. In order to find the classifier with a large margin, we need to find a linear classifier $\mathbf{f}(\mathbf{X}) = \mathbf{a}'\mathbf{X} + \mathbf{b}$ and get a label assignment of unlabeled data $\hat{y} = \{\hat{y_{l+1}}, \hat{y_{l+2}}, \ldots, \hat{y_n}\}$, which can be achieved by minimizing the following function

$$\begin{aligned} \min_{\mathbf{a},\hat{y}} \quad & \left\{ \frac{1}{2}\|\mathbf{a}\|_2^2 + \phi_1 \sum_{i=1}^{l} \mathbf{l}(y_i f(X_i)) + \phi_2 \sum_{j=l+1}^{n} \mathbf{l}(\hat{y_j} f(X_j)) \right\} \\ \text{s.t.} \quad & \hat{y_j} \in \{-1, +1\}, j = l+1, \ldots, n; \\ & |\frac{\sum_{j=l+1}^{n} \hat{y_j}}{n-l} - \frac{\sum_{i=1}^{l} y_i}{l}| \leq \beta. \end{aligned} \tag{17}$$

Here, $\mathbf{l}(z) = \max(0, 1-z)$ means the hinge loss in large margin separation. ϕ_1 and ϕ_2 are trade-off parameters that are used to balance the losses on the training and testing dataset. Considering that the objective function is similar to the traditional semi-supervised SVM (S3VM) [60], we employ the solution of S3VM to optimize the objective function. However, the solution of classical S3VM may not be able to meet the demand for cost-sensitive hypergraph structure optimization. For instance, classical S3VM usually utilizes the non-linear kernel while we need the linear kernel to achieve the optimization. Under these circumstances, we utilize alternating optimization [23] to address this problem.

First, we fix $\hat{y}$ and optimize $\mathbf{a}$. when $\hat{y}$ is fixed, Eq.17 is similar to the traditional linear SVM which can be solved by a linear SVM package. Then we fix $\mathbf{a}$ and optimize $\hat{y}$. When $\mathbf{a}$ is fixed, we note that the rank of $\hat{y}$ depends on the prediction $\mathbf{a}'\mathbf{X} + \mathbf{b}$ or the testing dataset [61]. Thus, according to [23], the predictive results on the testing dataset can be calculated as

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License. For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

This article has been accepted for publication in IEEE Open Journal of the Communications Society. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/OJCOMS.2024.3514774

$$\hat{y}_j = \begin{cases} +1 & \text{if } r_j \leq \left(\frac{2\sum_{i=1}^{l} y_i}{l} - \beta\right)(n-l) \\ -1 & \text{if } r_j \geq \left(\frac{2\sum_{i=1}^{l} y_i}{l} + \beta\right)(n-l) \\ sign(\mathbf{a}'\mathbf{X}_j + b) & \text{otherwise} \end{cases} . \tag{18}$$

Here, $\{r_1, r_2, \ldots, r_{n-l}\}$ represents the ranks of predictive results on the testing dataset. In order to further optimize the performance, we generally increase the importance of unlabeled samples in the learning process [60]. Finally, if the margins of some testing samples are still risky, we label them with a supervised learning method to ensure the robustness of the classifier. The procedure of the proposed method is shown in Algorithm 1.

Algorithm 1 The workflow of the proposed cost-sensitive hypergraph learning method focuses on preserving structural quality.

Input: The training samples $\{\mathbf{A}_i, y_i\}_{i=1}^{l}$ and the testing samples $\{\mathbf{A}_j\}_{j=l+1}^{n}$.
Parameter: The parameters of hypergraph $\{\lambda_1, \ldots, \lambda_m\}$ and $\{\mu_1, \ldots, \mu_m\}$, regularization parameters $\{\phi_1, \phi_2\}$, a set of cost values for positive category $\{C_{pos_1}, C_{pos_2}, \ldots, C_{pos_q}\}$
Output: The inferred label information for the test samples $\hat{y} = \{\hat{y_{l+1}}, \hat{y_{l+2}}, \ldots, \hat{y_n}\}$.

```
1:  For each trade-off parameters $\{\lambda\}$ and $\{\mu\}$ and cost values $\{C_{pos}\}$, construct a cost-sensitive hypergraph. Then we have a set of hypergraph classifiers $\{HG_t\}_{t=1}^{T}$.
2:  Obtain a set of mapping vectors $\{\boldsymbol{\omega}_t\}_{t=1}^{T}$ by minimizing the total cost.
3:  Calculate the labels by $\mathbf{A}\boldsymbol{\omega}$ and record the results of $HG_t$ as $\mathbf{L}^{(t)} = \{l_1^{(t)}, l_2^{(t)}, \ldots, l_n^{(t)}\}$.
4:  Construct the new feature space as $\{\mathbf{X}_i, y_i\}_{i=1}^{n}$.
5:  while $\hat{\phi_2} \geq \phi_2$ do
6:      while the result of Eq.17 does not decrease do
7:          Fix $\hat{y}$ and optimize $\mathbf{w}$ utilize a public linear SVM package.
8:          Fix $\mathbf{w}$ and optimize $\hat{y}$ by using Eq.18.
9:          $\hat{\phi_2} = 2\phi_2$
10:     end while
11: end while
12: if the margin of testing samples are still risky to use then
13:     Employ the supervised learning method to label uncertain samples.
14: end if
15: return $\hat{y} = \{\hat{y_{l+1}}, \hat{y_{l+2}}, \ldots, \hat{y_n}\}$
```

IV. Experiments

In this section, we briefly introduce the testing datasets, the commonly used evaluation criteria, compared methods, experimental results and discussion.

