Machine Learning in Software Defect Prediction – A Literature Survey

Amirhosein Abbasi
ECE Dept.
UBC
amirosein@ece.ubc.ca

Behzad Aminian
Mech. Eng. Dept.
UBC
aminian.bz@gmail.com

Marjane Namavar
ECE Dept.
UBC
marjane@ece.ubc.ca

Abstract

Software Defect Prediction(SDP) has been a popular topic in both the Software engineering (SE) and Machine Learning (ML) domain because of its significant contribution toward reducing the cost and time of projects. Researchers have published many individual studies for various ML methods for SDP over the past few years. In addition, several survey articles have systematically reviewed the literature too. The last review article on applications of machine learning in defect prediction is for 2014. Also, there has been some review articles for unsupervised methods and cross-project challenge in SDP in recent years. However, a review article that covers the most recent studies from 2014 to 2019, along with the identification of the current trends and issues, is needed. In this study, we reviewed 23 articles in the domain of SDP to investigate the use of different ML models, defect data features, and performance evaluation in SDP papers. Additionally, the latest trends in SDP are introduced, and detailed information on what has been done so far to address these challenges is provided.

1 Introduction

Software defect prediction is one of the most critical steps in software development life-cycle. It is super-useful for software practitioners to be aware of potential defects in their software program. Locating potential defects in software projects can help to decide on designating more resources for testing and developing problematic projects or components. Software quality assurance experts would focus more on parts of programs with higher likelihoods of being defect-pone. The term "Software defect" is defined as a difference between what is expected and what is happening after production. This variance can be happened due to an internal or external problem. A software defect is usually introduced to software because of a human error, which causes the program to perform unexpectedly. Different factors can affect how much a software program is prone to defects. Program complexity, size, structure, and design of different components, classes, or objects can affect the chance of software to have defects. The relation of different classes, objects, or components should be considered when predicting the chance of software to be defect-prone. There have been different machine learning methods trying to come up with accurate solutions for defect prediction. The mentioned metrics can be obtained from the source code of software programs and then be used to create a machine learning model to predict the rate of defects for new programs.

In this research, we have a review on articles published from 2015 or later. We considered 23 articles from the last recent years. We categorized them based on the machine learning techniques they used, feature metrics, database, performance evaluation metrics and the contribution in the literature. What makes this study different from the previous ones is that we have identified recent trends and challenges in defect prediction domain and individually studied what is done to deal with them and what the probable solutions are.

2 Related work

In 2011, Catal et al. [50] had a systematic review on machine learning in defect prediction. They reviewed 90 articles based on their contents and categorized them. In 2015 Malhotra et al. [2] had a comprehensive review on using machine learning approaches for fault prediction in software. Their group reviewed 64 primary papers from 1991 to 2014 and organized different machine learning techniques with respect to their model, metric, performance and database. Similar studies have been done in this area of research. For example, Li et al. [3] had a review only on unsupervised techniques used in defect prediction.

3 Methodology

In this section we briefly explain the approach that has been taken in order to perform this literature survey project. To reach our study goals, we followed the Systematic Literature Review (SLR) guideline suggested by Hall et al. [32]. this procedure consists of several steps such as identification of the research questions, strategic article search, article selections and so on. Figure 1 is a schematic diagram of the steps taken to conduct this research.

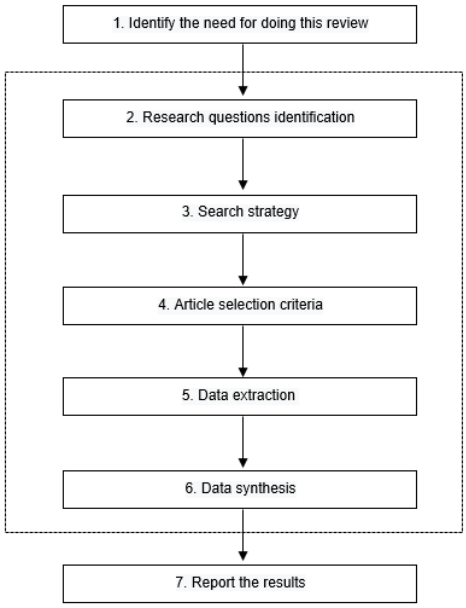

Figure 1: Methodology

3.1 Research questions

Based on previous literature reviews on similar subjects and by having a quick review of some articles we have carefully defined 5 study questions that help us achieve our goal which is to summarize, analyze and evaluate studies regarding their machine learning techniques, metrics, feature selection approach, and performance evaluation. Research questions are tabulated in table 1.

