Multi View Natural Network for Cross-Project Software Defect Prediction

Boy Setiawan[1] and Agus Subekti[1]
[1]Faculty of Computer Science, Nusa Mandiri University, Jakarta, Indonesia

Corresponding author: Boy Setiawan (e-mail: 14230023@nusamandiri.ac.id).

ABSTRACT Software Defect Prediction (SDP) plays a critical role in software engineering by enabling early identification of potentially defective modules, to assist developers and testers in prioritizing testing and inspection efforts to improve software quality and reliability. Driven by rapidly changing business requirements, defect prediction models have become increasingly essential in quality assurance workflows. Traditional approaches to SDP focused on Within-Project Defect Prediction (WPDP), where models are trained on historical data from the same project and effective under sufficient data conditions. This challenge motivates the adoption of Cross-Project Defect Prediction (CPDP), which leverages data from different projects. However, CPDP faces notable challenges including datasets distributional differences and class imbalance, which can degrade prediction performance and bias. To address these issues, recent studies have proposed data transformation, resampling, and domain adaptation techniques. In this study, we explore a multi-view learning approach using Neural Networks (NN) to enhance generalization and performance in CPDP scenarios. By leveraging multiple views of the same dataset—generated through concatenation of heterogeneous software metrics, imputation for missing values, normalization using Box-Cox transformation, and embedding-based feature transformation—we aim to construct a robust Multi-View Neural Network (MVNN). This architecture enables the integration of diverse information while mitigating the limitations of single-view learning in CPDP. Our method preserves more in-formation compared to conventional approaches that rely only on shared features. Experimental validation using benchmark SDP repositories demonstrates the competitiveness of our approach, offering improved performance over existing CPDP models and highlighting the potential of multi-view learning in defect prediction tasks.

KEYWORDS Cross-Project Defect Prediction, Multi-View Learning, Software Defect Prediction

I. INTRODUCTION

Software defect prediction (SDP) is a complex and critical task in the field of software engineering, focused on identifying potential defects in software systems at the early stages of development. Its primary objective is to assist developers and testers in allocating their efforts and resources more efficiently by targeting components of the codebase that are most likely to contain faults [1]. In a time where software is crucial and affected influence the bulk of daily lives, an accurate SDP helps to minimize the time and effort necessary for testing software products by automating the process of detecting the areas of software that are prone to defects early in the software development life cycle (SDLC). Defects in software include errors, flaws, mistakes, faults or bugs which may come from the absence of skills or experiences, misconceptions or requirements, uncontrollable development phase etc. [2]. As software projects continue to grow in size and complexity to facilitate rapid evolving business requirements, the demand for accelerated development has intensified in recent years. Consequently, quality assurance practices—particularly fault prediction models—have become increasingly vital. These models are primarily designed to enable the efficient allocation and prioritization of quality assurance activities, such as testing and code inspection.

The challenge with CPDP is the distribution difference between datasets from various software projects [3], [4] which cannot satisfy a similar distribution assumption in most cases. As a result, studies have been conducted to meet this challenge starting from Zimmerman et al. [5] 12 real-world applications CPDP study, and is still an active study in SDP where most CPDP methods are based on transferring the knowledge across different but related domains [3], [4], [6]. Another major challenge in CPDP is class imbalance [7], caused by fewer modules with defects in a software projects. This condition will impact the performance by favouring the majority class and introduce bias. As a result, strategies to address the class imbalance issue is also a common area of study in CPDP ranging from over sampling, under sampling and generative technique.

When data are insufficient and acquiring data from other sources is a non-trivial task especially when confidentiality is a concern in software projects, the need to maximize existing data are crucial. The standard machine learning (ML) techniques used in SDP usually consume one input for training; however, SDP can also be solved using multiple views (i.e., multiple feature vectors) [8]. Multi-view learning is an emerging field in ML which considers learning with multiple views to improve generalization, better known as data fusion or data integration [9]. Its aim is to learn to model each view and jointly optimizes all the models to improve generalization performance. A notable advantage of multi-view learning is improving generalization by manually generating multiple manual views to increase performance. Although many multiple-views ML methods such as sparse multi-view time Support Vector Machine (SVM) [10] and multi-view discriminant analysis (DA) [11] have proven capable on classification problem, we proposed NN approach since the application of multi-view is very profound in NN techniques.

Based on the problems stated above, we are focusing our study to use multiple-views with NN to select a sample dataset from SDP repositories which have different software metrics, and to construct a high quality MVNN. In order to maximize existing datasets, we are committing of using the same datasets to generate a different view of the same datasets to increase performance and generalization of the proposed MVNN. Many methods of CPDP approaches the different software metrics by utilizing only the same features between datasets [3], we opted of preserving as much information available by concatenating the datasets software metrics, and resorted to imputation to handle missing values. In the issue of dataset distribution, a Box-Cox transformation will be used to normalized the concatenate datasets to resemble a normal distribution. To generate another view of the datasets, we utilized an expand and reduce method of generating high dimensionality vector with tree-based embedding and employ a dimensionality reduction algorithm to resize it to a considerate size as the new view of the dataset in a different latent space. At the end, we will validate our findings with an empirical comparison from previous studies to show the competitiveness of our proposed method.

The key contributions of this works are as follows:

1. This study proposes a novel CPDP model based on MVNN to construct a prediction model that enhance the contribution of SDP in software engineering.
2. We introduce a novel dataset pre-processing step for CPDP to overcome the challenge of different distribution in the datasets.
3. We proposed a novel way to generate a different view of the datasets by expanding and reducing the dimensionality to produce the same datasets in a different latent space.
4. Finally, to verify the performance of the proposed method, we conducted experiments on various SDP dataset repositories with existing CPDP methods.

This paper follows the following structure. Section 1 gives introduction on the problem domain. Section 2 provides theoretical framework overview of the relevant CPDP work. Previous studies is the topic of Section 3. The presentation of our research methodology and experimental setups follow in section 4. The experimental result and discussion are presented in Section 5 along with the threats to internal, external, construct of our study and conclusions are covered in Section 6.

II. THEORETICAL FRAMEWORK

In this section we briefly introduced theoretical framework which underline our proposed MVNN method and related works on CPDP.

A. MULTI-VIEW NEURAL NETWORK

With the increasing amount of data volumes and varieties in recent years, the interest in multi-modal and heterogeneous representations has increased in popularity to enhance learning performance. MVNN have raised as a perfect approach in fusing multiple representations of data in a unified predictive model [12]. MVNN refer to neural networks architectures that integrate multiple representation of features (views) from the same data instance to improve learning performance, which leverage both redundant and complementary information across all modalities [9]. One of the fundamental challenges is representing and summarizing multimodal data in a way that exploits the complementarity and redundancy of using multiple modalities in the dataset [13]. The simplest example to overcome multiple modalities is concatenation of individual modality features known as early fusion [14] which integrates features immediately after they are extracted, resulted in a joint representations or unimodal data representation where NN excels and have become a popular method of choice. Although there are ML based multi-view algorithms for classification problems such as kernel support SVM, NN has demonstrated outstanding performance in a variety of tasks such as face recognition, object detection and classification with MVNN [9]. The superior performance has increased the popularity of NN based joint representations, combine with the ability to pre-train the representations in an unsupervised manner. However, the performance is strongly dependent on the amount of data available during training. Despite all the advantages shown by NN, one of the disadvantages comes from the inability to handle missing data, although there are ways to alleviate this issue [15].

B. MIN-MAX SCALER

The Min-Max scaler adjusts the scale of an attribute by shifting its values along the x-axis, ensuring that the transformed attribute's values fall within the interval of (0, 1) [16], according to (1):

$$x_{\text{scaled}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \quad (1)$$

In (1), the scaling factor is determined by the attribute's range, while the translational term is set as its minimum value. This approach guarantees that the attribute's values are transformed to a minimum of zero and a maximum of one which is the ideal value for NN input.