A. The Testing Dataset

To evaluate the effectiveness of our method, we conduct experiments using widely recognized IoT software defect prediction datasets, *i.e.*, specifically the CM1, KC3, MC2, MW1, PC1, PC3, PC4, and PC5 datasets from NASA [62], as well as the ant, ivy, jedit, lucene, synapse, velocity, and xalan datasets from the CK metrics dataset [62]. Moreover, we also conduct experiment on UCI dataset from other defect detection fields, *i.e.*, haberman, heartstatlog, LiverDisorders, sonar, SPECT, SPECTF, wpbc, australian data [63].

B. Evaluation Criteria

To assess the performance of the proposed method, we utilize the following widely accepted criteria.

1) Accuracy refers to the percentage of samples that are correctly classified relative to the total number of samples, which is calculated as Accuracy= $\frac{\text{TP+TN}}{\text{TP+FP+TN+FN}}$.
2) AUC: AUC quantifies the area under the ROC curve, which graphs the false positive rate (FPR) on the x-axis and the true positive rate (TPR) on the y-axis.
3) Precision: Precision is the proportion of defective samples that are correctly identified as defective out of the total samples classified as defective, which is calculated as Precision= $\frac{\text{TP}}{\text{TP+FP}}$.
4) F_1-measure: F_1-measure considers both PD and Precision, and it is defined as the harmonic mean of these two metrics, and calculated by $F_1 = \frac{2 \times PD \times Precision}{PD + Precision}$.

Accuracy, AUC, Precision, F_1-measure range in [0, 1], and a higher value represents a better performance.

C. Compared Methods

To measure the efficacy of our method, we benchmark it against various state-of-the-art techniques. The parameters for these comparison methods are configured to the appropriate values outlined in their publications. The details of compared methods are described as follows:

1) Cost-sensitive feature selection (CSFS) [64]. In CSFS, the authors optimized the cost information with F-measure information and employed cost information to solve the imbalance data classification issue.
2) Non-negative Sparse Graph Based Label Propagation (NGSLP) [45]. In NGSLP, the authors utilized the Laplacian score sampling strategy to construct a class-balance dataspace. The relationships between samples are evaluated through a non-negative sparse algorithm and represented as a graph structure. Then, label propagation is utilized based on this non-negative sparse graph.

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License. For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

This article has been accepted for publication in IEEE Open Journal of the Communications Society. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/OJCOMS.2024.3514774

3) Large margin graph quality judgment (LEAD) [23]. In LEAD, the authors presented a method for optimizing structure that is based on the conventional graph framework.

Additionally, we compare our method with traditional hypergraph learning techniques (HL) [53] and cost-sensitive hypergraph learning method (csHL) without structure optimization.

D. Experimental Settings

In the experiments, we randomly divided the dataset into training and testing sets, using 10%, 20%, and 30% for training, while the rest were designated for testing. This process is repeated 10 times, and the average results along with the standard deviation are reported. Moreover, in hypergraph construction process, we set the value of λ as $[0.1, 1, 10]$, the value of μ as $[0.1, 1, 10]$ and the value of cost for positive category as $[5, 10, 15]$. Then we construct 27 candidate cost-sensitive hypergraphs in total. In the hypergraph structure optimization process, the parameters ϕ_1, ϕ_2, and β are set as 1, 0.01, and 0.02, respectively.

Experiment results of all compared state-of-the-art methods on the UCI machine learning dataset, NASA dataset and CK metrics dataset are shown in Figure 2 and Figure 3. In these result figures, the bars indicate the average outcomes of various methods, while the lines denote the standard deviations for each corresponding method. Based on the experimental results, we observe that our method outperforms all the compared approaches. More specifically, we have the following observations.