Table 1. Research questions

Research Question	**Motivation**
RQ1. What modeling techniques are used?	To identify the common techniques that are being used for fault prediction.
RQ2. What dataset is used?	To find appropriate datasets for software fault prediction.
RQ3. Which software metrics are commonly used for	To find out the metric that works well for fault prediction.

defect prediction?	
RQ4. What is the feature extraction technique?	Identify which features perform the best.
RQ5. Which performance evaluation criteria are used?	Investigate the performance of the ML technique in prediction software defects.

3.2 Search strategy

After identifying our research goal and questions, we defined some subtopic of SFP like cross-project defect prediction (CPDP), within project defect prediction (WPDP) and classification techniques to mention but a few. We then searched for the related studies in the most famous software engineering journals. In the table 2, the distribution of papers among journals and conferences is presented. One of the priorities in the selection of papers is the impact factor of its' journal or conference. All of the studies in our review are selected from renowned software engineering or machine learning journals and conferences. The majority of studies (More than 60 percent) are selected from the top three software engineering journals and conferences. The following electronic databases were used for finding the primary studies:

- IEEE Xplore
- ScienceDirect
- Research gate
- Wiley Online Library
- SpringerLink

Table 2. Search Strategy

Journal or Conference	**Abbreviation**	**Studies**
International Conference on Software Engineering	ICSE	[4],[5],[10], [21],[37]
IEEE Transactions on Software Engineering	TSE	[1],[2],[6], [7],[15],[18], [28],[32], [36], [38]
Information & Software Technology	IST	[3],[8],[9], [23],[45], [47]
Empirical Software Engineering and Measurement	ESEM	[11]
The joint meeting of the European Software Engineering Conference/ Symposium on the Foundation of Software Engineering	ESEC/FSE	[12]
International Conference on Software Engineering & Knowledge Engineering	SEKE	[13]
Empirical Software Engineering	EMSE	[14],[26]
Computer Software and Applications Conference	COMPSAC	[16]
Journal of Computer Science and Technology	JCST	[17]
International Journal of Computer, Electrical, Automation, Control and Information Engineering	IJCAS	[19]

International Conference on Information and Communication Technology for Competitive Strategies	ICTCS	[20]
Automated Software Engineering	ASE	[27],[44], [46]
Miscellaneous	HPCC, SmartCity/DSS/SIC/ICBDA/ICML/TPAMI/ECCE	[22],[24], [33],[34], [35],[43]

3.3 Study selection criteria

Among approximately 50 related studies found, only 23 of them are considered in this study. The selection of the studies was done using the inclusion and exclusion criteria listed below:

3.3.1 Inclusion criteria

- The paper should be an empirical study of the application of Machine learning techniques in software defect prediction.
- The study must be published in 2015 or later.
- The study must be a peer-reviewed full research paper published in an authentic journal or conference in the domain of software engineering or machine learning.

3.3.2 Exclusion criteria

- Studies focused on topics other than the application of ML techniques in software defect prediction.
- Grey Literature or in a language other than English.
- Studies with unclear or invalid results.
- Studies that are published in journals or conferences other than software engineering or machine learning domain.

3.4 Data extraction

A form was designed including the article information and the research question for each primary study that was selected after considering the inclusion and exclusion criteria. The articles were then divided between 3 researchers. But before we cover all the studies, the data extracted by the researchers at each step were discussed in the group to guarantee the consistency between the researchers.

3.5 Data Synthesis

The primary goal of data synthesis is to express the data obtained from figures and tables in the primary study articles in a way that can be compared with other results. Particularly, this review aims to find the answer to the research question in such a way that can be interpreted and decided what machine learning models or software engineering metrics can do better than the others.

4 Results and Discussion

In this section, we answer the research questions based on the information we gained from 23 selected papers.