C. STANDARD SCALER

Standard scaling plays a crucial role in the SDP domain by addressing the issue of varying feature scales commonly found in datasets as a result of various metrics used on software modules. Metrics such as code complexity and lines of code often have diverse distributions and scales, which, if not properly normalized, can negatively impact the effectiveness of learning algorithms [17]. As emphasized by [18] to optimize ML performance, appropriate preprocessing steps—such as standard scaling—to ensure stable training and improve model accuracy. The standard scaler adjusts the scale of an attribute by centering the data around zero (μ becomes 0) and scales it to have a standard deviation (σ) of 1, ensuring features are on the same scale [16], according to (2):

$$x_{\text{scaled}} = \frac{x - \mu}{\sigma} \quad (2)$$

D. BOX-COX TRANSFORMATION

The Box-Cox transformation for non-negative responses is a function of the parameter λ [19] where the aim of the function is to resemble a normal gaussian distribution with mean and standard deviation of 0. Since most ML models assume a normal gaussian distribution of the dataset, this formula helps transform the dataset skew distribution. The transformed response is according to (3):

$$y(\lambda) = (y^{\lambda} - 1)/\lambda \ (\lambda \neq 0); \log y \ (\lambda = 0) \quad (3)$$

For λ = 1, there is no transformation applied. When λ = 1/2, a square root transformation is applied. For λ = 0, a logarithmic transformation is applied. For λ = -1, a reciprocal transformation is applied, ensuring continuity and avoiding issues at zero. One of the main problems of SDP comes from the different distribution or feature space of each dataset, which hinders an optimal performance across projects [3] for CPDP. The application of Box-Cox transformation in a skew dataset, helps to normalized the distribution and improve prediction results.

E. ISOMETRIC FEATURE MAPPING

Isometric Feature Mapping (Isomap) is a widely used technique for non-linear dimensionality reduction technique to overcome high dimensionality in a dataset compare to Principal Component Analysis (PCA) which excels on linear dataset. The most distinct feature of Isomap lies in its versatility tested across various applications, ranging from image processing, fault prediction in electromechanical systems, and anomaly detection in hyperspectral imagery [20]. Introduced in 2000 by Tenenbaum et al. as an improvement of multidimensional scaling (MDS) by replacing geodesic distances rather than Euclidean distances, this improvement allows Isomap to capture the true manifold structure of the dataset [21]. Beside the advantages, Isomap performs sub optimally when processing data that encompasses multiple clusters or manifold structures, but this drawback has spurred the development of modifications, including extensions from the original Isomap such as FastIsomap and Landmark Isomap, aimed at enhancing computational efficiency and the ability to handle more complex datasets effectively [22].

F. SYNTHETIC MINORITY OVERSAMPLING TECHNIQUE

Synthetic Minority Over-Sampling Technique (SMOTE) is a commonly used statistical method to address datasets with imbalance class by generating synthetic data for the minority class. Unlike simple duplication of minority instances like Random Over-Sampling (ROS), SMOTE synthesizes new data points by interpolating existing instances in the minority class until a balance minority class is achieved [23] proportionally. SDP important goal is to identify defective modules where a high proportion of the datasets suffer from significant class imbalance, with non-defective instances acts as a majority class which often leads to biased predictions favouring the majority class [3]. By applying SMOTE, the defective class presentation is bolstered to better differentiate between defective and non-defective instances without introducing bias to the model enhances the model's prediction.

G. K-NEAREST NEIGHBOUR IMPUTER

The problem of missing values in datasets is a common and well-known condition in a real-world application. This condition has a negative impact on the performance of ML models and resulted the training process and introduce errors in the model. The cause of missing values comes from various reasons and sources, but handling it is crucial to avoid bias and low final performance. A well-known technique to overcome missing value is the use of imputation by means of estimating and replacing missing values on available data, this enable the missing value to be replaced and produces a complete view of the dataset. The effectiveness of imputation strategies is influenced by the underlying missing data mechanism and categorized into three types: Missing Completely at Random (MCAR) is a missing condition where both observed and unobserved data is unrelated, and ideal for imputation.

Missing at Random (MAR) is a missing condition cause by random and depends on observed data, most imputation methods assume this condition. Missing Not at Random (MNAR) is a missing condition related to unobserved data where the missing value itself or unknown unmeasured factor influences the likelihood of missingness [24].

Advanced techniques of imputation such as K-Nearest Neighbours (KNN) does not rely on standard statistical calculation such as mean, media or mode imputation which may not capture the underlying data structure [24], but utilize the relationships between variables to predict the missing values. Its simplicity and proven effectiveness in many imputation problems has made KNN a popular choice to handle missing values [25], KNN seeks its K nearest variables and imputes by a weighted average of observed values of the identified neighbours where it was setup prior of fitting the dataset. The main problem of utilizing KNN is finding the most suitable K to produce the optimum imputation values, which could be an expensive computation experiments if done by trials and errors.

H. RANDOM TREE EMBEDDING

Random Tree Embedding (RTE) is a fundamental concept in computer science and mathematics, particularly in the representation and analysis of tree structures within different graph frameworks. This concept plays a crucial role in various applications, such as network design, data structure optimization, and algorithm development. The central idea behind RTE is to map tree-like structures into geometric spaces or graphs in a manner that maintains key properties, including the distance relationships between nodes [26]. The ability of RTE to transforms datasets to sparse high dimensionality, which maps each data point to a binary vector indicating which leaf node it originates within each tree expand the datasets in an efficient manner to uncover hidden and complex pattern which can be exploited by ML models to increase the prediction performance overall.

I. SOFTWARE DEFECT DATASETS

SDP is one of the most active research fields in software engineering and plays an important role in software quality assurance by avoiding incorrect results or unexpected behaviours from the software being developed [3]. Among various datasets available, the NASA Metrics Data Program (MDP) stands out and has garnered significant attention to become a foundational resource for SDP studies, with other notable datasets include SOFTLAB, AEEEM, ReLink and the Predictive Models in Software Engineering (PROMISE) of various open-source projects. The NASA MDP dataset consists of data from various NASA software projects which recorded historical records of software metrics and corresponding defect data, making it a benchmark for empirical evaluations in defect prediction methodologies methodologies [27]. However, study by methodologies [28] emphasis data quality issues concerning the MDP dataset ranging from inconsistencies between versions, implausible values, missing data and insufficient documentation on data pre-processing. The concerns are significant because the MDP dataset are extensively employed in SDP research and affect comparability and reliability of the studies. In this study, the [28] release of the NASA corpus as presented in Table I was utilized, which includes comprehensive datasets tailored for SDP tasks.

The NASA MDP consists of software metrics derived from static code analysis, a method that examines software without executing it to ensure the detection of potential defects early in the development cycle. This method often measures software metrics such as cyclomatic complexity, lines of code, coupling, cohesion, and others that are critical indicators of software quality and maintainability [29]. These metrics are crucial in developing reliable defect prediction models and includes recognized metrics such as McCabe's Cyclomatic Complexity and Halstead's software metrics, which play critical roles in assessing software quality and predicting potential defects [30]. One commonly analyzed metric is McCabe's Cyclomatic Complexity, which quantifies the control flow complexity of a program by measuring the number of linearly independent paths through the code. This metric helps determine the potential difficulty in understanding and maintaining the code, which correlates with defect rates [31]. Halstead's metrics extend this assessment by considering the complexity of the program through measures such as the number of operators and operands, which provide insight into the cognitive load required for software comprehension [30]. The varieties of the datasets, with multiple and different software projects, metrics and software defect patterns, ensures that the proposed models are evaluated against diverse real-world challenges, enhancing generalization and robustness of the proposed model.

Another SDP datasets utilized in this study is the SOFTLAB dataset, which is another prominent resource in the realm of SDP, designed to assist researchers and practitioners in identifying defect-prone components within software projects. It encapsulates data from multiple software projects, specifically five distinct projects developed by a Turkish software company that specializes in creating embedded controllers for home appliances. These projects—AR1, AR3, AR4, AR5, and AR6—record a consistent set of 29 software metrics, facilitating a comprehensive analysis aimed at predicting defects within software systems. This offers a unique opportunity for researchers to explore various defect prediction methodologies that can enhance the accuracy and reliability of SDP models across different contexts [32].

The significance of the Relink dataset for SDP is underscored by its use in various studies that apply advanced machine learning techniques such as federated learning and prototype learning, where recent research has tested federated prototype learning techniques on projects within the Relink dataset, aiming to improve defect prediction while maintaining data privacy through decentralized model training [33]. Moreover, the Relink dataset is instrumental for comparative studies in defect prediction. Its varied project metrics allow for in-depth performance evaluations against other prominent datasets such as AEEEM and NASA MDP.