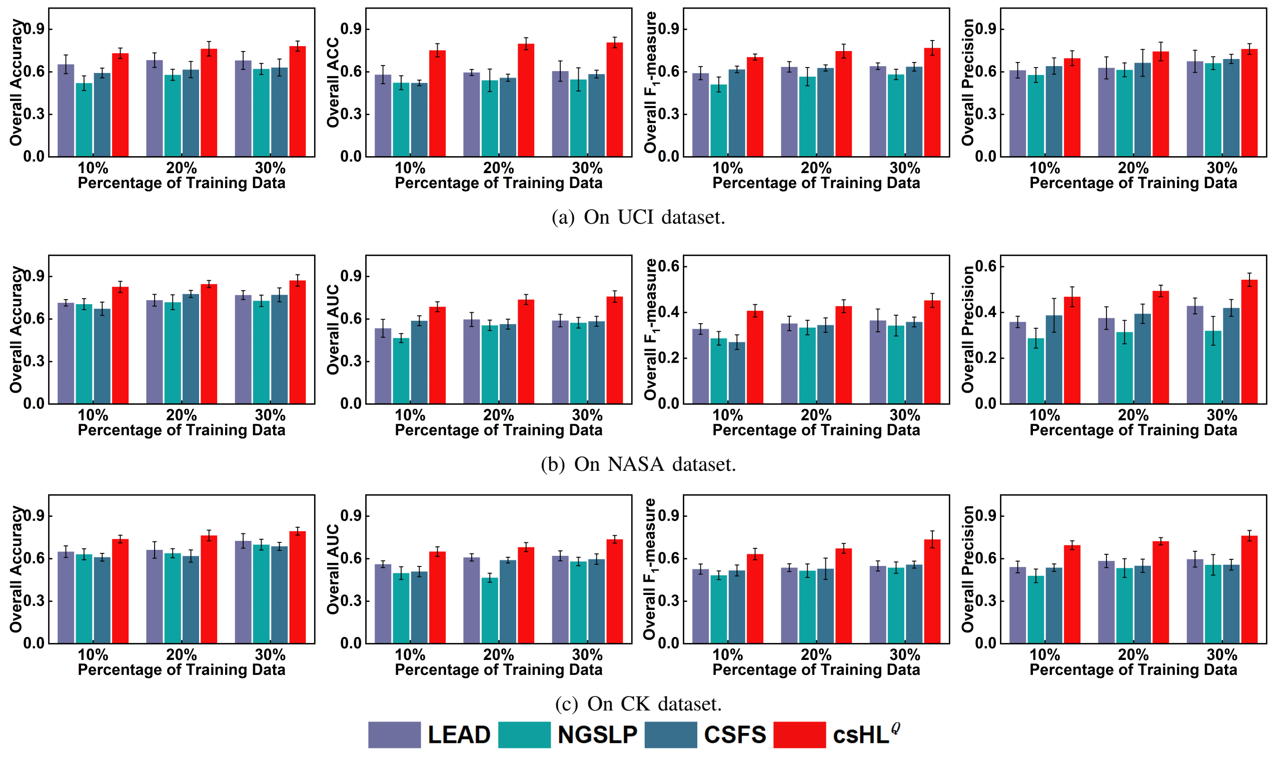
E. On Comparison with State-of-the-Art Methods

Experimental results on these three datasets are shown in the Figure2. These results indicate that our method achieves superior performance compared to state-of-the-art approaches. The detailed comparison results are shown as follows:

1) Compared with the cost-sensitive classification method, *i.e.*, CSFS, our proposed method demonstrates superior performance across all evaluation criteria. For instance, our method achieves improvements of 21.0%, 23.4%, and 15.6% in accuracy and 22.4%, 27.1%, and 31.9% in F$_1$- measure with 10%, 20%, and 30% training data on the overall CK dataset. On NASA dataset, our method achieves gains of 16.7%, 30.8%, and 28.3% in AUC and 20.9%, 25.3%, and 29.1% in precision with 10%, 20%, and 30% training data. On the UCI dataset, our method shows improvements of 14.1%, 18.9%, and 20.9% in F$_1$-measure, and 8.6%, 12.1%, and 10.2% in precision with 10%, 20%, and 30% of the training data, respectively.
2) Compared with graph-based structure optimization method, *i.e.*, LEAD, the proposed method achieves gains of 29.6%, 24.1%, and 33.3% in AUC and 13.9%, 18.4%, and 12.9% in precision with 10%, 20%, and 30% training data. on the UCI dataset. On the CK dataset, the proposed method shows enhancements of 13.7%, 15.4%, and 9.4% in accuracy, along with improvements of 20.2%, 25.3%, and 34.2% in F$_1$-measure with 10%, 20%, and 30% of the training data. These results demonstrate the superiority of the hypergraph structure. Additionally, we observe that LEAD outperforms the graph-based method without structure optimization, *i.e.*, NGSLP. For instance, on the NASA dataset, LEAD achieves improvements of 1.3%, 2.0%, and 5.6% in terms of accuracy and 14.8%, 7.5%, and 2.7% in AUC with 10%, 20%, and 30% training data, respectively. The superior performance of LEAD demonstrates that the preservation of graph quality is necessary in the learning process.
3) Compared with graph-based method, *i.e.*, NGSLP, the proposed method achieves gains of 40.6%, 31.7%, and 25.9% in terms of accuracy and 20.4%, 21.1%, and 15.1% in precision on UCI dataset. Similar compared results can be found with other evaluation criteria on other datasets. Furthermore, we find that the traditional hypergraph learning method, *i.e.*, HL, also outperforms the graph-based method, *i.e.*, NGSLP. More specifically, HL achieves gains of 12.6%, 10.3%, and 12.5% in terms of accuracy and 11.9%, 10.9%, and 11.0% in terms of F$_1$-measure on UCI dataset. These results demonstrate that the use of hypergraphs offers advantages in effectively leveraging data representation compared to traditional graph-based methods, and that preserving the quality of graph structure is essential for graph-based approaches.