4.1 RQ1: What ML techniques have been used for SDP?

There have been different methods in machine learning literature, trying to address this issue. Previous studies believe that classification has a trivial impact on software defect prediction [28, 27, 1]. For example, [1] explores an analysis based on 42 studies to find out

what factors have a significant impact on software defect prediction. They surprisingly report that the choice of classifiers has an impact of only 1.3 percent on SDF. Based on their analysis, the researcher group is the most significant factor in SDP. But, another study [4] believes that their result is biased through their target dataset. They state that the NASA dataset, which these studies are based upon, has some implausible noisy input, which misleads the results of classification methods. This paper proves that some classification methods outperform others, and there are different clusters of ML methods. Thus, considering their results, we discuss all classification methods and include what studies have done them so far. There is a wide range of machine learning methods that have been used for software defect prediction in different studies. Both supervised and supervised learning methods were used.

4.1.1 Supervised methods

Table 3 shows supervised ML methods that were used in studied papers.

Statistical Techniques: These techniques are based on probabilities, and they output the chance of each category for each software program. Major defined categories in SDP are defect-prone and not defect-prone. These techniques model the input data and try to find patterns in the data set; consequently, they are based on big assumptions. For instance, Naive Bayes is a statistical model that assumes that all predictors are independent of each other [14, 19, 17, 8, 15, 9, 26]. Also, there have been some studies that use logistic regression, which is also based on probabilities, as the predictor. [13, 8, 15]. Moreover, the Bayesian Network is used in one study [8]. Also, non-parametric statistical methods, like the nearest neighborhood, have been adopted by some studies [10, 15]. These methods, like KNN, compute the distance between input data to find nearest neighbors to one data to classify it. Lazy learning and expensive testing are the most notable characteristics of this method.

Decision Trees: These methods represent a tree as the model, and each node in the tree represents a rule for one feature. Leaf nodes are categorized into defective or non-defective. Each input data goes through a path from the root to a leaf node to be classified. Logistic Model Tree (LMT) and J48 are some examples of decision trees as the predictor in SDP previously [29, 30]. The results from using these methods have been considered in a prior study [4]. Another popular category of decision tree models is the random forest used in previous studies [15, 26, 16]. This model generates trees from random bootstraps of data and averages their results.

Regression Methods: A number of prior studies have chosen SVM (Support-Vector-Machine) as their machine learning model [15, 8, 26]. SVM is a non-probabilistic binary linear classifier which finds a hyperplane to separate the defective or non-defective classes when used in SDP. One of the prior studies reviews the result of the Sequential Minimal Optimization(SMO) SVM technique [4].

Ensemble Methods: Ensemble methods use the result of different models to come up with another model for classifying examples. In our study, we found previous studies that adopted these methods for SDP: Max Pooling, Average Pooling [16], Bagging [31], AdaBoost [33], Rotation Forest [34], and Random Subspace [35]. Bagging consists of forming bootstraps of data and then aggregating their results. It uses voting or averaging of different subsets of data to predict the outcome. AdaBoost uses the output of different models to a weighted sum and performs multiple iterations with different weights and use voting for the final result. Rotation forest performs different feature extractions to subsets of data with the help of Principal Component Analysis (PCA). Then the newly generated set of features is used in the final ensemble. Moreover, Random Subspace creates a random forest of multiple decision trees and use it as the prediction model.

Table 3. Supervised ML techniques

ML technique	Studies
Naïve Bayes	[14], [19], [17],[8], [15],[9], [26]

Logistic Regression	[13], [8], [15]
Random Forest	[15], [26], [16]
Nearest Neighbour	[10], [15]
Decision Tree	[26], [8]
SVM	[15], [8], [26],
Decision Table	[8]
Bayesian Net (statistical)	[8]
Max voting	[16]
Average Voting	[16]

4.1.2 Unsupervised methods

Table 4 shows unsupervised ML techniques.

K-means: k-Partition is based on the distance of data points, assign them into k exclusive partitions or groups, where each partition represents a cluster. K-means is a subfamily of K-Partition where each data point is assigned into the cluster according to the distance with the centroid of a cluster, which is the mean of all points within the cluster. The initial number of clusters is assigned manually.