The PROMISE repository is a well-established and publicly accessible dataset collection that plays a vital role in software engineering research, particularly in the domain of software defect prediction where it hosted various java based open-source projects for SDP datasets such as Apache Ant, Camel, Ivy, JEdit, Log4j, Lucene, Poi, Synapse, and Xerces. Assembled by [34] with the help of tools like BugInfo and CKJM, each record in the dataset represents a software module (commonly a class file), described by a set of software quality metrics such as the CK metrics, Halstead metrics, McCabe cyclomatic complexity, and LOC-based metrics [35]. Besides being used for WPDP and CPDP, the unique nature of the datasets which provided different versions of the same projects, enable it to be extensively used in Cross-Version SDP (CVDP).

The last SDP dataset used in this study is the Apache Eclipse Evolution Metrics (AEEEM) dataset, which is specifically designed to enhance the understanding and methodologies related to software quality. It originates from various projects at the Apache Software Foundation and Eclipse Foundation, and undergone detailed software metrics derived from real-world projects which comprised of 61 distinct metrics that are crucial for defect prediction analysis. These metrics encompass object-oriented measures, prior defect metrics, and code change metrics, collectively facilitating a comprehensive examination of defect-prone software components [36]. This gives AEEEM the advantage as a pivotal dataset for CPDP, where models are trained using one project's dataset and validated on another. A brief description of all the datasets used in this study is shown on Table I.

TABLE I
SDP DATASETS

Dataset		Instances	Features	Defective Instances	%	Non-Defective Instances	%
PROMISE	ivy-1.2	351	21	39	0.11	312	0.89
	jedit-4.3	491	21	11	0.02	480	0.98
	log4j-1.2	204	21	188	0.92	16	0.08
	lucene-2.4	339	21	202	0.60	137	0.40
	poi-3.0	441	21	280	0.63	161	0.37
	synapse-1.2	255	21	85	0.33	170	0.67
	velocity-1.6	228	21	77	0.34	151	0.66
	xalan-2.7	908	21	897	0.99	11	0.01
	xerces-1.4	587	21	436	0.74	151	0.26
	ant-1.7	744	21	166	0.22	578	0.78
	camel-1.6	964	21	188	0.20	776	0.80
NASA MDP	cm1	327	38	42	0.13	285	0.87
	kc1	1162	22	294	0.25	868	0.75
	mc1	1952	39	36	0.02	1916	0.98
	kc3	194	40	36	0.19	158	0.81
	mc2	124	40	44	0.35	80	0.65
	pc2	722	37	16	0.02	706	0.98
	mw1	250	38	25	0.10	225	0.90
	pc1	679	38	55	0.08	624	0.92
	jm1	7720	22	1612	0.21	6108	0.79
	pc3	1053	38	130	0.12	923	0.88
	pc4	1270	38	176	0.14	1094	0.86
	pc5	1694	39	458	0.27	1236	0.73
SOFTLAB	ar1	121	30	9	0.07	112	0.93
	ar3	63	30	8	0.13	55	0.87
	ar4	107	30	20	0.19	87	0.81
	ar5	36	30	8	0.22	28	0.78
	ar6	101	30	15	0.15	86	0.85
ReLink	zxing	399	27	118	0.30	281	0.70
	apache	194	27	98	0.51	96	0.49
	safe	56	27	22	0.39	34	0.61
AEEEM	lc	691	62	64	0.09	627	0.91
	ml	1862	62	245	0.13	1617	0.87
	pde	1497	62	209	0.14	1288	0.86
	eq	324	62	129	0.40	195	0.60
	jdt	997	62	206	0.21	791	0.79

J. CROSS-PROJECT DEFECT SOFTWARE METRICS

While WPDP is a process of using the same project dataset both as training and testing, Jing et al. [3] differentiate CPDP using the metrics similarity and size used between datasets of various projects. CPDP methods are based on the assumption that the data of source and target companies should have the same software metrics. When no common metrics exist between source and target projects, in general existing CPDP methods cannot be used for defect prediction. In this study we opted to concatenate the software metrics rather than to reduce it in order to keep as much information possible, and apply imputation for any missing values.

K. STRATIFIED CROSS-VALIDATION

Stratified cross-validation (CV) is prevalent and an essential technique when class imbalance exists in the datasets. By maintaining the relative distribution of the classes, each fold ensures that the distribution of the classes is preserved during training and testing, which is beneficial for underrepresented classes compare with K-fold CV [37] and increases the generalizability of the models and performance across metrics such as precision, recall, F1 scores. As shown in Table 1 and 2, class imbalance is a major challenge in SDP datasets, where the minor class of defective equals Yes is only a small fraction of the overall instances and is the primary target class to predict. The use of imbalanced datasets for training a classifier will most likely generate a classifier that tends to over-predict the presence of the majority class but a lower probability of predicting the minority or faulty modules. When the model predicts the minority class, it often has a higher error rate compared to predictions for the majority class. Although there are various methods to overcome the disadvantages of training a model with an imbalance dataset such as over and under sampling either with random or synthetics data, very few studies focused in the area of SDP. Most notable study concludes that sampling techniques improved the prediction performance of linear and logistics models, whilst NN and tree-based classification tree did not have a better performance upon application of the sampling techniques [3].

III. PREVIOUS RESEARCH

Over the years, a variety of approaches have been proposed and applied within SDP to assist practitioners in optimizing the use of limited testing resources by focusing on modules that are more likely to be defective. Initial research efforts primarily emphasized WPDP, where models are trained using historical data commonly organized in the form of datasets [3] from the same project and subsequently employed to predict defects in upcoming releases. Findings from early studies suggest that when sufficient training data from the same project is available, the resulting prediction models tend to perform well within that context [7]. As a result, researchers naturally consider using the sample data from well-known software projects in order to learn the model and apply it to predict defects in other software projects, which is the CPDP model's design principle [3], especially in a condition when data are insufficient or non-existent for building quality defect classifier. A number of studies have been done over time to improve SDP efficacy, which can spot defect early in the SDLC. Researchers have experiment with techniques such as ML, data mining and statistical analysis [7], where a majority of research focused on WPDP. Recently greater attention is directed towards CPDP and leverage training form other projects leveraging a diverse method to construct the best CPDP model.

In the area of ML, active learning was used in the study of [4] to choose representative unlabelled modules from the target project combine with TrAdaBoost to weight the source and target project and applying the weight to a SVM. To enhance the effectiveness of cross project defect prediction, by employing a technique known as kernel twin support vector machine (DA-KTSVM) to learn the domain adaptation model, Jin et al. [41] attempted to maximize the similarity between the feature distributions of the source and target projects.

While multi-source CPDP (MSCPDP) is the focus study of [38] with tackling the challenge of using multiple dataset source to build high performance model. Liu et al. [40] proposed a two-phase transfer learning model (TPTL) that builds two defect predictors based on the two selected projects independently using TCA+ and combines their prediction probabilities to improve performance.

A novel hybrid approach using NN is done by [7] called SMOTE Correlation and Attention Gated recurrent unit based Long Short-Term Memory optimization (SCAG-LSTM), which employs a novel hybrid technique that extends the SMOTE with edited nearest neighbours (ENN) to rebalance class distributions and mitigate the issues caused by noisy and irrelevant instances in both source and target domains. Another approach of handling class imbalance and different data distribution using two-phase feature importance amplification (TFIA) is done by [39] which yield significant improvement for CPDP. Abdu et al. [1] presented GB-CPDP, a graph-based feature learning model for CPDP that uses LSTM networks to develop predictive models and Node2Vec to convert CFGs and DDGs into numerical vectors.

Another approach to tackle class imbalance issue in transfer learning was conducted by [42] called Weighted Balanced Distribution Adaptation (W-BDA) by not only considers the distribution adaptation between domains but also adaptively changes the weight of each class. An improvement of WBDA caused by increasing data or variances in the data sampling which affected the model performance is done by [32] called WBDA+ to improve the performance of the previous study on balanced distribution adaption.

