Hindawi
Mobile Information Systems
Volume 2022, Article ID 2522202, 11 pages
https://doi.org/10.1155/2022/2522202

Research Article

Software Defect Prediction and Analysis Using Enhanced Random Forest (extRF) Technique: A Business Process Management and Improvement Concept in IOT-Based Application Processing Environment

Fahad H. Alshammari

College of Computing and Information Technology, Shaqra University, Shaqra, Saudi Arabia

Correspondence should be addressed to Fahad H. Alshammari; fahad.h@su.edu.sa

Received 13 July 2022; Revised 6 August 2022; Accepted 11 August 2022; Published 7 September 2022

Academic Editor: Sikandar Ali

Software defect prediction is a thriving study area in the realm of software engineering and processing in the IOT-based environment. Defect prediction creates a list of defective source code artifacts so that quality assurance companies may successfully assign limited methods for certifying programming things by investing more effort into the bad source code. Defect prediction can assist estimate maintenance times, which can help with quality assurance, dependability, security, and cost reduction. Many predictions in IOT-based processing environment and business process management and enhancement challenges still exist in defect expectation ponders, and there are various noteworthy concerns. In addition, it is difficult to apply these methodologies practically because most of the investigations verified in open-source programming ventures with the goal that present forecast models might not work for other programming items including business programming. Investigating security issues in cross-project deformity expectation is required since if we have more accessible restrictive datasets, the assessment of forecast models will be more stable. In general, every defect is essential regarding quality, reliability, security, and cost-effectiveness. Therefore, an enhanced and improved maintenance schedule is required to acknowledge forecasting techniques. Therefore, in this article, we have evaluated different Semi-Supervised Learning (SSL) techniques, among which Extended Random Forest (extRF) technique is one for defective system prediction. The Extended Random Forest (extRF) technique is the extended form of the Random Forest (RF), which is a supervised learning technique into semi-supervised learning getting the hang of refining every arbitrary tree given an individual-training worldview. An enhancing technique is recommended, and a weighted mixture of irregular trees creates the final forecast results.

1. Introduction

Defect prediction of software is a vibrant research domain in the field of software engineering environment [1]. the Defect prediction outcomes of the defective source code antiquities with the goal that quality confirmation in anorganizations can successfully assign restricted means for approving programming items by putting additional exertion on the poor source code [2]. The term defect usually mentions to some problem with the software, either with its exterior performance or with its interior characteristics. A software defect is an error, flaw, failure, or fault in a computer program or system that sources it to crop an improper or unpredicted outcome, or to an act in unintended behaviors [3]. Furthermore, when an individual wants to ensure that, if feasible, such type of remaining defects will create minimal interference or destruction. Most advanced software systems beyond restricted personal utilization have become gradually huge and more complex because of the augmented necessity for automation, functions, characteristics, structures, and services. It is almost impossible to entirely prevent or remove defects in such huge complex systems [4].

As the amount of software produces ends up noticeably bigger, defect forecast systems will assume an essential part to help software engineers and additionally accelerate time to a marketplace with more solid software products. Organizations are capitalizing heavily on the operational environment and organizational applications. Software defects in the operational environment are defined as unpredicted interruptions which affect the system productivity and have also an impact on the cost [5]. To minimize the unscheduled interruptions and increase the performance, many defect prediction management techniques are introduced. IT service providers are constantly seeking more efficient procedures and methods to improve the effectiveness and superiority of the process. IT Substructure Library is the most common framework for IT services due to its best management guidelines [6]. It provides the best guidelines on how to manage, develop, and maintain the IT substructure. In addition to this, it also gives guidelines on improving the quality of the IT substructure. Software defects on software systems propose that most of the defects occur during the system up-gradation, during the maintenance task, and maybe sometimes due to the system integration. There are many causes of the defects of software systems [7]. Defects in software systems during operation are unavoidable. That results in the unavailability of the system which results in cost and dissatisfied customers and clients. These defects need to be reduced and removed for cost-effectiveness and the satisfaction of the users. Most common and agreed causes are insufficient testing or poor testing, flaws in documentation or a poor understanding of the system complexity, system overload, resource exhaustion, and complex defect detection routines [8, 9].

The major reasons that cause the systems to be defective are essentially the complex structure and inter-dependencies of the components. A defect or even a partial defect of one system can cause other systems that depend on it to malfunction. This problem can create a chain of system failures that propagates until it reaches critical components and causes the system to flop. We first discuss the software defects and what types of defects can be faced using the software. With the extensive use of software systems in the current society, the dissentient influence of software defects is also expanding [10]. Different quality assurance (QA) alternatives and interrelated techniques can be utilized in a concerted struggle to efficiently and successfully guarantee their quality. Testing is most of the maximum typically performed quality assurance actions for software. It predicts execution troubles so that underlying reasons may be recognized and fixed. Inspection, on the other hand, directly prevents and modifies software troubles without resorting to implementation. Other (QA) replacements such as official verification, defect prevention, and fault tolerance, address within their approaches. Close inspection of how excellent QA replacements cope with defects can assist one in improved utilization of them for unique applications [11–13].

We can perceive that failures, faults, and errors are diverse features of defects. The wrong algorithm is smeared in numerous modules and becomes the source of many defects (faults or bugs), and a single fault can cause various failures in repetitive executions [14]. A particular error can create different faults, such as when an incorrect algorithm is trying in several components and that becomes the source of several defects (faults or bugs) and a single fault can create numerous failures in repetitive modules [15]. Conversely, a similar defect can be caused by many faults, such as an interface or interaction failure including several modules, and a similar fault can be there because of various errors. Programming deformity forecast purposes to intuitively perceive problematic programming modules for effective programming tests popular to expand the brilliance of a product framework in the current period of data innovation [16]. Current defect prediction techniques do not perform well for a complex software system. Predicting defects in complex software is no easy task as it is hard to understand and maintain. Many prediction techniques flop in these systems, and performance of the most of them is compromised [10]. Limitations of machine learning moved the trend toward supervised learning and unsupervised learning [17]. But the fact is that supervised learning has some restrictions to be considered. Such supervised learning requires the labeled datasets which are in the historical form, which is costly and time-consuming. Gathering historical datasets manually, automatic engineering is the waste of time and money [18, 19]. To overcome the problems and restrictions of the machine learning methods, an alternative approach "semi-supervised" is explored in [20]. Semi-supervised learning is a special case of the machine learning methods, but cannot be completely considered under the umbrella of supervised learning [21]. Semi-supervised learning is a broad concept and has several functions in it to minimize the problems that the previous approaches to machine learning cannot overcome. Semi-supervised learning has reduced the effort of gathering a large amount of historical data (as required in supervised learning) and has also made it possible to choose only a few instances actively from the large pool of the unlabeled data to be labeled. This research is the combination of the semi-supervised learning approach with the supervised learning classifier "Support vector machine, random forest, STDDL." Two best approaches are combined to predict the failure incidents of the system software [22].