Neural-Gas clustering: Neural-Gas clustering is a competitive learning technique with SoftMax learning rule. Its inspiration comes from a type of neural network called SOM (self-organizing map). Neural gas is similar to both neural network-based clustering and partitioned one. Neural gas gained popularity because of its robust convergence compared to online k-means clustering. It is mostly used in speech recognition and image processing for data compression or vector quantization.

Fuzzy C-means: Fuzzy c-means (FCM) is a method of clustering which allows one piece of data to belong to two or more clusters. Assigns membership to each data point corresponding to each cluster center on the basis of the distance between the center and itself. This method is frequently used in pattern recognition.

Clustering and labelling: The primary objective of CLA is to classify objects into relatively homogeneous groups based on the set of variables considered. Objects in a group are relatively similar in terms of these variables, and different from objects from other groups. Computes instances violation degree based on the relation between metric values and corresponding median values, then clustering by the violation or consistency information.

CLAMI: Clustering by comparing each metric value with the corresponding median or average value. Based on CLA, add two extra steps: Metric selection and instance selection based on violation degree.

Average Clustering and labelling: Clustering by metrics of instances violation score (MIVS), which is computed based on the relation between each metric value and their average metric value.

Self-organizing maps: A self-organizing map (SOM) is a type of artificial neural network (ANN) that is trained using unsupervised learning to produce a low-dimensional (typically two-dimensional), discretized representation of the input space of the training samples, called a map, and is therefore a method to do dimensionality reduction.

Threshold: Metric Threshold. It might be determined by experiences, rules, calculations, or machine learning, etc. Uses some metric thresholds to classify instances directly. e.g., given a metric threshold vector, if any metric of an instance exceeds the corresponding threshold, it will be defect-prone.

Fuzzy self-organizing maps: In Fuzzy clustering, each data point can belong to two or more clusters. Fuzzy Self-Organizing Maps combines SOMs with the concept of fuzziness in fuzzy clustering.

Spectral clustering: Makes use of the eigenvalues of the similarity matrix to reduce dimensionality before clustering. Partitions a dataset based on the connectivity between its nodes in a graph with spectral clustering.

Expectation maximization clustering (EM): Gets probabilities of cluster memberships based on probability distributions. Finds the maximum likelihood by iteratively alternating between expectation step and maximization step.

Metric ranking: Used for some change metrics in just in time prediction. Considers 1/Metric to rank instances in ascending order.

Table 4. Unsupervised ML techniques

ML technique	Studies
K-means	[20], [21]
Neural-Gas clustering	[21]
Fuzzy C-means	[21]
Clustering and labelling (CLA)	[22], [44]
CLA + Metric selection and instance selection based on violation degree (CLAMI)	[22], [44]
Average Clustering and labelling (ACL)	[22],
Self-organizing maps	[22], [23]
Threshold	[8], [23], [44]
Fuzzy self-organizing maps	[24]
Spectral Clustering	[25]

Expectation maximization clustering	[25], [44]
Metric ranking	[45]

4.2 RQ2: What datasets have been used for SDP? What are the defect distributions in them?

Two types of datasets can be used for SDP. First, publicly available datasets and second, private datasets. The first category of datasets is better since their results are repeatable. But, for the private datasets, the distribution of faults is not available. Below is the list of datasets that are used for studies. These datasets contain source codes as well as their label, which is defective or non-defective.

- NASA dataset. This dataset is publicly available for researchers. The defect distribution is available too. Most of the studies use this dataset for their research. Also, there is a previous work that believes that this dataset contains noisy and implausible data, and prior studies have biased to this noisy data. [41].
- PROMISE repository dataset: This dataset is publicly available and is also created by NASA to encourage repeatable, verifiable, refutable, and improvable predictive models of software engineering. [40].
- Other open-source datasets like ANT, IVY, LUCENE, POI, TOMCAT, KCI, JEdit, Eclipse.

In table 5, we represent the datasets that are used in different studies. Some studies use different datasets for file-level and change-level SDP. For instance, one study [5, 6] uses all java open source projects from the promise data set for the file-level defect prediction. The numbers of files of the projects in their file-level dataset range from150 to 1,046, and the buggy rates of the projects have a minimum value of 13.4% and a maximum value of 49.7%. However, they use different sets of data for change-level defect prediction. Linux kernel, PostgreSQL, Xorg, Jdt (from Eclipse), Lucene, and Jackrabbit are used for change-level defect prediction in this study. The defect distribution in these datasets are different.