IV. RESEARCH METHOD

This section details the experimental steps undertaken in this study to evaluate the proposed SDP methods. Figure 1 provides a schematic overview of the experimental framework used to validate the effectiveness of the suggested approaches. The framework was designed to ensure an empirical evaluation of the models, leveraging software metric datasets sourced from the NASA MDP, SOFTLAB, Relink, PROMISE and AEEEM repositories where arbitrary projects datasets are chosen from each SDP datasets as training and while the rests are chosen as the testing datasets.

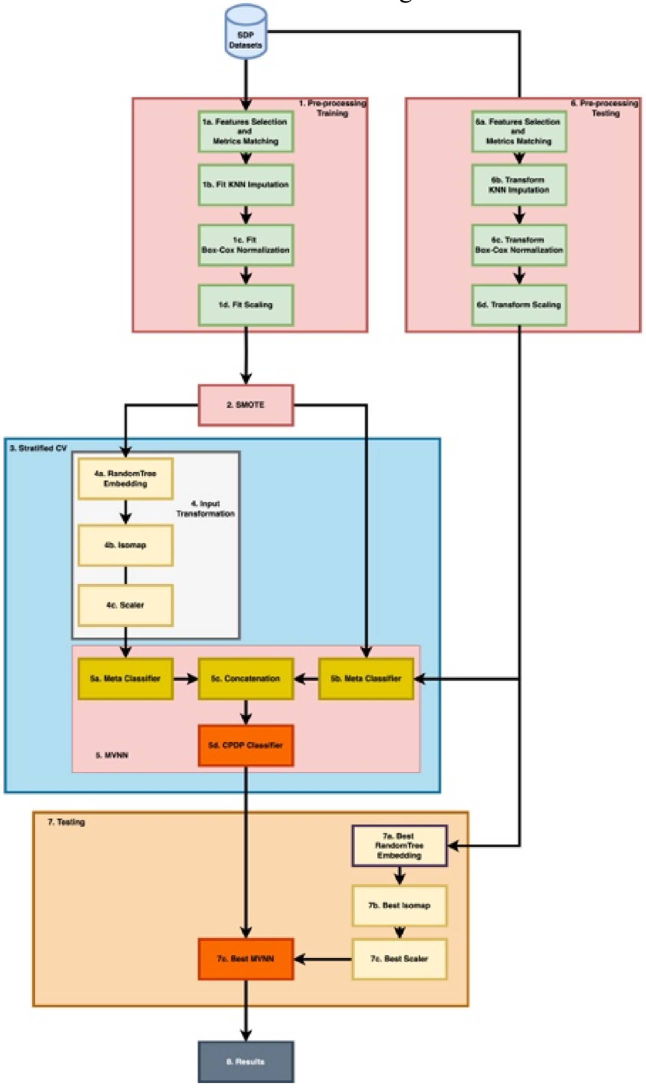

Figure 1. **Experimental Framework**

The pre-processing steps start with features selection and metrics matching, in this study we opted by selecting all the features from the training datasets and chose it as the final features for the model. As the number of features differs between datasets, null values will be introduced in the new dataset where an imputation using KNN imputer with n_neighbors = 2 was applied to overcome this condition. Unlike statistic approach of imputation of using either mean, modus etc. of the feature target, imputation using KNN calculate the value from the mean between close instances and will produce a more realistic imputation value. A study by [3] on the major challenges of SDP datasets and the impact on low prediction performances conclude that one of the causes originate from the different data distribution of each datasets especially for CPDP and Heterogenous SDP (HDP). To overcome the different distribution of each dataset, we utilize Box-Cox normalization to transform the datasets distribution to resemble a gaussian normal distribution as shown by Figure 2 on dataset kc1. Prior of normalization, the distribution of kc1 varies showing multiple different skewed distribution shape. The impact of Box-Cox transformation is clearly seen by the bell shape of the features distribution graph, where most of them have a close mean and standard deviation of 0.

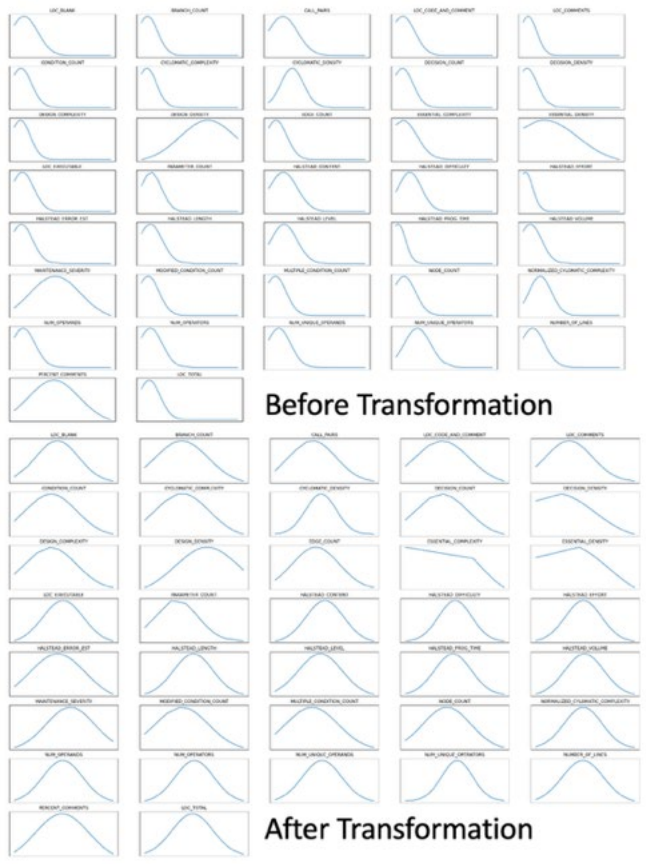

Figure 2. **Box-Cox Transformation on kc1**

Although NN has the abilities to extract features and complex relationship in the dataset and has proven effective with or without FE across various domains as detailed by [43] on comparing between ML and NN, the transformed dataset will boost NN dynamically to process input data, and learn to recognize patterns and assimilate high-level features in a hierarchical manner, effectively managing complexities in relationships among features. The last pre-processing is applying scaling to equalize the datasets and help avoid bigger value to influence any calculation during training.

To overcome the imbalance on the dataset, during the training phase SMOTE was utilized to balance the classes by interpolating existing instances in the minority class until a balance with the majority class is achieved proportionally prior of being feed for the next phase.

In this study, we employed 10 folds stratified CV in the training phase with a constant random state for reproduction. In each fold the distributions between each class is maintained to ensure the distribution is the same for all training in a 9:1 split of training and testing datasets. Each fold will be feed to a RTE (refers to Table II for hyperparameters) where the output is a sparse one hot encoding of 43197 high dimensionality one-encoding matrix, which will be reduced to a 100 dense matrix utilizing Isomap (refers to Table II for hyperparameters). A descriptive transformation of the dataset is shown in Figure 3.

TABLE II
HYPERPARAMETERS

Name	Hyperparameter	Value
RTE	n_estimators	1000
	max_depth	10
	min_samples_split	2
	min_samples_leaf	1
Isomap	n_components	100
	n_neigbours	2

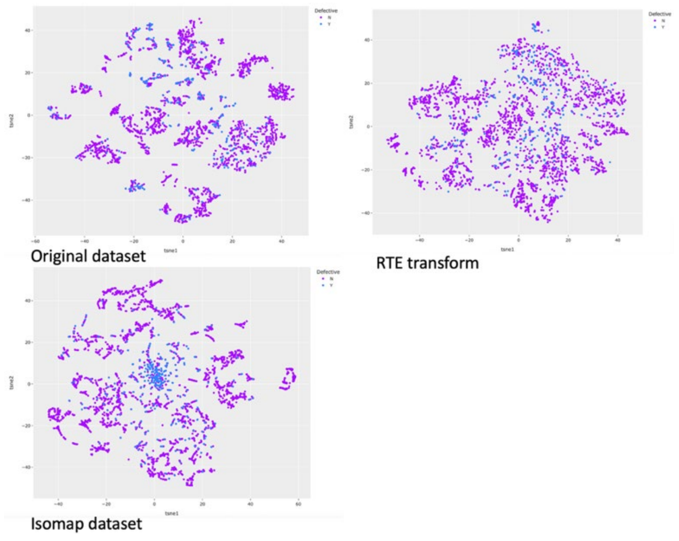

Figure 3. Original, RTE and Isomap Transformation on training dataset on concatenation of cm1, mw1 and pc1 of NASA MDP

The upper left image shows the original training dataset where it is clearly seen that the defective class is cluttered with the non-defective class. RTE transforms stretch the dataset to a condition that there is a clear and visible gap between both classes shown in the upper right image, but since the transformed dataset has a high dimensionality, training with NN int his stage will consume a huge amount of computing resources. Since there is no clear dividing line to differentiate both classes on the RTE transformed dataset, we concluded that the dataset is more closely to a non-linear dataset. The last phase is to bring down the dimensionality to a more considerate size so training can be done efficiently, the bottom left image shows the effect of Isomap where a considerable amount of defective class formed a cluster in the middle.