Investigating security issues in cross-project deformity expectation is required since if we have more accessible restrictive datasets, the assessment of forecast models will be more stable. In general, every defect is essential regarding quality, reliability, security, and cost-effectiveness. Therefore, an enhanced and improved maintenance schedule should have to acknowledge with forecasting techniques. Therefore, in this study, we evaluated different semi-supervised Learning (SSL) methodologies in which the extRF technique is one for defective system prediction. The extRF is the extended form of the Random Forest, which is a supervised learning approach to semi-supervised learning getting the hang of, refining every arbitrary tree given an individual-training worldview.

An enhanced and improved maintenance schedule is required to acknowledge forecasting techniques. Therefore, in this article, we have evaluated different Semi-Supervised

Learning (SSL) techniques, among which Extended Random Forest (extRF) technique is one for defective system prediction. The Extended Random Forest (extRF) technique is the extended form of the Random Forest (RF), which is a supervised learning technique into semi-supervised learning getting the hang of refining every arbitrary tree given an individual-training worldview. An enhancing technique is recommended, and a weighted mixture of irregular trees creates the final forecast results.

The current section is about the detailed introduction of domain knowledge, while the rest of the paper is organized as follows: Section 2 is the literature review, Section 3 is research methodology, Section 4 is result and analysis, and Section 5 is the conclusion and future work.

2. Literature Review

Supervised learning classification algorithms in machine learning (ML) can be used to construct the prediction model with preceding software features and preceding defect labels. However, occasionally we cannot have sufficient defect data to construct accurate models. For instance, few project partners may not gather defect data for some project constituents or the implementation cost of features gathering tools on the entire system may be highly costly. In these conditions, they require strong or dominant classifiers which can construct precise classification models with restricted defect data or dominant semi-supervised classification procedures which can advantage from unlabeled information combined with labeled one. This research issue can be termed software defect prediction with restricted defect data [20].

According to [23], the Naive Bayes algorithm is the greatest selection to create a semi-supervised defect forecasting model for minor datasets and YATSI (yet another two-stage idea) algorithm may provide better performance of Naive Bayes for huge datasets [24]. Dahiya and Srivastava [25] compared four different semi-supervised cataloging methods for the prediction of defects which exist in the software comprising Low-Density Separation (LDS), Expectation-Maximization (EMSEMI), Support Vector Machine (SVM), and Class Mass Normalization (CMN) approaches. They presented that the LDS algorithm is superior to SVM when the dataset is in a huge amount, and LDS-centered prediction technique is recommended for the prediction of defects from software especially when the defected information is limited. Sindhwani et al. [26] introduce a Semi Supervised Learning (SSL) kernel that is not restricted to the unlabeled data but describes overall input space. The kernel thus helps with induction. The kernel is a novel explanation of the multiple regularization frameworks. Preliminary from a base kernel K describe over the entire input space (e.g., linear kernels, RBF kernels), the writers adjust the RKHS by keeping the similar function space but altering the standard [27, 28]. The consistency of graph-centered Semi-Supervised Learning (SSL) algorithms is an exposed research space. Consistency means whether cataloging comes together to the best result as the amount of labeled and unlabeled information increases to limitlessness [29, 30]. Newly von Lux burg et al. [31] learn the constancy of spectral clustering approaches. The authors identify that the standardized Laplace is improved than the non-normalized for spectral clustering [32].

The [33] claimed that a generalization fault assured for Semi Supervised Learning (SSL) algorithms with manifold learners, in addition to co-training. The investigator demonstrates if several learning algorithms are enforced to create the same theories (i.e., to decide) assuming a similar training set, and such suppositions still have less training fault, then the generalization fault bound is tighter. The unlabeled information is utilized to evaluate the contract among suppositions. The author suggests a new Agreement Boost procedure to implement the process. The generative model named Hidden Markov Model (HMM) for the semi-supervised sequence-learning algorithm proposed by is an example, claimed specifically the Baum-Welsh Hidden Markov Model training procedure [34, 35]. It is extremely important for the order of versions of the Expectation-Maximization (EM) procedure on mixture models. Another study [36] offered a review of related studies to evaluate the software metrics for the defect forecast. The researchers claimed that the OO metrics at 49 percent of the extreme usage, followed by the prior process metrics at about 24% and source code features at 27% are used highly for the prediction of software defects [10, 37]. They decided that it is beneficial to use the OO process metrics for defect prediction to evaluate traditional scope or complexity metrics. Moreover, they amend that these metrics produced significantly improved outcomes in predicting post-delivery defects compared to the static code features.

Radjenovic et al. prolonged Kitchenham's evaluation work (Kitchenham 2010) and evaluated the implementation of software features or metrics for defect prediction [36]. However, they did not intend to incorporate other features of the software defect prediction that can be affecting the implementation of software metrics [38]. Recently, a study that facilitate an extensive history and overview of the defect prediction of the software and also about its components is presented by Kamei and Shihab (2016) [39]. The study of Kamei and Shihab mainly emphasized activities accomplishment done in software defect prediction as well as argued on the present trends in relating fields. Additionally, some of the future challenges for software fault prediction have been identified and discussed. However, the study of Kamei and Shihab did not deliver information on several works on software defect prediction in terms of semi-supervised learning [40]. Semi-supervised learning techniques are based on expectation maximization, clustering, and graph-based dictionary techniques as well as different techniques of sampling which makes the problem easy to tackle [41]. Although this survey is done by other people in the IT field, my work has little resemblance with [42, 43] but is different from others as my focus is just on those techniques which are based on semi-supervised learning which is a very useful technique for little labeled and a large amount of unlabeled data. As it is costly to get the labeled data so, Semi-Supervised Learning (SSL) techniques are very helpful in this domain. As per the knowledge of the author, all survey papers are categorized collectively into Semi-

Supervised Learning (SSL) and Supervised Learning (SL) techniques, but core focus of this article is just on those techniques which are based on pure semi-supervised learning. Researchers have worked collectively on the Semi Supervised Learning (SSL) and Supervised Learning (SL) techniques [44, 45] but due to thecollective learning, they did not focused spicificaly on Semi Supervised Learning (SSL) techniques categorizations. This study covers all those techniques that are exactly based on semi-supervised learning.

3. Research Methodology

This section provides detailed information on the materials and the methods used in different Semi-Supervised Learning (SSL). The methodology adopted to perform the prediction task of the software defect is discussed in detail in the following sections.

3.1. Sample-Based Software Defect Prediction. This section describes the suggested sample-based bug/defect forecast technique. This technique can be categorized into three methods: sampling with conventional Machine Learning (ML), sampling with Semi-Supervised Learning (SSL), and sampling with active Semi-Supervised Learning (SSL) by Ming Li et al. [46].

Normally, software defect forecast techniques rely on the prior information of software. But the problem is that recently developed software has no prior information to be based on for defect forecast which is the cause that conventional techniques do not support. A novel sample-based bug or defect forecast technique performs better in this case [43, 47].