Compared with state-of-the-art methods, *i.e.*, CSFS, NGSLP, LEAD, the superior performance of csHL^Q can be attributed to two main advantages. First, the proposed method utilizes a hypergraph structure to leverage high-order relationships within the dataset, contributing to the enhanced performance of all hypergraph-based approaches. The hypergraph structure can establish complex relationships through flexible hyperedges, allowing for the connection of vertices without any constraints. As shown in the experimental results, when compared with traditional graph structure *i.e.*, NGSLP, LEAD, which models the correlations among multiple samples by pairwise connection, the superiority of the hypergraph-based method is obvious. The second advantage lies in our optimization of cost information during the cost-sensitive hypergraph learning process while simultaneously preserving the quality of the hypergraph structure. Considering that the cost-sensitive hypergraph structure which constructed with uncertain cost values seriously affects the performance of cost-sensitive hypergraph learning method, preservation of the hypergraph structure quality is desirable.

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License. For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

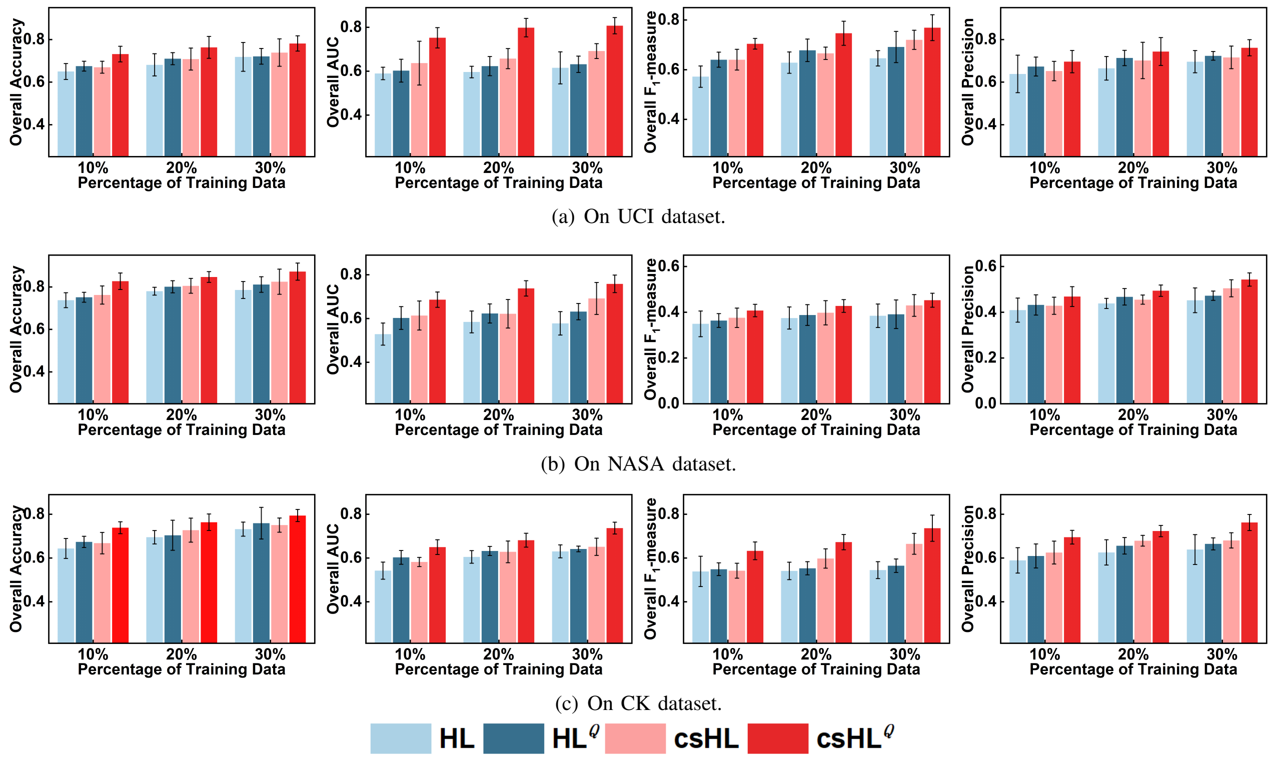


(a) On UCI dataset.

(b) On NASA dataset.

(c) On CK dataset.

FIGURE 2. Experimental results compared with state-of-the-art methods.



(a) On UCI dataset.

(b) On NASA dataset.

(c) On CK dataset.

FIGURE 3. Experimental results compared with other hypergraph-based methods.