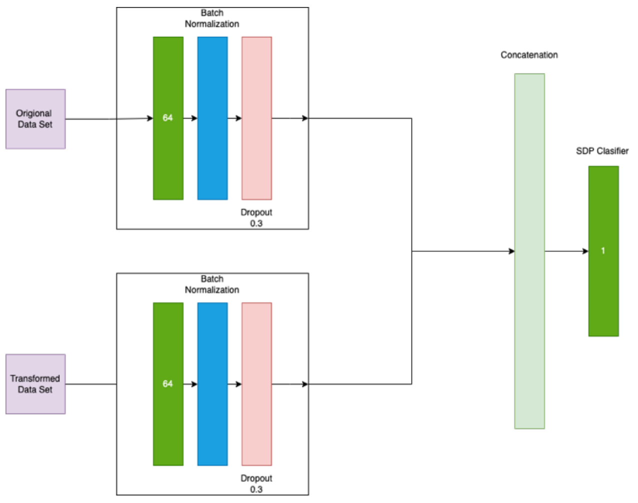

Figure 4. Our proposed MVNN

A NN with two views as inputs will be used to predict SDP with the same architecture as shown in Figure 2. Both meta models consist of one layer with 64 nodes with ReLu activation, batch normalization and dropout for regularization. The first meta-model input is the output of the pre-processing phase, while the output of the input transformation will be feed to the second meta-model. The outputs of both meta-models will be concatenated and feed to the final classifier which will predict the final result. A depiction on the proposed MVNN is shown on Figure 4.

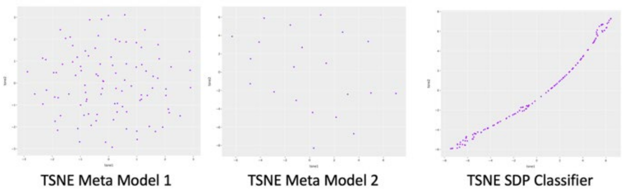

Figure 5. Our proposed MVNN

A depiction of the weights of the both meta models and the final classifier is shown in Figure 5. The upper left image shows the weights of the meta model 2 with the input from Isomap dataset transformation, compared with the meta model 1 with input from the original dataset, the Isomap transformation is spread out and shows randomness. Both views (the original and transformed datasets) are concatenated to the final SDP classifier layer where it transforms it to a more distinct solid line clearly visible to produce the final MVNN output.

TABLE III
MULTI VIEW EFFECT ON PERFORMANCE

Dataset	Model	Train	Test	acc	prc	rec	auc	f1-scr
NASA MDP	**MVNN**	**concat [kc1, pc2, mc2]**	cm1	0.92	0.93	0.92	0.91	0.93
			jm1	0.76	0.74	0.76	0.59	0.75
			kc1	0.88	0.87	0.88	0.78	0.87
			kc3	0.93	0.94	0.93	0.92	0.93
			mc1	0.93	0.95	0.93	0.91	0.94
			mc2	0.96	0.96	0.96	0.96	0.96
			mw1	0.94	0.95	0.94	0.93	0.95
			pc1	0.92	0.93	0.92	0.90	0.92
			pc2	0.96	0.97	0.96	0.95	0.96
			pc3	0.91	0.91	0.91	0.86	0.91
			pc4	0.90	0.90	0.90	0.81	0.90
			pc5	0.84	0.83	0.84	0.71	0.83
Mean				**0.91**	**0.91**	**0.91**	**0.85**	**0.90**
NASA MDP	**Single View Meta Model 1**	**concat [kc1, pc2, mc2]**	cm1	0.86	0.88	0.86	0.83	0.87
			jm1	0.81	0.77	0.81	0.56	0.76
			kc1	0.82	0.81	0.82	0.69	0.81
			kc3	0.86	0.88	0.86	0.82	0.86
			mc1	0.91	0.93	0.91	0.85	0.92
			mc2	0.87	0.89	0.87	0.86	0.87
			mw1	0.87	0.89	0.87	0.84	0.87
			pc1	0.87	0.89	0.87	0.82	0.88
			pc2	0.90	0.91	0.90	0.86	0.90
			pc3	0.86	0.87	0.86	0.77	0.86
			pc4	0.86	0.86	0.86	0.74	0.86
			pc5	0.80	0.78	0.80	0.65	0.79
Mean				**0.86**	**0.86**	**0.86**	**0.77**	**0.86**
NASA MDP	**Single View Meta Model 2**	**concat [kc1, pc2, mc2]**	cm1	0.89	0.92	0.89	0.90	0.90
			jm1	0.68	0.73	0.68	0.60	0.70
			kc1	0.81	0.84	0.81	0.77	0.82
			kc3	0.92	0.93	0.92	0.90	0.92
			mc1	0.81	0.92	0.81	0.84	0.85
			mc2	0.95	0.95	0.95	0.95	0.95
			mw1	0.91	0.93	0.91	0.91	0.92
			pc1	0.86	0.91	0.86	0.87	0.87
			pc2	0.95	0.96	0.95	0.94	0.95
			pc3	0.83	0.89	0.83	0.83	0.85
			pc4	0.82	0.87	0.82	0.79	0.84
			pc5	0.80	0.83	0.80	0.77	0.81
Mean				**0.85**	**0.89**	**0.85**	**0.84**	**0.86**

V. RESULT AND DISCUSSION

In this section, the results of the study are presented and summarizing the performance parameters used such as accuracy, precision, recall, Compute Area Under the Curve (AUC), F1-Score, and the effect of multi views as shown in Table III.

A. RESULT

Tabel III shows the result of our proposed multi views model and single view from our proposed MVNN. The first meta model with the input from the original dataset on average has the same performance as the second meta model with the Isomap transformed dataset, with the exception of 0.77 auc which is quite small compared to 0.84 of the second meta model auc result. The worst performance of both meta models come from the jm1 test dataset. Even if our proposed MVNN manage to increased the performance results on all evaluation metrics, it still performs worst on average. Even so on average, the use of both views have increased the performance results.

Using our proposed pre-processing on kc1, pc2 and mc2 datasets on the NASA MDP as training the mean final results for accuracy, precision, recall, auc and f1-score is 0.91, 0.91, 0.91, 0.85 and 0.90 respectively. The best result on the testing dataset was achieved on pc2 with auc and f1-score of 0.95 and 0.96 respectively, while the worst result was against jm1 where the auc was 0.59. A close up look at the confusion metrics on Figure 6, clearly shows there is a significant amount of wrong prediction made by the MVNN with 0.61 of false negatives and 0.16 of false positives which impacted the final performance results. On the SOFTLAB datasets, the model was retrained with ar1 and ar3 as training and the rest as testing with the average result of 0.87 auc and 0.94 f1-score. The worst result is against ar4 with auc of 0.73, the