3.2. Method for SSL Technique of SDP. Despite taking entire components of huge software, a sample of them is taken for check, and afterward, a model is created to forecast the defect of residual components of the software. In these outlines, this study describes the suggested sample-based bug or defect forecast technique. This technique can be categorized into three methods: sampling with conventional ML, sampling with SSL, and sampling with active Semi-Supervised Learning (SSL) [48].

3.2.1. Sampling with the Conventional ML Method. Software defect forecast, which leads to forecast whether a specific software component comprises any defect, can be troupe into a categorization issue in machine learning, where metrics of the software are mined from each software component to build an example with manually allocated labels faulty (having one or more bugs or defects) and defective-free (no any defects). Such types of training instances are then utilized to learn the classifier which afterward is utilized in forecasting the defective and non-defective status of unidentified software components. Sample-based bug or defect forecast technique does not base on the assumption that the recent project has a similar bug or defect features as the prior projects [49]. The predictable machine learners (e.g., Logistic Regression, Decision Tree, Naive Bayes, etc.) can be smeared to the categorization. Sample-based software defect prediction Sampling with conventional ML Sampling with SSL [50]. Semi-Supervised Learning (SSL). The CoForest method Sampling with active Semi Supervised Learning method 0 Advanced software organizations often comprise hundreds or even thousands of components. An organization is generally not able to have enough money for extensive testing for all components particularly when time and resources are inadequate [44, 51].

3.2.2. Sampling with SSL: The CoForest Method. To enhance the working of the sample-based bug forecast, Semi-Supervised Learning (SSL) for classifier creation is implemented, which initially learns preliminary classifier from a minor sample of labeled trained set and improves it further by manipulating a huge number of existing unlabeled information. In Semi-Supervised Learning (SSL), an effective model is recognized as disagreement-based Semi-Supervised Learning (SSL), where numerous learners are experts for similar chores, and the disagreements among the learners are exploited throughout learning. In this model, unlabeled information can be considered as an exceptional information interchange "platform" [52]. In this technique, the active method CoForest is implemented for defect forecast. Its performance is based on a well-recognized ensemble learning procedure called Random Forest to control the issues of influential the highly confident instances to label and generate the final assumption [53].

3.2.3. Sampling with Active SSL: The ACoForest Method. Although a random example can be utilized to estimate the characteristics of entire the software components in the present projects, a random tester is seemingly not data-effective since a random taster neglects the "necessities" of the learners for attaining better working and hence may comprise redundant material that the learner has previously apprehended during the learning procedure. Instinctively, if a learner is an expert information that is required most for refining its working, it may need less labeled information than the learner's expert without concerning its necessities for learning; put it an alternative way, if a similar amount of labeled information is utilized, the learner that expertly using the labeled information, it needs further improved working than the expert learner without concerning its necessities for learning. According to [54], active learning, which is an exceptional main approach for learning in the manifestation of a huge number of unlabeled information, goals to attain better working by learning with as little labeled information as possible.

4. Results and Analysis

In this section, we have evaluated different techniques analytically and suggested a novel approach based on the results of different techniques of software defect prediction. SSL tends to this issue by utilizing an extensive measure of unlabeled information, together with the marked

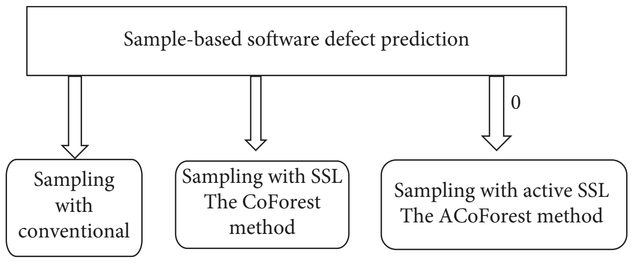

Sample-based software defect prediction
0
Sampling with conventional
Sampling with SSL The CoForest method
Sampling with active SSL The ACoForest method

Figure 1: Sample-based software defect prediction.