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License. For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

This article has been accepted for publication in IEEE Open Journal of the Communications Society. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/OJCOMS.2024.3514774

F. Comparison with the Hypergraph Structure Optimization without Cost-Sensitive Learning

To further evaluate the performance of our method, we compare it with other hypergraph-based approaches, *i.e.*, traditional hypergraph (HL), hypergraph learning with quality preservation (HL^Q), cost-sensitive hypergraph learning (csHL), and cost-sensitive hypergraph learning with quality preservation(csHL^Q). As shown in Figure 3, we demonstrate the comparison results of all the compared methods. Based on the experimental results, our method demonstrates superior performance compared to other hypergraph-based approaches. For instance, compared with HL, our method achieves accuracy gains of 15.8%, 11.8%, and 18.9%, and AUC improvements of 20.2%, 25.3%, and 34.2%, when using 10%, 20%, and 30% of the training data from the CK dataset. On the UCI dataset, the proposed method attains improvements of 19.2%, 17.5%, and 20.2% in F$_1$-measure, as well as increases of 13.9%, 18.4%, and 12.9% in precision, utilizing 10%, 20%, and 30% of the training data. These observations can demonstrate the effectiveness of cost-sensitive learning process and hypergraph structure quality preservation process on the cost-sensitive hypergraph learning. We then investigate the performance of these two processes separately, specifically focusing on cost-sensitive learning and the maintenance of hypergraph structure quality. To assess the impact of cost-sensitive learning on hypergraph structure, we compare our method with conventional hypergraph learning that includes quality preservation, *i.e.*, HL^Q, the proposed method demonstrates improvements of 37.8%, 31.6%, and 32.2% in F$_1$-measure, along with gains of 20.4%, 21.1%, and 15.1% in precision on the UCI dataset using 10%, 20%, and 30% of the training data. On the NASA dataset, it achieves increases of 17.3%, 17.9%, and 19.7% in accuracy, as well as 47.3%, 32.8%, and 32.1% in AUC with the same proportions of training data. Additionally, we observe that csHL also performs better than the traditional hypergraph learning method, *i.e.*, HL. For instance, csHL achieves gains of 3.1%, 3.9%, and 2.8% in terms of accuracy and 8.0%, 10.2%, and 12.4% in terms of AUC on UCI dataset with 10%, 20%, and 30% training data. These results demonstrate the effectiveness of cost-sensitive learning process on the hypergraph structure. To further show the necessity of the hypergraph structure quality preservation process, we compare our method with cost-sensitive hypergraph learning without quality preservation, *i.e.*, csHL. On the CK dataset, our method achieves gains of 27.6%, 15.5%, and 23.5% in terms of AUC and 29.5%, 31.5%, and 36.7% in terms of precision with 10%, 20%, and 30% training data. Moreover, we further observe HL^Q performs better than HL. For example, on the UCI dataset, HL^Q achieves gains of 40.5%, 31.7%, and 25.9% in accuracy and 14.1%, 18.9%, and 20.9% in F$_1$-measure using 10%, 20%, and 30% of training data. These experimental results indicate that the quality of the hypergraph can impact classification performance, highlighting the importance of preserving the quality of the hypergraph structure.

V. Conclusion

IoT software defect prediction focuses on identifying defect-prone software modules, which is crucial for maintaining high-quality software systems. In this paper, we present a cost-sensitive hypergraph learning method that preserves structural quality for IoT software defect prediction. Hypergraph-based methods have been widely applied in various real-world applications due to their effectiveness in modeling high-order relationships. However, the quality of the hypergraph structure can significantly impact the effectiveness of the proposed method. Moreover, in various real-world scenarios, the dataset inherently involves cost sensitivity, as misclassification costs often vary significantly across categories. But for most cost-sensitive learning methods, it is impractical to ascertain the exact unique cost information. These limitations make hypergraph-based methods and cost-sensitive learning methods do not always have stable performance in many applications. Thus, the proposed method aims to tackle these challenging issues simultaneously. More specifically, the proposed method combines the cost information into hypergraph learning to solve the cost-sensitive issue in the dataset. In order to determine the precise cost value and preserve the quality of hypergraph structure, the proposed method constructs a set of cost-sensitive hypergraphs with uncertain cost information and hyperparameters. Utilizing the classification results of these hypergraphs, the proposed method also applies the large margin assumption to assess the quality of the hypergraphs and maintain the performance of the cost-sensitive hypergraph classifier. We conducted experiments using well-known IoT software defect prediction datasets, *i.e.*NASA and CK datasets and anomaly detection dataset UCI, and the results demonstrate the superiority of our proposed method.

Although the proposed method shows its advantage in defect prediction, there are still several limitations. One important limitation is the computational challenge, hypergraph learning methods require matrix operation which may limit the performance of the classifier when processing substantial datasets. To address this limitation, we will utilize an inductive learning approach instead of a transductive one to enhance the speed of our method.

REFERENCES

[1] M. Welsh, “The end of programming,” *Commun. ACM*, vol. 66, no. 1, pp. 34–35, 2023.

[2] D. M. Yellin, “The premature obituary of programming,” *Commun. ACM*, vol. 66, no. 2, pp. 41–44, 2023.