Table 5. Datasets

Dataset Name	**Project**	**Study**
NASA	PROMISE, MDP	[17], [19], [26], [21], [22], [25], [2], [4], [5], [6]
Jureczko (JUR)	Ant, Camel, Ivy, Log4j, POI, Synapse, JEdit, Xalan, etc.	[8],[9],[10],[12],[13],[16],[23]
AEEEM	JDT core, JDT Eclipse, PDE Eclipse, Mylyn, Apache Lucence	[12],[14],[21],[22],[23], [5], [6], [47]
ReLink	Apache HTTP Server, OpenIntents Safe, and ZXing	[12],[22]
SoftLab	PostgreSQL (POS), Ruby on Rails (RUB), Rhino (RHI)).	[24], [47],
Miscellaneous	Licq, GitHub, Linux kernel, Jackrabbit, Mozilla	[14],[20], [5], [6], [47]

4.3 RQ3: What SE metrics(features) have been used for SDP?

For predicting software defects, the first step is to find a way to characterize software programs with some metrics. Software engineering experts adopt various metrics to quantify

software programs. Previous studies offer different software metrics, for example, object-oriented metrics like Chidamber and Kemerer (CK) metrics [36], Process metrics like the number of changes, recent activity [37], Product metrics like size of program and complexity [38], Historical metrics for monitoring changes [38], and structural metrics like cyclomatic complexity or coupling [39]. However, in our review, all the above metrics were not adopted by researchers for SDP. Below is the list of software metrics categories that were chosen in studied works.

Traditional metrics: These are the standard software engineering metrics that we exhibited above. They try to address the characteristics of software programs from different points of view, like size, complexity, structure, behavior, and the relation of different software components [36, 37, 38, 39].

Abstract Syntax Tree (AST): Derived feature. AST is a detailed tree representation of the Java source code. AST is utilized for extracting syntactic information from the source code.

There are different nodes in AST:

- nodes of method invocations and class instance creations
- declaration nodes, i.e., method declarations, type declarations, and enum declarations
- control-flow nodes such as while statements, catch clauses, if statements, throw statements, etc.

Some studies [5, 6] create vectors of AST nodes and use those vectors as features.

Semantic features: Derived feature- These features cannot be found in the code. They are hidden deeply in the source code and they are useful for bug detection or program comprehension. Table 6 shows all metrics that were used in studied papers.

4.4 RQ4: What types of feature reduction (metrics reduction) has been used for SDP?

In order to consider only metrics related to fault-proneness in the prediction models, in [23] they performed univariate logistic regression analyses on the different metrics initially considered (SLOC, CBO, RFC, WMC, LCOM, DIT and NOC). The first conclusion we can draw is that SLOC, CBO, RFC, WMC and LCOM metrics are relevant for fault-proneness prediction. logistic regression was used to determine for each source code metric if it is good at determining the fault-proneness of a class. In another paper for determining the importance of the software metrics for the defect detection task, the information gain (IG) measure was used [24]. From the software metrics whose IG values were higher than a given threshold, a subset of metrics that measure different characteristics of the software system were finally selected. The initial set was all McCabe and Halstead metrics that finally halstead_vocabulary, total_operands, total_operators, executable_loc, halstead_length, total_loc, condition_count, branch_count and decision_count were selected. In CLAMI [44] metric selection was conducted based on the metric violation scores (MVS). Since the quality of defect prediction models is highly dependent on the quality of the metrics, metric selection to choose the most informative metrics for prediction models has been widely adopted in defect prediction. Metric selection in CLAMI is based on removing metrics that can minimize violations in the defect-proneness tendency of defect datasets. Not all metrics follow the defect-proneness tendency so that metric selection of CLAMI can be helpful to build a better prediction model. A violation is a metric value that does not follow the defect-proneness tendency. Some supervised learning methods in order to select features, filter out infrequent AST nodes and consider code changes like code addition or the context of the code in the change [5, 6].

4.5 RQ5: What performance measures have been used for evaluating different methods?

For answering this research question, we inspect different ways that studies have used to evaluate their software defect prediction method. Some studies have used more than one

performance metric for making comparison or giving more fine-grained results about the performance. Table 7 shows all performance measures that were used in studied papers.