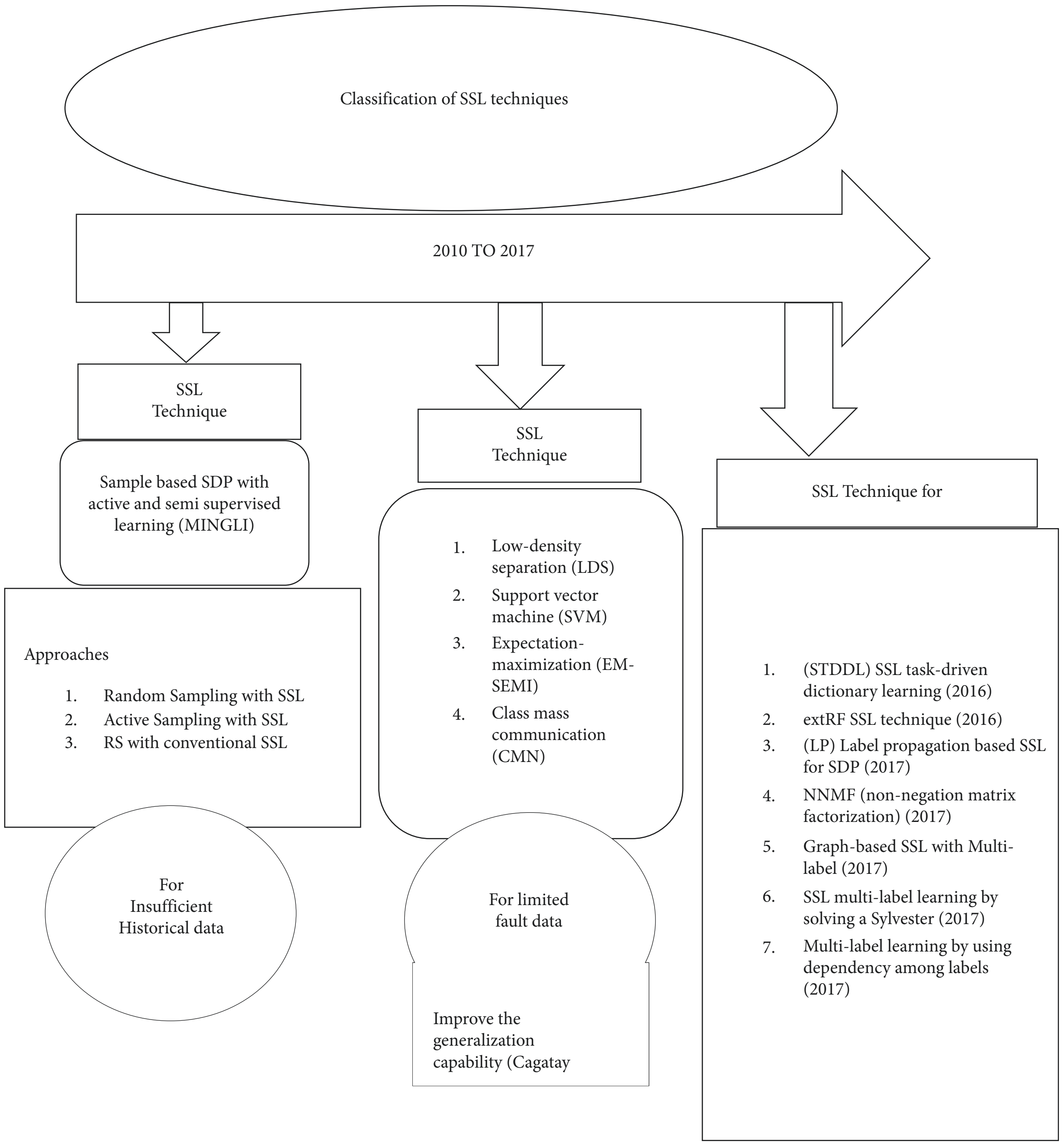
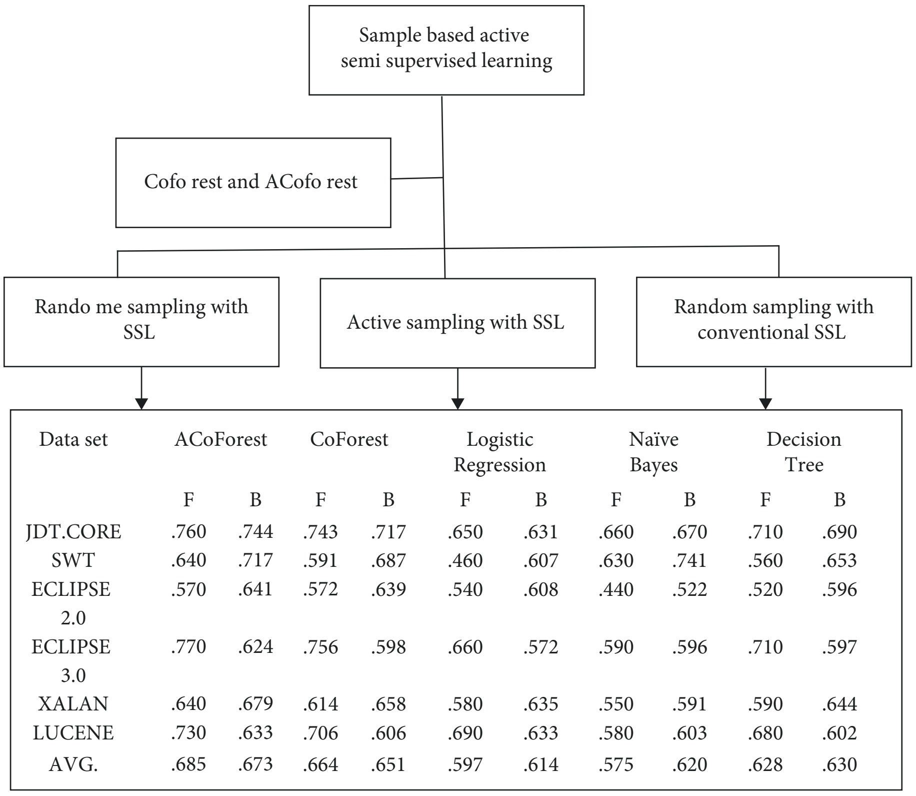

Classification of SSL techniques
2010 TO 2017
SSL Technique
Sample based SDP with active and semi supervised learning (MINGLI)
Approaches
1. Random Sampling with SSL
2. Active Sampling with SSL
3. RS with conventional SSL
For Insufficient Historical data
SSL Technique
1. Low-density separation (LDS)
2. Support vector machine (SVM)
3. Expectation-maximization (EM-SEMI)
4. Class mass communication (CMN)
For limited fault data
Improve the generalization capability (Cagatay
SSL Technique for
1. (STDDL) SSL task-driven dictionary learning (2016)
2. extRF SSL technique (2016)
3. (LP) Label propagation based SSL for SDP (2017)
4. NNMF (non-negation matrix factorization) (2017)
5. Graph-based SSL with Multi-label (2017)
6. SSL multi-label learning by solving a Sylvester (2017)
7. Multi-label learning by using dependency among labels (2017)

Figure 2: Representation of different SSL techniques.


Sample based active semi supervised learning
Cofo rest and ACofo rest
Rando me sampling with SSL
Active sampling with SSL
Random sampling with conventional SSL

Data set	ACoForest		CoForest		Logistic Regression		Naïve Bayes		Decision Tree	
	F	B	F	B	F	B	F	B	F	B
JDT.CORE	.760	.744	.743	.717	.650	.631	.660	.670	.710	.690
SWT	.640	.717	.591	.687	.460	.607	.630	.741	.560	.653
ECLIPSE 2.0	.570	.641	.572	.639	.540	.608	.440	.522	.520	.596
ECLIPSE 3.0	.770	.624	.756	.598	.660	.572	.590	.596	.710	.597
XALAN	.640	.679	.614	.658	.580	.635	.550	.591	.590	.644
LUCENE	.730	.633	.706	.606	.690	.633	.580	.603	.680	.602
AVG.	.685	.673	.664	.651	.597	.614	.575	.620	.628	.630

FIGURE 3: Description of the sample-based active and semi-supervised learning.

information, to assemble better classifiers. Since SSL requires less human effort and gives higher precision, it is of a mind-blowing premium both in principle and down-to-earth terms. Figure 1 demonstrates the hierarchy of the diverse prediction techniques in terms of semi-supervised learning examinations.

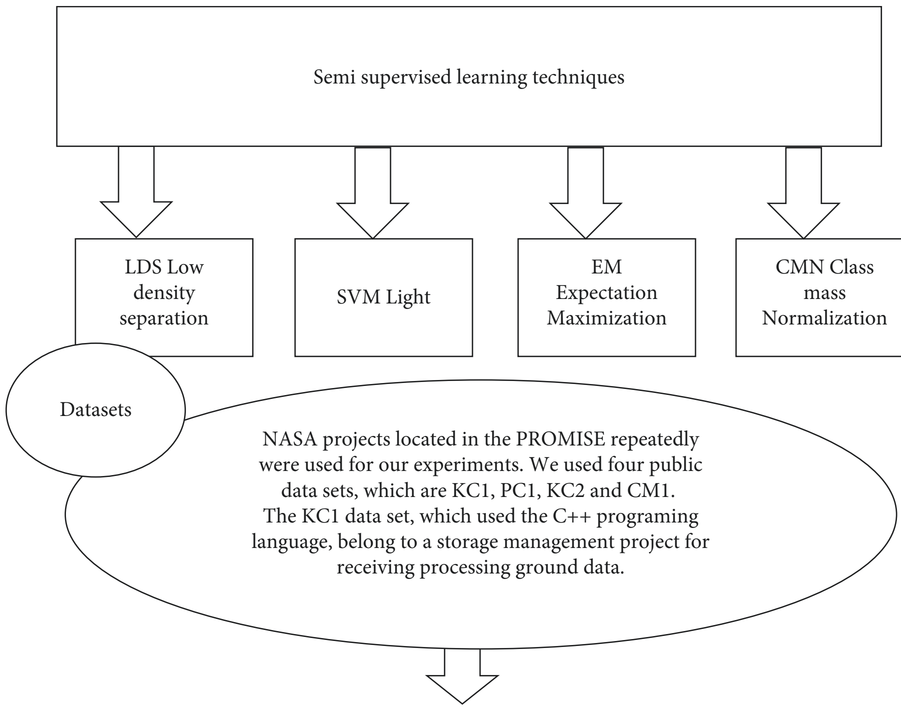
Figure 2 represents the different Semi-Supervised Learning (SSL) techniques for software defect prediction during the last few years. This figure is going to display different semi-supervised techniques in an ordered manner which shows the techniques along with their different approaches diagrammatically.