[3] M. Daun and J. Brings, “How chatgpt will change software engineering education,” in *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education*, M. Laakso, M. Monga, Simon, and J. Sheard, Eds. ACM, 2023, pp. 110–116.

[4] W. Ma, S. Liu, W. Wang, Q. Hu, Y. Liu, C. Zhang, L. Nie, and Y. Liu, “The scope of chatgpt in software engineering: A thorough investigation,” *CoRR*, vol. abs/2305.12138, 2023.

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License. For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

This article has been accepted for publication in IEEE Open Journal of the Communications Society. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/OJCOMS.2024.3514774

[5] J. J. Sauvola, S. Tarkoma, M. Klemettinen, J. Riekki, and D. S. Doermann, "Future of software development with generative AI," *Autom. Softw. Eng.*, vol. 31, no. 1, p. 26, 2024.

[6] R. Choudhuri, D. Liu, I. Steinmacher, M. A. Gerosa, and A. Sarma, "How far are we? the triumphs and trials of generative AI in learning software engineering," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 2024, pp. 184:1–184:13.

[7] C. Ebert, J. P. Arockiasamy, L. Hettich, and M. Weyrich, "Hints for generative AI software development," *IEEE Softw.*, vol. 41, no. 5, pp. 24–33, 2024.

[8] Y. Zhan, Y. Fu, L. Huang, J. Guo, H. Shi, H. Song, and C. Hu, "Cube-evo: A query-efficient black-box attack on video classification system," *IEEE Trans. Reliab.*, vol. 73, no. 2, pp. 1160–1171, 2024.

[9] C. Hu, R. Yu, B. Zeng, Y. Zhan, Y. Fu, Q. Zhang, R. Liu, and H. Shi, "Hyperattack: Multi-gradient-guided white-box adversarial structure attack of hypergraph neural networks," *CoRR*, vol. abs/2302.12407, 2023.

[10] H. Tong, D. Zhang, J. Liu, W. Xing, L. Lu, W. Lu, and Y. Wu, "MASTER: multi-source transfer weighted ensemble learning for multiple sources cross-project defect prediction," *IEEE Trans. Software Eng.*, vol. 50, no. 5, pp. 1281–1305, 2024.

[11] A. Abdu, Z. Zhai, H. A. Abdo, and R. Algabri, "Software defect prediction based on deep representation learning of source code from contextual syntax and semantic graph," *IEEE Trans. Reliab.*, vol. 73, no. 2, pp. 820–834, 2024.

[12] Z. Li, Q. Du, H. Zhang, X. Jing, and F. Wu, "An empirical study of data sampling techniques for just-in-time software defect prediction," *Autom. Softw. Eng.*, vol. 31, no. 2, p. 56, 2024.

[13] C. M. Liapis, A. Karanikola, and S. Kotsiantis, "Data-efficient software defect prediction: A comparative analysis of active learning-enhanced models and voting ensembles," *Inf. Sci.*, vol. 676, p. 120786, 2024.

[14] J. Chen, K. Hu, Y. Yu, Z. Chen, Q. Xuan, Y. Liu, and V. Filkov, "Software visualization and deep transfer learning for effective software defect prediction," in *ICSE '20: 42nd International Conference on Software Engineering*. ACM, 2020, pp. 578–589.

[15] P. R. Bal and S. Kumar, "A data transfer and relevant metrics matching based approach for heterogeneous defect prediction," *IEEE Trans. Software Eng.*, vol. 49, no. 3, pp. 1232–1245, 2023.

[16] C. Catal, "A comparison of semi-supervised classification approaches for software defect prediction," *Journal of Intelligent Systems*, vol. 23, no. 1, pp. 75–82, 2014.

[17] J. Han, C. Huang, and J. Liu, "bjcnet: A contrastive learning-based framework for software defect prediction," *Comput. Secur.*, vol. 145, p. 104024, 2024.

[18] A. Mishra and A. Sharma, "Deep learning based continuous integration and continuous delivery software defect prediction with effective optimization strategy," *Knowl. Based Syst.*, vol. 296, p. 111835, 2024.

[19] Z.-W. Zhang, X.-Y. Jing, and T.-J. Wang, "Label propagation based semi-supervised learning for software defect prediction," *Automated Software Engineering*, pp. 1–23, 2016.

[20] J. Xu, J. Ai, J. Liu, and T. Shi, "ACGDP: an augmented code graph-based system for software defect prediction," *IEEE Trans. Reliab.*, vol. 71, no. 2, pp. 850–864, 2022.

[21] F. Li, Z. Liu, J. Duan, X. Mao, H. Shi, and S. Zhang, "Exploiting conversation-branch-tweet hypergraph structure to detect misinformation on social media," *ACM Trans. Knowl. Discov. Data*, vol. 18, no. 2, pp. 33:1–33:20, 2024.

[22] V. Vapnik, "Statistical learning theory," *Wiley*, vol. 16, 1998.

[23] Y. Li, S. Wang, and Z. Zhou, "Graph quality judgement: A large margin expedition," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2016, pp. 1725–1731.