TABLE IV
OUR PROPOSED MVNN RESULTS ON VARIOUS SDP DATASETS

Dataset	Train	Test	acc	prc	rec	auc	f1-scr
NASA MDP	**concat [kc1, pc2, mc2]**	cm1	0.92	0.93	0.92	0.91	0.93
		jm1	0.76	0.74	0.76	0.59	0.75
		kc1	0.88	0.87	0.88	0.78	0.87
		kc3	0.93	0.94	0.93	0.92	0.93
		mc1	0.93	0.95	0.93	0.91	0.94
		mc2	0.96	0.96	0.96	0.96	0.96
		mw1	0.94	0.95	0.94	0.93	0.95
		pc1	0.92	0.93	0.92	0.90	0.92
		pc2	0.96	0.97	0.96	0.95	0.96
		pc3	0.91	0.91	0.91	0.86	0.91
		pc4	0.90	0.90	0.90	0.81	0.90
		pc5	0.84	0.83	0.84	0.71	0.83
Mean			**0.91**	**0.91**	**0.91**	**0.85**	**0.90**
SOFTLAB	**concat [ar1, ar3]**	ar1	0.97	0.98	0.97	0.99	0.98
		ar3	0.98	0.99	0.98	0.99	0.98
		ar4	0.92	0.91	0.92	0.73	0.91
		ar5	0.95	0.95	0.95	0.88	0.95
		ar6	0.90	0.91	0.90	0.78	0.90
Mean			**0.94**	**0.95**	**0.94**	**0.87**	**0.94**
ReLink	**concat [apache, safe]**	apache	0.93	0.93	0.93	0.93	0.93
		safe	0.89	0.89	0.89	0.89	0.89
		zxing	0.77	0.77	0.77	0.74	0.77
Mean			**0.86**	**0.86**	**0.86**	**0.85**	**0.86**
AEEEM	**concat [eq, jdt]**	eq	0.98	0.98	0.98	0.98	0.98
		jdt	0.98	0.99	0.98	0.98	0.98
		lc	0.90	0.92	0.90	0.91	0.91
		ml	0.91	0.90	0.91	0.79	0.90
		pde	0.79	0.85	0.79	0.80	0.81
Mean			**0.91**	**0.93**	**0.91**	**0.89**	**0.92**
PROMISE	**concat [ant-1.7, camel-1.6]**	ant-1.7	0.96	0.96	0.96	0.96	0.96
		camel-1.6	0.96	0.96	0.96	0.95	0.96
		ivy-1.2	0.91	0.93	0.91	0.92	0.92
		jedit-4.3	0.89	0.93	0.89	0.91	0.90
		log4j-1.2	0.89	0.88	0.89	0.83	0.88
		lucene-2.4	0.90	0.90	0.90	0.86	0.90
		poi-3.0	0.87	0.87	0.87	0.82	0.87
		synapse-1.2	0.91	0.92	0.91	0.89	0.91
		velocity-1.6	0.92	0.92	0.92	0.89	0.92
		xalan-2.7	0.71	0.77	0.71	0.69	0.68
		xerces-1.4	0.82	0.82	0.82	0.75	0.80
Mean			**0.88**	**0.90**	**0.88**	**0.86**	**0.88**

result of 0.22 false negatives and 0.07 false positives shown in Figure 6. The same model was retrained with ReLink using apache and safe datasets as training and zxing as testing dataset achieved a below average of the training datasets with 0.85 of auc. The confusion metric of zxing shows a low false negative of 0.27 and 0.21 of false positive. The final results on the AEEEM using eq and jdt as training, showed the worst result was against pde with 0.81 of f1-score and 0.53 of false negative. The results from PROMISE using ant.1-7 and camel.1-6 using our proposed MVNN as training datasets, resulted the worst result on the testing dataset was against xalan.2-7 with 0.69 auc score and 0.09 false negative and 0.35 false positive.

B. DISCUSSION

Overall, our proposed method has shown promising result to predict defective software module by learning from some projects and inferred it in other projects with average performance above 0.87%. To validate our proposed method, we verify it with various previous CPDP studies and conduct the same experiments to compare the results. In a condition where the datasets were used in our study as training, we exempt the results for fairness and avoiding bias.

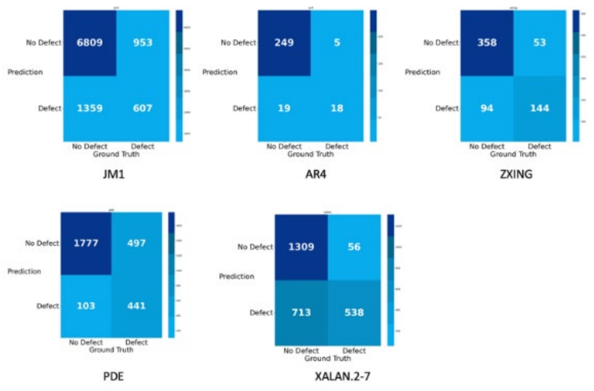

Figure 6. Test datasets confusion metric

TABLE V
OUR PROPOSED MVNN PERFORMANCE WITH AEEEM

Source	Target	ALTRA		MSCPDP		TFIA		SCAG- LSTM		Ours	
		AUC	F1-Score	AUC	F1-Score	AUC	F1-Score	AUC	F1-Score	AUC	F1-Score
EQ	JDT	0.27	0.45	0.63	0.41	0.74	0.87	0.88	0.90	0.67	0.57
EQ	LC	0.29	0.45	0.64	0.26	0.74	0.90	0.75	0.92	0.78	0.73
EQ	ML	0.25	0.30	0.56	0.24	0.70	0.88	0.73	0.90	0.61	0.51
EQ	PDE	0.20	0.42	0.56	0.26	0.74	0.86	0.77	0.88	0.64	0.58
JDT	EQ	0.39	0.53	0.58	0.27	0.72	0.86	0.90	0.89	0.82	0.89
JDT	LC	0.27	0.70	0.57	0.26	0.73	0.88	0.81	0.88	0.88	0.93
JDT	ML	0.61	0.73	0.57	0.26	0.71	0.88	0.76	0.89	0.72	0.88
JDT	PDE	0.67	0.71	0.58	0.28	0.72	0.87	0.76	0.88	0.78	0.88
LC	EQ	0.30	0.47	0.59	0.31	0.71	0.86	0.85	0.88	0.66	0.82
LC	JDT	0.44	0.87	0.67	0.49	0.77	0.91	0.85	0.92	0.67	0.86
LC	ML	0.34	0.86	0.59	0.30	0.73	0.86	0.74	0.87	0.65	0.86
LC	PDE	0.61	0.79	0.58	0.27	0.76	0.87	0.76	0.87	0.65	0.85
ML	EQ	0.58	0.71	0.53	0.16	0.76	0.89	0.85	0.89	0.83	0.91
ML	JDT	0.31	0.75	0.59	0.31	0.73	0.88	0.82	0.90	0.77	0.90
ML	LC	0.54	0.81	0.53	0.12	0.81	0.93	0.84	0.94	0.89	0.94
ML	PDE	0.29	0.81	0.56	0.23	0.72	0.87	0.75	0.87	0.80	0.88
PDE	EQ	0.37	0.64	0.55	0.23	0.79	0.90	0.88	0.91	0.83	0.89
PDE	JDT	0.39	0.80	0.62	0.39	0.75	0.88	0.79	0.89	0.82	0.90
PDE	LC	0.38	0.80	0.54	0.16	0.76	0.87	0.82	0.88	0.88	0.93
PDE	ML	0.49	0.80	0.56	0.22	0.72	0.85	0.73	0.88	0.76	0.87
Average		**0.40**	**0.67**	**0.58**	**0.27**	**0.74**	**0.88**	**0.80**	**0.89**	**0.76**	**0.83**

Table V shows the performance of various CPDP methods such as ALTRA [4], MSCPDP [38], TFIA [39], and SCAG-LSTM [7]. On average SCAG-LSTM performs the best follows by TFIA compared with ours and other methods. The low average performance happened when trained using a single dataset training which hinders our proposed method of concatenating multiple datasets input, but on any other datasets our proposed method is in par with SCAG-LSTM and TFIA and the devoid concatenation affects little on the overall performance as shown in Table IV.

Table VI compares the performance of our proposed method with WBDA [42] and WBDA+ [32] where our proposed method excels compared with WBDA and WBDA+. Compared with the experiment in Table V with only a single dataset for training, the results of Table VI show the performance of our proposed pre-processing method when trained with multiple input datasets.

Table VII compares the performance of our proposed method with TPTL [40], DA-KTSVM [41] and GB-CPDP [1] where the average mean performance shows that our proposed method outperform the other methods.