To assess the viability of test-based imperfection expectation techniques, the authors perform tests utilizing datasets accessible on the PROMISE website. This examination gathered the Eclipse, Lucene, and Xalan datasets. The Eclipse datasets contain 198 traits, including the code and many quality measurements, such as LOC, Cyclomatic Intricacy, number of classes, and also the measurements about random trees i.e.Number of squares, number of if articulations, technique references, and so on. The Eclipse imperfection information was gathered by mining Eclipse's bug databases and adaptation documents. In this examination, the authors explore different possibilities regarding Eclipse 2.0 and 3.0. To demonstrate the all-inclusive statement of the outcomes, we utilize the class-level information for Eclipse 3.0 and the file-level information for Eclipse 2.0. They likewise pick two Eclipse parts: JDT.Core and SWT in Eclipse 3.0 to assess the deformity expectation execution for littler Eclipse ventures. This examination just analyzed the pre-discharge bugs, which announced a half year before of the discharge. The information is compressed in Figure 3.

Figure 4 depicts the diverse SSL approaches alongside their datasets, exploratory outcomes, and assessment of expectation programming.

Exploratory outcomes check the predominant execution of our proposed technique on nine NASA datasets, both quantitatively and subjectively. The trials are directed on the three datasets: JM1(large), KC1(median), and PC1(small). Figure 5 demonstrates the inclinations of execution of a considerable number of strategies at various name rates. We can analyze that the STDDL dependably beats other thought about techniques at various class irregularity rates. At the point when the class conveyance is adjusted, all strategies can accomplish a better execution. With the expansion of the class imbalance rate, the prevalence of STDDL is more dominant. The evaluation of STDDL semi-supervised learning is depicted in Figure 5.

4.1. Evaluation of extRF for Software Defect Prediction. This section presents the detail of the performance assessment of the SSL approach Extended Random Forest (extRF) to abandon expectation. It expands Random Forest into a semi-regulated group picking up, refining every arbitrary tree given self-preparing. A boosting procedure is presented, and the last expectation result is created by a weighted mix of irregular trees. Our trials are led on Eclipse informational index. We concentrate the measurements on two variants (Eclipse 2.0 and 3.0), and two segments of form 3.0 (JDT.Core and SWT). The study point out that the Extended Random Forest (extRF) prepared with a little size of marked dataset accomplishes, similar exactness to that of regulated


Semi supervised learning techniques
LDS Low density separation
SVM Light
EM Expectation Maximization
CMN Class mass Normalization
Datasets
NASA projects located in the PROMISE repeatedly were used for our experiments. We used four public data sets, which are KC1, PC1, KC2 and CM1. The KC1 data set, which used the C++ programing language, belong to a storage management project for receiving processing ground data.

Performance results on CM1

Algorithms	5%	10%	20%
LDS	0.66	0.70	0.73
SVM	0.76	0.76	0.73
SM	0.65	0.67	0.74
EM-SEMI	0.47	0.47	0.53

Performance results on KC2

Algorithms	5%	10%	20%
LDS	0.80	0.81	0.82
SVM	0.84	0.85	0.85
SMM	0.69	0.69	0.69
EM-SEMI	0.47	0.56	0.55

Performance results on KC1

Algorithms	5%	10%	20%
LDS	0.77	0.78	0.79
SVM	0.76	0.75	0.70
SMM	0.73	0.76	0.75
EM-SEMI	0.49	0.48	0.50

Performance results on PC1

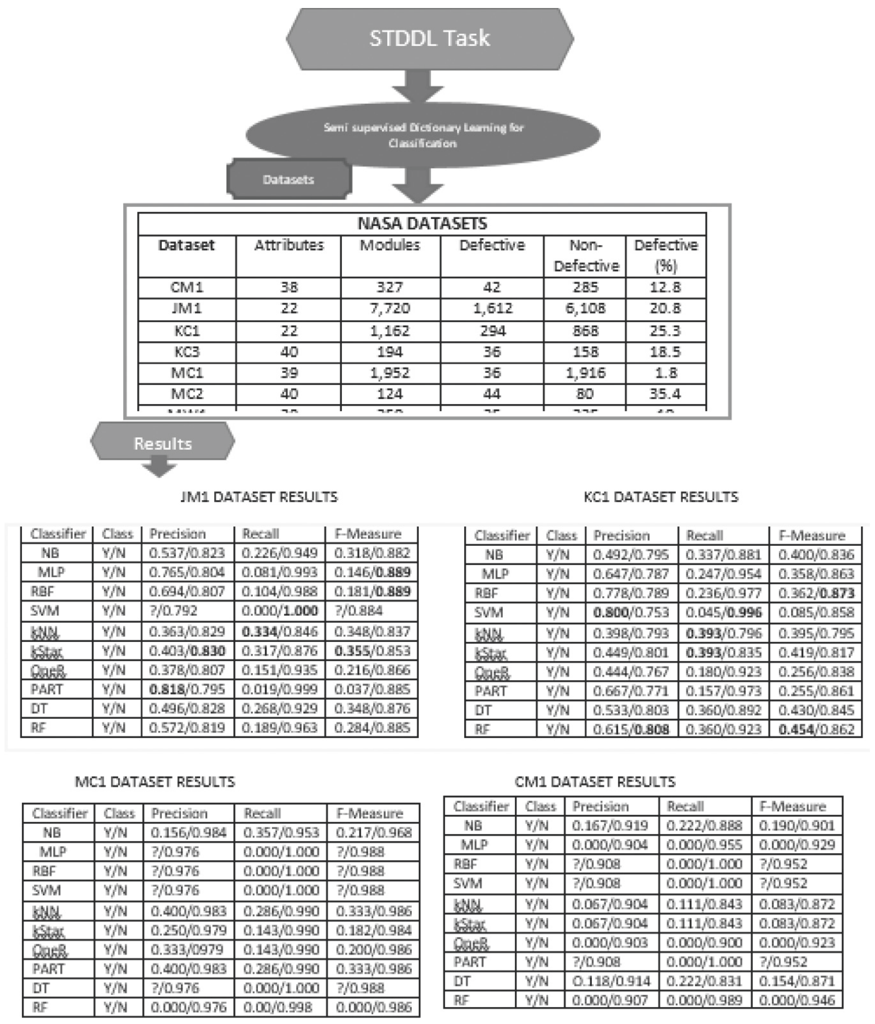
Algorithms	5%	10%	20%
LDS	0.73	0.75	0.75
SVM	0.70	0.73	0.70
SMM	0.63	0.61	0.80
EM-SEMI	0.60	0.57	0.55

Figure 4: Combination of LDS, SVM, EM, and CMN.

approach prepared with a bigger size of named dataset. While utilizing Extended Random Forest (extRF) on change burst measurements, imperfection forecast accomplishes the best execution with an F-measure of 0.75.