[24] H. Xu, Y. Li, O. Balogun, S. Wu, Y. Wang, and Z. Cai, "Security risks concerns of generative AI in the iot," *IEEE Internet Things Mag.*, vol. 7, no. 3, pp. 62–67, 2024.

[25] F. Alwahedi, A. Aldhaheri, M. A. Ferrag, A. Battah, and N. Tihanyi, "Machine learning techniques for iot security: Current research and future vision with generative ai and large language models," *Internet of Things and Cyber-Physical Systems*, 2024.

[26] J. Wen, J. Nie, J. Kang, D. Niyato, H. Du, Y. Zhang, and M. Guizani, "From generative ai to generative internet of things: Fundamentals, framework, and outlooks," *IEEE Internet of Things Magazine*, vol. 7, no. 3, pp. 30–37, 2024.

[27] A. Iorliam and J. A. Ingio, "A comparative analysis of generative artificial intelligence tools for natural language processing," *Journal of Computing Theories and Applications ISSN*, vol. 3024, p. 9104, 2024.

[28] S. Ali, P. Ravi, R. Williams, D. DiPaola, and C. Breazeal, "Constructing dreams using generative ai," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 21, 2024, pp. 23 268–23 275.

[29] C. Ebert and P. Louridas, "Generative ai for software practitioners," *IEEE Software*, vol. 40, no. 4, pp. 30–38, 2023.

[30] A. D. Carleton, D. Falessi, H. Zhang, and X. Xia, "Generative AI: redefining the future of software engineering," *IEEE Softw.*, vol. 41, no. 6, pp. 34–37, 2024.

[31] J. Cámara, J. Troya, J. Montes-Torres, and F. J. Jaime, "Generative AI in the software modeling classroom: An experience report with chatgpt and unified modeling language," *IEEE Softw.*, vol. 41, no. 6, pp. 73–81, 2024.

[32] J. Sun, Q. V. Liao, M. Muller, M. Agarwal, S. Houde, K. Talamadupula, and J. D. Weisz, "Investigating explainability of generative ai for code through scenario-based design," in *Proceedings of the 27th International Conference on Intelligent User Interfaces*, 2022, pp. 212–228.

[33] A. Ding, G. Li, X. Yi, X. Lin, J. Li, and C. Zhang, "Generative AI for software security analysis: Fundamentals, applications, and challenges," *IEEE Softw.*, vol. 41, no. 6, pp. 46–54, 2024.

[34] S. Kassaymeh, M. M. Al-Laham, M. A. Al-Betar, M. Alweshah, S. Abdullah, and S. N. Makhadmeh, "Backpropagation neural network optimization and software defect estimation modelling using a hybrid salp swarm optimizer-based simulated annealing algorithm," *Knowl. Based Syst.*, vol. 244, p. 108511, 2022.

[35] X. Wan, Z. Zheng, and Y. Liu, "Spe: Self-paced ensemble of ensembles for software defect prediction," *IEEE Trans. Reliab.*, vol. 71, no. 2, pp. 865–879, 2022.

[36] J. Chen, J. Xu, S. Cai, X. Wang, H. Chen, and Z. Li, "Software defect prediction approach based on a diversity ensemble combined with neural network," *IEEE Trans. Reliab.*, vol. 73, no. 3, pp. 1487–1501, 2024.

[37] X. Yang, K. Tang, and X. Yao, "A learning-to-rank approach to software defect prediction," *IEEE Transactions on Reliability*, vol. 64, no. 1, pp. 234–246, 2015.

[38] Y. Fan, X. Xia, D. A. da Costa, D. Lo, A. E. Hassan, and S. Li, "The impact of mislabeled changes by SZZ on just-in-time defect prediction," *IEEE Trans. Software Eng.*, vol. 47, no. 8, pp. 1559–1586, 2021.

[39] A. Ali, N. Khan, M. I. Abu-Tair, J. Noppen, S. I. McClean, and I. R. McChesney, "Discriminating features-based cost-sensitive approach for software defect prediction," *Autom. Softw. Eng.*, vol. 28, no. 2, p. 11, 2021.

[40] M. J. Siers and M. Z. Islam, "Novel algorithms for cost-sensitive classification and knowledge discovery in class imbalanced datasets with an application to NASA software defects," *Inf. Sci.*, vol. 459, pp. 53–70, 2018.

[41] S. Herbold, "On the costs and profit of software defect prediction," *IEEE Trans. Software Eng.*, vol. 47, no. 11, pp. 2617–2631, 2021.

[42] M. Liu, L. Miao, and D. Zhang, "Two-stage cost-sensitive learning for software defect prediction," *IEEE Transactions on Reliability*, vol. 63, no. 2, pp. 676–686, 2014.

[43] T. Wang, Z. Zhang, X. Jing, and L. Zhang, "Multiple kernel ensemble learning for software defect prediction," *Automated Software Engineering*, pp. 1–22, 2015.