C. THREATS TO VALIDITY

As with every empirical experiment, the results of our works are subject to some threats to validity.

D. CONSTRUCT VALIDITY

We admit that during our experiments, only a subset of various SDP repositories were used and not all datasets were included in the case of PROMISE. Although it would be best to include all of them, but the limitation of resources hinders us to take this step. For objectivity, we only used datasets from the same repository to do a benchmarking or testing, we reserved ourself from modifying unless it is necessary to conduct the experiment. Since most studies on SDP uses an open and public datasets, we consider the datasets is complete and adequately fixed and reliable to be used in our study.

E. INTERNAL VALIDITY

Although there are questions regarding the validity of the dataset, especially the NASA MDP. We found it to be constructive and the necessary adjustment have been made and verified by previous studies. Therefore, the validity of the datasets should be minor and will cause little effect on the results. Another issue about the validity is the complex nature of software engineering especially the human factor such as developers' skills and experiences which might affects the likelihood of defect being introduced in the software.

F. EXTERNAL VALIDITY

We validated our findings using open and public datasets from different sources and different software metrics to gain more confidence in the external validity of our study. By doing so, we hope to achieve generalization with our proposed method, and any replicated studies with our method will be a step to improve our method.

VI. CONCLUSION

In this article, we propose MVNN for CPDP which shows to be reliable compared with previous studies. The challenges of CPDP besides the different distribution between datasets, is to find a suitable features or software metrics that can be accepted and used universally among software projects. Although there are methods and techniques to overcome the challenges of CPDP, but the complex nature of software projects still proves to be a

TABLE VI
OUR PROPOSED MVNN PERFORMANCE WITH RELINK, SOFTLAB AND NASA

Dataset	Source	Target	WBDA AUC	WBDA F1-Score	WBDA+ AUC	WBDA+ F1-Score	Ours AUC	Ours F1-Score
ReLink	apache, zxing	safe	0.74	0.70	0.72	0.70	0.91	0.92
SOFTLAB	ar1, ar4, ar5, ar6	ar3	0.69	0.57	0.65	0.59	0.92	0.94
	ar1, ar3, ar4, ar6	ar5	0.71	0.62	0.72	0.71	0.92	0.95
NASA	cm1, pc1, pc3, pc5	mw1	0.72	0.67	0.80	0.70	0.88	0.90
	mw1, pc1, pc3, pc5	cm1	0.74	0.65	0.80	0.72	0.94	0.95
Average			**0.72**	**0.64**	**0.74**	**0.68**	**0.92**	**0.93**

TABLE VII
OUR PROPOSED MVNN PERFORMANCE WITH PROMISE

Source	Target	TPTL AUC	TPTL F1-Score	DA-KTSVM AUC	DA-KTSVM F1-Score	GB-CPDP AUC	GB-CPDP F1-Score	Ours AUC	Ours F1-Score
ant-1.6	poi-3.0	0.52	0.35	0.38	0.32	0.57	0.38	0.78	0.77
camel-1.4	ant-1.6	0.54	0.58	0.66	0.46	0.53	0.42	0.86	0.89
camel-1.4	jedit_4.1	0.33	0.40	0.44	0.40	0.47	0.36	0.85	0.90
jedit-4.1	camel-1.4	0.27	0.45	0.36	0.50	0.56	0.50	0.71	0.81
jedit-4.1	xalan-2.4	0.43	0.33	0.56	0.39	0.67	0.44	0.75	0.80
lucene-2.2	ant-1.6	0.41	0.38	0.57	0.54	0.66	0.67	0.68	0.65
synapse-1.2	poi-2.5	0.49	0.46	0.50	0.53	0.59	0.63	0.62	0.57
synapse-1.2	xerces-1.2	0.49	0.43	0.56	0.54	0.68	0.47	0.75	0.84
xerces-1.3	poi-2.5	0.59	0.35	0.48	0.54	0.57	0.54	0.67	0.70
xerces-1.3	synapse-1.1	0.49	0.54	0.47	0.33	0.50	0.47	0.80	0.85
Average		**0.46**	**0.43**	**0.50**	**0.45**	**0.58**	**0.49**	**0.75**	**0.78**

challenging field in the future to improve software engineering, by finding an efficient tools to find and predict defect during the life cycle of software development. In this method, we used multiple steps of pre-processing prior of training ranging from concatenation of features, imputation, distribution normalization and scaling to overcome the different software metrics, and to handle the different distribution of each dataset. Although our proposed method was meant to concatenate more than one dataset, but it still adapts with only a single dataset and still give a satisfactory results compare with previous CDPD methods. Besides the original dataset as the primary view, we opted to create a different view from the dataset by utilizing high dimensionality embedding based on Random Tree, where given enough trees the model will fit the dataset eventually. The use of Isomap as a dimensionality reduction will reduce the size of the MVNN input, so a smaller yet effective NN classifier can be trained using stratified CV. Empirical studies with some notably SDP datasets show the effectiveness of our proposed method compared with previous methods on CPDP. In the future, we would like to extend our research to HDP incorporating other methods for pre-processing and align towards more deep layers NN or using attention mechanism to fully exploit the high dimensionality of the Random Tree output to further improve the performance our proposed MVNN.

AUTHORS CONTRIBUTION

Boy Setiawan: Conceptualization, Methodology, Research, Investigation, Formal Analysis, Resources, Software, Visualization, Original Draft Writing.

Agus Subekti: Supervision, Validation, Original Draft Writing Preparation, Review Writing & Editing.

COPYRIGHT

REFERENCES

[1] S. L. Ahmed Abdu Zhengjun Zhai, Hakim A. Abdo, Redhwan Algabri, "Graph-Based Feature Learning for Cross-Project Software Defect Prediction," Comput. Mater. Contin., vol. 77, no. 1, pp. 161–180, 2023, doi: 10.32604/cmc.2023.043680.

[2] M. A. Elsabagh, M. S. Farhan, and M. G. Gafar, "Cross-projects software defect prediction using spotted hyena optimizer algorithm," SN Appl. Sci., vol. 2, no. 4, p. 538, Mar. 2020, doi: 10.1007/s42452-020-2320-4.

[3] X.-Y. Jing, H. Chen, and B. Xu, Intelligent Software Defect Prediction. 2023. doi: 10.1007/978-981-99-2842-2.

[4] Z. Yuan, X. Chen, Z. Cui, and Y. Mu, "ALTRA: Cross-Project Software Defect Prediction via Active Learning and Tradaboost," IEEE Access, vol. 8, pp. 30037–30049, 2020, doi: 10.1109/ACCESS.2020.2972644.

[5] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: a large scale experiment on data vs. domain vs. process," in Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, in ESEC/FSE '09. New York, NY, USA: Association for Computing Machinery, 2009, pp. 91–100. doi: 10.1145/1595696.1595713.

[6] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," Inf. Softw. Technol., vol. 54, no. 3, pp. 248–256, 2012, doi: https://doi.org/10.1016/j.infsof.2011.09.007.

[7] K. Javed, R. Shengbing, M. Asim, and M. A. Wani, "Cross-Project Defect Prediction Based on Domain Adaptation and LSTM Optimization," Algorithms, vol. 17, no. 5, 2024, doi: 10.3390/a17050175.