4.2. SSL Dimension Reduction Technique and Combination Evaluation of GSSL, NSG, and NSGLP. In contrast with numerous state-of-the-art representative SSL methods of predicting software defects, experimental results on ten NASA datasets present that the suggested NSGLP methodology performs better. This education proposes a new nonnegative sparse graph-based label propagation approach (NSGLP) for SSL in software defect forecast, which usages not only insufficient labeled information but also plentiful unlabeled information to increase the generality proficiency.

A graphical representation of diverse SSL software defect prediction techniques is shown in Figure 6, which displays the percentage of SSL evaluation of SDP. This graph displays the percentage of the Extended Random Forest (extRF) Semi-Supervised Learning (SSL) that is better than other SSL techniques. Therefore, we recommend that the results of the Extended Random Forest (extRF) are a better predicting method used in Semi-Supervised Learning (SSL) for the quality assurance and reliability of the software in the current age of the predicting domain, which also reduces the



NASA DATASETS

Dataset	Attributes	Modules	Defective	Non-Defective	Defective (%)
CM1	38	327	42	285	12.8
JM1	22	7,720	1,612	6,108	20.8
KC1	22	1,162	294	868	25.3
KC3	40	194	36	158	18.5
MC1	39	1,952	36	1,916	1.8
MC2	40	124	44	80	35.4

JM1 DATASET RESULTS

Classifier	Class	Precision	Recall	F-Measure
NB	Y/N	0.537/0.823	0.226/0.949	0.318/0.882
MLP	Y/N	0.765/0.804	0.081/0.993	0.146/**0.889**
RBF	Y/N	0.694/0.807	0.104/0.988	0.181/**0.889**
SVM	Y/N	?/0.792	0.000/**1.000**	?/0.884
kNN	Y/N	0.363/0.829	**0.334**/0.846	0.348/0.837
kStar	Y/N	0.403/**0.830**	0.317/0.876	**0.355**/0.853
OneR	Y/N	0.378/0.807	0.151/0.935	0.216/0.866
PART	Y/N	**0.818**/0.795	0.019/0.999	0.037/0.885
DT	Y/N	0.496/0.828	0.268/0.929	0.348/0.876
RF	Y/N	0.572/0.819	0.189/0.963	0.284/0.885

KC1 DATASET RESULTS

Classifier	Class	Precision	Recall	F-Measure
NB	Y/N	0.492/0.795	0.337/0.881	0.400/0.836
MLP	Y/N	0.647/0.787	0.247/0.954	0.358/0.863
RBF	Y/N	0.778/0.789	0.236/0.977	0.362/**0.873**
SVM	Y/N	**0.800**/0.753	0.045/**0.996**	0.085/0.858
kNN	Y/N	0.398/0.793	**0.393**/0.796	0.395/0.795
kStar	Y/N	0.449/0.801	**0.393**/0.835	0.419/0.817
OneR	Y/N	0.444/0.767	0.180/0.923	0.256/0.838
PART	Y/N	0.667/0.771	0.157/0.973	0.255/0.861
DT	Y/N	0.533/0.803	0.360/0.892	0.430/0.845
RF	Y/N	0.615/**0.808**	0.360/0.923	**0.454**/0.862

MC1 DATASET RESULTS

Classifier	Class	Precision	Recall	F-Measure
NB	Y/N	0.156/0.984	0.357/0.953	0.217/0.968
MLP	Y/N	?/0.976	0.000/1.000	?/0.988
RBF	Y/N	?/0.976	0.000/1.000	?/0.988
SVM	Y/N	?/0.976	0.000/1.000	?/0.988
kNN	Y/N	0.400/0.983	0.286/0.990	0.333/0.986
kStar	Y/N	0.250/0.979	0.143/0.990	0.182/0.984
OneR	Y/N	0.333/0979	0.143/0.990	0.200/0.986
PART	Y/N	0.400/0.983	0.286/0.990	0.333/0.986
DT	Y/N	?/0.976	0.000/1.000	?/0.988
RF	Y/N	0.000/0.976	0.00/0.998	0.000/0.986

CM1 DATASET RESULTS

Classifier	Class	Precision	Recall	F-Measure
NB	Y/N	0.167/0.919	0.222/0.888	0.190/0.901
MLP	Y/N	0.000/0.904	0.000/0.955	0.000/0.929
RBF	Y/N	?/0.908	0.000/1.000	?/0.952
SVM	Y/N	?/0.908	0.000/1.000	?/0.952
kNN	Y/N	0.067/0.904	0.111/0.843	0.083/0.872
kStar	Y/N	0.067/0.904	0.111/0.843	0.083/0.872
OneR	Y/N	0.000/0.903	0.000/0.900	0.000/0.923
PART	Y/N	?/0.908	0.000/1.000	?/0.952
DT	Y/N	O.118/0.914	0.222/0.831	0.154/0.871
RF	Y/N	0.000/0.907	0.000/0.989	0.000/0.946

Figure 5: Evaluation of STDDL semi-supervised learning.

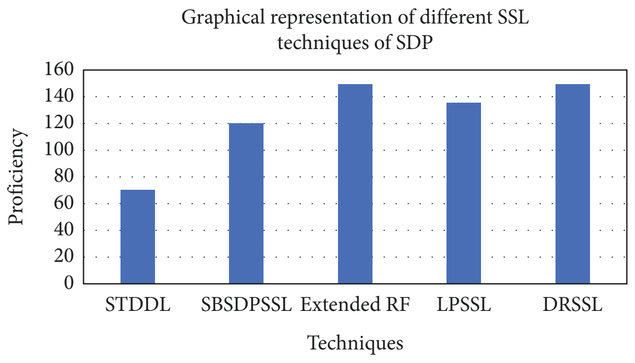


Figure 6: Graphical representation of different SSL techniques of SDP.

cost and time of business environments. Smearing the proposed techniques, our evaluation display that a slighter sample can attain similar defect forecast performance as greater samples do. The model can assist as a primary labeled drill set that symbolizes the primary data dispersal of the whole dataset. If there is insufficient prior information about datasets for developing an effective bug prediction prototype, for a new venture we can select randomly models which have a small percentage of constitutes to test, for this purpose have to attain their defect status (defect prone or defect-free), and then utilize the selected sample for the developing purpose of defect prediction for this project. Our evaluation also presents that in common, sampling with Semi-Supervised Learning (SSL) and active learning can attain improved prediction presentation than sampling with predictable ML techniques. A sample might comprise abundant information that a predictable ML learner has already educated very well but might comprise minor information that the learner requires for increasing the present prediction accurateness.