[44] J. Ma, Y. Sun, P. He, and Z. Zeng, "Gsage2defect: An improved approach to software defect prediction based on inductive graph neural network," in *The 35th International Conference on Software Engineering and Knowledge Engineering*, S. Chang, Ed., 2023, pp. 45–50.

[45] Z.-W. Zhang, X.-Y. Jing, and T.-J. Wang, "Label propagation based semi-supervised learning for software defect prediction," *Automated Software Engineering*, vol. 24, no. 1, pp. 47–69, 2017.

[46] Y. Zhang, X. Zhang, X. Yuan, and C. Liu, "Large-scale graph-based semi-supervised learning via tree laplacian solver," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2016, pp. 2344–2350.

[47] B. Jiang, H. Chen, B. Yuan, and X. Yao, "Scalable graph-based semi-supervised learning through sparse bayesian model," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 12, pp. 2758–2771, 2017.

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License. For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

This article has been accepted for publication in IEEE Open Journal of the Communications Society. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/OJCOMS.2024.3514774

[48] E. Cavalleri and M. Mesiti, “Construction and enhancement of an rna-based knowledge graph for discovering new RNA drugs,” in *40th IEEE International Conference on Data Engineering*. IEEE, 2024, pp. 5639–5643.
[49] S. Zhang and W. Che, “Observer-based self-triggered resilient control for multiagent systems: A k-connected graph construction approach,” *IEEE Trans. Cybern.*, vol. 54, no. 2, pp. 706–716, 2024.
[50] T. Jebara, J. Wang, and S. F. Chang, “Graph construction and b - matching for semi-supervised learning,” in *International Conference on Machine Learning*, 2009.
[51] J. Yan, C. Li, Y. Li, and G. Cao, “Adaptive discrete hypergraph matching,” *IEEE Trans. Cybernetics*, vol. 48, no. 2, pp. 765–779, 2018.
[52] D. Du, H. Qi, L. Wen, Q. Tian, Q. Huang, and S. Lyu, “Geometric hypergraph learning for visual tracking,” *IEEE Trans. Cybernetics*, vol. 47, no. 12, pp. 4182–4195, 2017.
[53] D. Zhou, J. Huang, and B. Schölkopf, “Learning with hypergraphs: Clustering, classification, and embedding,” in *International Conference on Neural Information Processing Systems*, 2006.
[54] X. Zhao, N. Wang, Y. Zhang, S. Du, Y. Gao, and J. Sun, “Beyond pairwise matching: Person reidentification via high-order relevance learning,” *IEEE Trans. Neural Networks Learn. Syst.*, vol. 29, no. 8, pp. 3701–3714, 2018.
[55] F. Luo, B. Du, L. Zhang, L. Zhang, and D. Tao, “Feature learning using spatial-spectral hypergraph discriminant analysis for hyperspectral image,” *IEEE Trans. Cybernetics*, vol. 49, no. 7, pp. 2406–2419, 2019.
[56] L. Wen, D. Du, S. Li, X. Bian, and S. Lyu, “Learning non-uniform hypergraph for multi-object tracking,” in *The Thirty-Third AAAI Conference on Artificial Intelligence*. AAAI Press, 2019, pp. 8981–8988.
[57] Y. Zeng, Q. Jin, T. Bao, and W. Li, “Multi-modal knowledge hypergraph for diverse image retrieval,” in *Thirty-Seventh AAAI Conference on Artificial Intelligence*, B. Williams, Y. Chen, and J. Neville, Eds. AAAI Press, 2023, pp. 3376–3383.
[58] N. Wang, Y. Zhang, X. Zhao, Y. Zheng, H. Fan, B. Zhou, and Y. Gao, “Search-based cost-sensitive hypergraph learning for anomaly detection,” *Inf. Sci.*, vol. 617, pp. 451–463, 2022.
[59] H. Shi, B. Zeng, R. Yu, Y. Yang, Z. Zouxia, C. Hu, and R. Shi, “H3NI: non-target-specific node injection attacks on hypergraph neural networks via genetic algorithm,” *Neurocomputing*, vol. 613, p. 128746, 2025. [Online]. Available: https://doi.org/10.1016/j.neucom.2024.128746
[60] T. Joachims, “Transductive inference for text classification using support vector machines,” in *International Conference on Machine Learning*, 1999.
[61] K. Zhang, I. W. Tsang, and J. T. Kwok, “Maximum margin clustering made practical,” in *International Conference on Machine Learning*, 2007.
[62] T. Menzies, R. Krishna, and D. Pryor, “The promise repository of empirical software engineering data,” 2015. [Online]. Available: http://openscience.us/repo.
[63] D. Dheeru and E. Karra Taniskidou, “UCI machine learning repository,” 2017. [Online]. Available: http://archive.ics.uci.edu/ml
[64] M. Liu, C. Xu, Y. Luo, X. Chao, Y. Wen, and D. Tao, “Cost-sensitive feature selection by optimizing f-measures,” *IEEE Trans. Image Processing*, vol. PP, no. 99, 2018.

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License. For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/