[8] E. O. Kiyak, D. Birant, and K. U. Birant, "Multi-view learning for software defect prediction," E-Inform. Softw. Eng. J., vol. 15, no. 1, pp. 163–184, Nov. 2021, doi: 10.37190/e-Inf210108.
[9] J. Zhao, X. Xie, X. Xu, and S. Sun, "Multi-view learning overview: Recent progress and new challenges," Inf. Fusion, vol. 38, pp. 43–54, Nov. 2017, doi: 10.1016/j.inffus.2017.02.007.
[10] X. Xie and S. Sun, "Multi-view twin support vector machines," Intell Data Anal, vol. 19, no. 4, pp. 701–712, Jul. 2015, doi: 10.3233/IDA-150740.
[11] Y. Makihara, A. Mansur, D. Muramatsu, M. Uddin, and Y. Yagi, "Multi-view discriminant analysis with tensor representation and its application to cross-view gait recognition," Jul. 2015, doi: 10.1109/FG.2015.7163131.
[12] X. Yan, S. Hu, Y. Mao, Y. Ye, and H. Yu, "Deep multi-view learning methods: A review," Neurocomputing, vol. 448, pp. 106–129, Aug. 2021, doi: 10.1016/j.neucom.2021.03.090.
[13] T. Baltrusaitis, C. Ahuja, and L.-P. Morency, "Multimodal Machine Learning: A Survey and Taxonomy," IEEE Trans Pattern Anal Mach Intell, vol. 41, no. 2, pp. 423–443, Feb. 2019, doi: 10.1109/TPAMI.2018.2798607.
[14] S. K. D'mello and J. Kory, "A Review and Meta-Analysis of Multimodal Affect Detection Systems," ACM Comput Surv, vol. 47, no. 3, Feb. 2015, doi: 10.1145/2682899.
[15] D. Wang, P. Cui, M. Ou, and W. Zhu, "Deep multimodal hashing with orthogonal regularization.," in IJCAI, 2015, pp. 2291–2297.
[16] B. De Amorim, C. Lucas, G. D.C., and M. O. R. Cruz, "The choice of scaling technique matters for classification performance," 2022, [Online]. Available: https://arxiv.org/pdf/2212.12343
[17] S. Wang, T. Liu, J. Nam, and L. Tan, "Deep Semantic Feature Learning for Software Defect Prediction," IEEE Trans. Softw. Eng., vol. 46, no. 12, pp. 1267–1293, Dec. 2020, doi: 10.1109/TSE.2018.2877612.
[18] Y. Zhao, Y. Wang, D. Zhang, and Y. Gong, "Eliminating the high false-positive rate in defect prediction through BayesNet with adjustable weight," Expert Syst., vol. 39, no. 6, p. e12977, Jul. 2022, doi: 10.1111/exsy.12977.
[19] A. C. Atkinson, M. Riani, and A. Corbellini, "The Box–Cox Transformation: Review and Extensions," Stat. Sci., vol. 36, no. 2, pp. 239–255, 2021, doi: 10.1214/20-STS778.
[20] F. J. O. Gómez, G. O. López, E. Filatovas, O. Kurasova, and G. E. M. Garzón, "Hyperspectral Image Classification Using Isomap with SMACOF," Informatica, vol. 30, no. 2, pp. 349–365, 2019, doi: 10.15388/Informatica.2019.209.
[21] T. Qu and Z. Cai, "An improved Isomap method for manifold learning," Int. J. Intell. Comput. Cybern., vol. 10, pp. 30–40, Mar. 2017, doi: 10.1108/IJICC-03-2016-0014.
[22] M. Yousaf, T. U. Rehman, and L. Jing, "An Extended Isomap Approach for Nonlinear Dimension Reduction," SN Comput. Sci., vol. 1, no. 3, p. 160, May 2020, doi: 10.1007/s42979-020-00179-y.
[23] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," J. Artif. Intell. Res., vol. 16, pp. 321–357, Jun. 2002, doi: 10.1613/jair.953.
[24] Y. Sun, J. Li, Y. Xu, T. Zhang, and X. Wang, "Deep learning versus conventional methods for missing data imputation: A review and comparative study," Expert Syst. Appl., vol. 227, p. 120201, 2023, doi: https://doi.org/10.1016/j.eswa.2023.120201.
[25] Abdul Fadlil, Herman, and Dikky Praseptian M, "K Nearest Neighbor Imputation Performance on Missing Value Data Graduate User Satisfaction," J. RESTI Rekayasa Sist. Dan Teknol. Inf., vol. 6, no. 4, Aug. 2022, doi: 10.29207/resti.v6i4.4173.
[26] B. Sudakov and J. Vondrák, "A randomized embedding algorithm for trees," Combinatorica, vol. 30, no. 4, pp. 445–470, Jul. 2010, doi: 10.1007/s00493-010-2422-5.
[27] H. Alsghaier and M. Akour, "Software fault prediction using particle swarm algorithm with genetic algorithm and support vector machine classifier," Softw. Pract. Exp., vol. 50, no. 4, pp. 407–427, 2020, doi: https://doi.org/10.1002/spe.2784.
[28] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data Quality: Some Comments on the NASA Software Defect Datasets," IEEE Trans. Softw. Eng., vol. 39, no. 9, pp. 1208–1215, 2013, doi: 10.1109/TSE.2013.11.
[29] J. Pachouly, S. Ahirrao, K. Kotecha, G. Selvachandran, and A. Abraham, "A systematic literature review on software defect prediction using artificial intelligence: Datasets, Data Validation Methods, Approaches, and Tools," Eng. Appl. Artif. Intell., vol. 111, p. 104773, 2022, doi: https://doi.org/10.1016/j.engappai.2022.104773.
[30] Y. JIAN, X. YU, Z. XU, and Z. MA, "A Hybrid Feature Selection Method for Software Fault Prediction," IEICE Trans. Inf. Syst., vol. E102.D, no. 10, pp. 1966–1975, 2019, doi: 10.1587/transinf.2019EDP7033.
[31] M. Canaparo, E. Ronchieri, and G. Bertaccini, "Software defect prediction: A study on software metrics using statistical and machine learning methods," PoS, vol. ISGC2022, p. 020, 2022, doi: 10.22323/1.415.0020.
[32] O. P. Omondiagbe, S. A. Licorish, and S. G. MacDonell, "Improving Transfer Learning for Cross Project Defect Prediction," 2022, doi: 10.36227/techrxiv.19517029.v2.
[33] A. Wang, L. Yang, H. Wu, and Y. Iwahori, "Heterogeneous Defect Prediction Based on Federated Prototype Learning," IEEE Access, vol. 11, pp. 98618–98632, 2023, doi: 10.1109/ACCESS.2023.3313001.
[34] M. Jureczko and L. Madeyski, Towards identifying software project clusters with regard to defect prediction, vol. 9. 2010. doi: 10.1145/1868328.1868342.
[35] L. Gong, S. Jiang, Q. Yu, and L. Jiang, "Unsupervised Deep Domain Adaptation for Heterogeneous Defect Prediction," Ieice Trans. Inf. Syst., vol. E102.D, no. 3, pp. 537–549, 2019, doi: 10.1587/transinf.2018edp7289.
[36] J. Nam and S. Kim, "Heterogeneous defect prediction," in Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, in ESEC/FSE 2015. New York, NY, USA: Association for Computing Machinery, 2015, pp. 508–519. doi: 10.1145/2786805.2786814.
[37] S. Widodo, H. Brawijaya, and S. Samudi, "Stratified k-fold cross validation optimization on machine learning for prediction.," Sinkron, 2022, doi: 10.33395/sinkron.v7i4.11792.
[38] Y. Zhao, Y. Zhu, Q. Yu, and X. Chen, "Cross-Project Defect Prediction Considering Multiple Data Distribution Simultaneously," Symmetry, vol. 14, no. 2, 2022, doi: 10.3390/sym14020401.
[39] Y. Xing, W. Lin, X. Lin, B. Yang, and Z. Tan, "Cross-Project Defect Prediction Based on Two-Phase Feature Importance Amplification.," Comput. Intell. Neurosci., vol. 2022, p. 2320447, 2022, doi: 10.1155/2022/2320447.
[40] C. Liu, D. Yang, X. Xia, M. Yan, and X. Zhang, "A two-phase transfer learning model for cross-project defect prediction," Inf. Softw. Technol., vol. 107, pp. 125–136, 2019, doi: https://doi.org/10.1016/j.infsof.2018.11.005.
[41] C. Jin, "Cross-project software defect prediction based on domain adaptation learning and optimization," Expert Syst. Appl., vol. 171, p. 114637, 2021, doi: https://doi.org/10.1016/j.eswa.2021.114637.
[42] J. Wang, Y. Chen, S. Hao, F. Wenjie, and Z. Shen, Balanced Distribution Adaptation for Transfer Learning. 2017, p. 1134. doi: 10.1109/ICDM.2017.150.
[43] Y. L. Alemu, T. Lahmer, and C. Walther, "Damage Detection with Data-Driven Machine Learning Models on an Experimental Structure," Eng, vol. 5, no. 2, pp. 629–656, 2024, doi: 10.3390/eng5020036.