4.3. Future Challenges. There are still many prediction problems with defect expectation ponders. Even though there have been many noteworthy investigations, it is challenging to employ those approaches in practice for the following reasons: With the understanding that the existing prediction models might not be applicable to other types of programming, including business programming, the majority of study was confirmed in open-source programming projects. Since forecast model evaluation will be more stable, if we have more easily accessible limited datasets it is vital to reexamine security considerations in cross-project deformity expectation. Additionally, the cross forecast continues to be a particularly difficult challenge in missing expectations from two angles. As programming projects expand, file-level imperfection predictions may not be sufficient in terms of cost sustainability. There are not many studies on finer expectation granularity yet. Attention must be paid to finer-grained deformity forecasting, such as change categorization and line-level imperfection expectation. It's possible that the defect forecast measurements and models put forward up to this point do not always guarantee excellent expectation execution.

New categories of improvement process data that are never used for imperfection expectation measurements or models can be removed from programming archives as they develop. The study of new measures and models should continue.

5. Conclusion and Future Work

Generally, each software defect is essential regarding quality, reliability, security, and cost-effectiveness. Defect prediction help in predicting the maintenance times, which counteract quality assurance, reliability, security richness, and reduce costs. This study evaluated and analyzed different SSL methodologies in which the Extended Random Forest (extRF) technique is used for the defective system prediction. The Extended Random Forest (extRF) technique is an extended form of the Random Forest approach, which is a supervised learning approach to semi-supervised learning getting the hang of refining every arbitrary tree given an individual-training worldview. A boosting procedure is conferred, and a weighted mixture of irregular trees creates the final forecast results. After analyzing the experimental results of this study, we can conclude that sampling with Semi-Supervised Learning (SSL) and active learning can attain improved prediction presentation than sampling with predictable ML techniques. A sample might comprise abundant information that a predictable ML learner has already educated very well; however, it might comprise minor information that the learner requires for increasing the present prediction accurateness. In future work, this study can be extended to incorporate the research on the legality of our evaluation and its comparison with the other proposed models for defect prediction. We have provided an overview of the previous approaches for defect prediction using semi-supervised learning algorithms. The future work should provide a clear distinction between supervised and semi-supervised learning and compare the efficiency of both techniques. SSL consists of many techniques for choosing the promising data; the future work can also incorporate research on these techniques and among them, which one is best for what kind of data and in which scenario. A detailed study is required that clearly describes the conditions under which one should switch between semi-supervised learning and supervised learning approaches. Availability of the required resources can also be a major concept of discussion in the future for the choice of machine learning approaches. This study has touched on the topic of "why SSL in terms of prediction and evaluation purposes." The future study can also provide an analytical evaluation of the machine learning techniques for prediction purposes.

Data Availability

The data used to support the findings of this study are included within this article.

Conflicts of Interest

The author declares no conflicts of interest.

Acknowledgments

The research work was supported by the College of Computing and Information Technology, Shaqra University, KSA.

References

[1] R. Premraj and K. Herzig, "Network versus code metrics to predict defects: a replication study," in *Proceedings of the International Symposium on Empirical Software Engineering and Measurement*, September 2011.

[2] P. Jalote, *An Integrated Approach to Software Engineering*, Springer Science & Business Media, Berlin, Germany, 2012.

[3] P. Kapur, H. Pham, and A. Gupta, *Software Reliability Assessment with OR Applications*, Springer, 2011.

[4] J. Li, P. He, and J. Zhu, "Software defect prediction via convolutional neural network," in *Proceedings of the IEEE International Conference on Software Quality, Reliability and Security (QRS)*, IEEE, Prague, Czech Republic, July 2017.

[5] X. Cai, Y. Niu, S. Geng et al., "An under-sampled software defect prediction method based on hybrid multi-objective cuckoo search," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 5, Article ID e5478, 2020.

[6] S. Wang, T. Liu, J. Nam, and L. Tan, "Deep semantic feature learning for software defect prediction," *IEEE Transactions on Software Engineering*, vol. 46, no. 12, pp. 1267–1293, 2020.

[7] N. Li, M. Shepperd, and Y. Guo, "A systematic review of unsupervised learning techniques for software defect prediction," *Information and Software Technology*, vol. 122, Article ID 106287, 2020.

[8] Z. Xu, L. Jin, X. Luo, and Z. Yang, "Software defect prediction based on kernel PCA and weighted extreme learning machine," *Information and Software Technology*, vol. 106, pp. 182–200, 2019.

[9] H. K. Dam, T. Pham, S. Wee, and T. Tran, "A deep tree-based model for software defect prediction," 2018, https://arxiv.org/abs/1802.00921.

[10] M. K. Thota, F. H. Shajin, and P. Rajesh, "Survey on software defect prediction techniques," *International Journal of Applied Science & Engineering*, vol. 17, no. 4, pp. 331–344, 2020.

[11] C. Rausch, *Algorithms for Geometric Optimization and Enrichment in Industrialized Building Construction*, 2021.

[12] Z. Li, X.-Y. Jing, and X. Zhu, "Progress on approaches to software defect prediction," *IET Software*, vol. 12, no. 3, pp. 161–175, 2018.

[13] D.-L. Miholca, G. Czibula, and I. G. Czibula, "A novel approach for software defect prediction through hybridizing

gradual relational association rules with artificial neural networks," *Information Sciences*, vol. 441, pp. 152–170, 2018.
[14] S. Herbold, "On the costs and profit of software defect prediction," *IEEE Transactions on Software Engineering*, vol. 47, 2019.
[15] X. Huo, Y. Yang, and M. Li, "Learning semantic features for software defect prediction by code comments embedding," in *Proceedings of the IEEE international conference on data mining (ICDM)*, IEEE, Singapore, November 2018.
[16] J. Jiarpakdee, C. Tantithamthavorn, and J. Grundy, "Practitioners' perceptions of the goals and visual explanations of defect prediction models," 2021, https://arxiv.org/abs/2102.12007.
[17] J. E. Van Engelen and H. H. Hoos, "A survey on semi-supervised learning," *Machine Learning*, vol. 109, no. 2, pp. 373–440, 2020.
[18] R. Caruana, N. Karampatziakis, and A. Yessenalina, "An empirical evaluation of supervised learning in high dimensions," in *Proceedings of the 25th international conference on Machine learning*, Helsinki Finland, July 2008.
[19] A. Pramod, H. S. Naicker, and A. K. Tyagi, "Machine learning and deep learning: open issues and future research directions for the next 10 years," *Computational Analysis and Deep Learning for Medical Care: Principles, Methods, and Applications*, pp. 463–490, 2021.
[20] U. A. Butt, M. Mehmood, S. B. H. Shah et al., "A review of machine learning algorithms for cloud computing security," *Electronics*, vol. 9, no. 9, 2020.
[21] B. Wu, D. Meng, and H. Zhao, "Semi-supervised learning for seismic impedance inversion using generative adversarial networks," *Remote Sensing*, vol. 13, no. 5, pp. 909–2021, 2021.
[22] M. Alipour and D. K. Harris, "A big data analytics strategy for scalable urban infrastructure condition assessment using semi-supervised multi-transform self-training," *Journal of Civil Structural Health Monitoring*, vol. 10, no. 2, pp. 313–332, 2020.
[23] 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.
[24] A. H. Khan and M. Zubair, "Classification of multi-lingual tweets, into multi-class model using Naïve Bayes and semi-supervised learning," *Multimedia Tools and Applications*, vol. 79, no. 43-44, Article ID 32749, 2020.
[25] P. Dahiya and D. K. Srivastava, "A comparative evolution of unsupervised techniques for effective network intrusion detection in hadoop," in *Proceedings of the International Conference on Advances in Computing and Data Sciences*, Springer, Dehradun, India, April, 2018.
[26] Sindhwani, Vikas, M. Belkin, and P. Niyogi, "The geometric basis of semi-supervised learning,".
[27] Y. Cho, *Kernel Methods for Deep Learning*, University of California, San Diego, CL, USA, 2012.
[28] S. Ghosh, N. Das, T. Goncalves, P. Quaresma, and M. Kundu, "The journey of graph kernels through two decades," *Computer Science Review*, vol. 27, pp. 88–111, 2018.
[29] S. Bogdan and K. Zdenko, "Fuzzy controller design based on the phase plane isoclines," in *Proceedings of the 14th Mediterranean Conference on Control and Automation*, Ancona, Italy, June 2006.
[30] C. Kristiansen and D. Shanti, *Integrating Solutions to Solving the Cold Start Problem in the Wikipedia Recommender System*, 2013.
[31] M. Hein, J. Y. Audibert, and U. V. Luxburg, "From graphs to manifolds–weak and strong pointwise consistency of graph Laplacians," in *Proceedings of the International Conference on Computational Learning Theory*, pp. 470–485, Springer, Berlin, Heidelberg, 2005.
[32] J. Bloom and J. Richards, "Data mining and machine-learning in time-domain discovery & classification," *Advances in Machine Learning and Data Mining for Astronomy*, pp. 89–112, 2011.
[33] A. Subramanya and J. Bilmes, "Semi-supervised learning with measure propagation," *Journal of Machine Learning Research*, vol. 12, no. 11, 2011.
[34] B. Chidlovskii and J. Fuselier, "HTML-to-XML migration by means of sequential learning and grammatical inference," in *Proceedings of the Workshop on Grammatical Inference Applications*, Edinburgh, Scotland, July, 2005.
[35] D. Bruckner, *Probabilistic Models in Building Automation: Recognizing Scenarios with Statistical Methods*, 2007.
[36] A. Majd, M. Vahidi-Asl, A. Khalilian, P. Poorsarvi-Tehrani, and H. Haghighi, "SLDeep: statement-level software defect prediction using deep-learning model on static code features," *Expert Systems with Applications*, vol. 147, Article ID 113156, 2020.
[37] F. Pecorelli, F. Palomba, and D. D. Nucci, "Comparing heuristic and machine learning approaches for metric-based code smell detection," in *Proceedings of the IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, IEEE, Montreal, QC, Canada, May 2019.
[38] E. Kupiainen, M. V. Mäntylä, and J. Itkonen, "Using metrics in Agile and Lean Software Development–A systematic literature review of industrial studies," *Information and Software Technology*, vol. 62, pp. 143–163, 2015.
[39] X. Xia, S. Emad, and Y. Kamei, "Predicting crashing releases of mobile applications," in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, Ciudad Real Spain, September, 2016.
[40] Q. Huang, X. Xia, and D. Lo, "Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction," *Empirical Software Engineering*, vol. 24, no. 5, pp. 2823–2862, 2019.
[41] Y. Gao, J. Ma, and A. L. Yuille, "Semi-supervised sparse representation based classification for face recognition with insufficient labeled samples," *IEEE Transactions on Image Processing*, vol. 26, no. 5, pp. 2545–2560, 2017.
[42] R. K. Saxena, A. Hake, A. J. Hingane et al., "Translational Pigeonpea Genomics Consortium for accelerating genetic gains in pigeonpea (Cajanus cajan L.)," *Agronomy*, vol. 10, no. 9, 2020.
[43] F. Matloob, T. M. Ghazal, N. Taleb et al., "Software defect prediction using ensemble learning: a systematic literature review," *IEEE Access*, vol. 9, 2021.
[44] D. Homan, *Tree Species Identification and Leaf Segmentation from Natural Images Using Deep Semi-supervised Learning*, Stellenbosch University, Stellenbosch, South Africa, 2022.
[45] D. Homan, "Tree species identification and leaf segmentation from natural images using deep semi-supervised learning," 2022, https://arxiv.org/abs/2110.03994.
[46] G. Chao, S. Sun, and J. Bi, "A survey on multi-view clustering," *IEEE Transactions on Artificial Intelligence*, vol. 2, 2017.
[47] Y. Leng, X. Xu, and G. Qi, "Combining active learning and semi-supervised learning to construct SVM classifier," *Knowledge-Based Systems*, vol. 44, pp. 121–131, 2013.

[48] Y. ShAo, B. Liu, S. Wang, and P. Xiao, "A novel test case prioritization method based on problems of numerical software code statement defect prediction," *Eksploatacja i Niezawodnosc-Maintenance and Reliability*, vol. 22, no. 3, pp. 419–431, 2020.

[49] T. De Bie, T. T. Maia, and A. de Pádua Braga, "Machine learning with labeled and unlabeled data," in *Proceedings of the 17th European Symposium on Artificial Neural Networks*, Bruges, Belgium, April 2009.

[50] M. Prasad, L. F. Florence, and A. AryaIII, "A study on software metrics based software defect prediction using data mining and machine learning techniques," *International Journal of Database Theory and Application*, vol. 8, no. 3, pp. 179–190, 2015.

[51] S. K. Pandey and A. K. Tripathi, "Class imbalance issue in software defect prediction models by various machine learning techniques: an empirical study," in *Proceedings of the 8th International Conference on Smart Computing and Communications (ICSCC)*, IEEE, Kochi, Kerala, India, July 2021.

[52] A. Cummaudo, *Improving the Reliability of Cloud-Based Pretrained Machine Learning Models*, Deakin University, Geelong, Australia, 2021.

[53] I. Triguero, S. García, and F. Herrera, "Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study," *Knowledge and Information Systems*, vol. 42, no. 2, pp. 245–284, 2015.

[54] W. Li, W. Zhang, X. Jia, and Z. Huang, "Effort-Aware semi-Supervised just-in-Time defect prediction," *Information and Software Technology*, vol. 126, Article ID 106364, 2020.