5 Current research trends and SDP challenges

In this section, we explore 3 main areas of challenges in SDP that are hot topics and research trends as well.

5.1 Feature selection and extraction

One of the most important factors in improving a machine learning model is to use proper features. Feature extraction, feature selection, and also deriving new features from source code are always a considerable part of researches in SDP literature. There are myriad of metrics to quantify software codes, and lots of them are introduced in the software engineering domain. Consequently, it would be overwhelming for software engineering practitioners to select the best features for their software defect predictor model. In addition, software defect data are hindered by obstacles like redundancy, correlation, feature irrelevance, and missing samples. Some prior [5, 6] studies focus on extracting semantic features from source code [5, 6]. They believe that the syntactic data of the source code can not cover the whole characteristics of the code. So they try to get the semantics of the source code with the help of existing syntactic features. They show that the accuracy of the SDP model significantly increases while they are trained with semantic features. Moreover, some other works in literature contribute to the machine learning part, but they also introduce a good set of features that fit their model best [4]. Some studies select features by combining different methods of feature selections [48, 49].

5.2 Dealing with imbalanced Data

Imbalanced classes are a common problem in machine learning classification where there is a disproportionate ratio of observations in each class. Class imbalance can be found in many different areas including medical diagnosis, spam filtering, and fraud detection. Most machine learning algorithms work best when the number of samples in each class are about equal. This is because most algorithms are designed to maximize accuracy and reduce error. There are various approaches that could be taken to deal with imbalanced data problem:

Change the performance metrics: Accuracy is not the best metric to use when evaluating imbalanced datasets as it can be very misleading. Confusion matrix, precision, recall and F-measure are better choices [15]. Confusion matrix is a table showing correct predictions and types of incorrect predictions. Precision is the number of true positives divided by all positive predictions. Precision is also called Positive Predictive Value. It is a measure of a classifier's exactness. Low precision indicates a high number of false positives. Recall is the number of true positives divided by the number of positive values in the test data. Recall is also called Sensitivity or the True Positive Rate. It is a measure of a classifier's completeness. Low recall indicates a high number of false negatives. F1-measure is the weighted average of precision and recall.

Change the algorithm: While in every machine learning problem, it's a good rule of thumb to try a variety of algorithms, it can be especially beneficial with imbalanced datasets. Decision trees frequently perform well on imbalanced data [32].

Oversample minority class: Oversampling can be defined as adding more copies of the minority class. Oversampling can be a good choice when you don't have a ton of data to work with [43].

Table 6. Software metric

Metrics category	metrics	Description of metrics	Interpretation	Studies
Size (LOC counts)	LOC LOC_total, LOC_blank, LOC_comment, LOC_code_and_comment, LOC_executable, andNumberoflines	Measures the size of the software	Softwares with more lines of codes have a higher probability for being defective.	[45],[21], [22], [9] ,[26], [12], [15], [17],[8], [10], [14], [16]
McCabe Software Metrics	cyclomatic_complexity, cyclomatic_density, design_complexity, essential_complexity, and pathological_complexity, max_cc and avg_cc	Measures of the branching complexity of the software module	More complexity means bigger chance of being defective.	[24], [25], [44], [21], [22]
Halstead attributes	content, difficulty, effort, errorest, length, level, progtime, volume, num_operands, num_operators,num_unique_operands, num_unique_operators	Estimates of the complexity of reading a software module based on the vocabulary used (e.g.,number of operators and operands).	More complexity means bigger chance of being defective.	[24], [25], [44], [21], [22]
CK	wmc, dit, cbo, noc, lcom, rfc, ic, cbm, amc, lcom3	Measures of the complexity of a class within an object-oriented system design	More complexity means bigger chance of being defective.	[21], [22], [20], [23]
QMOOD	dam, moa, mfa, cam, npm	Measures of the behavioural and structural design properties of classes, objects, and the relations between them.	More QMOOD means higher quality code. Software codes with lower QMOOD are prone to defects.	[21], [22]
Martin's	ca, ce	Measures of the relationship between software components, including calls as well as number of instances	Higher values indicate low encapsulation,reusability, and maintainability [45], which may lead to defects	[21], [22]
Object Orient Metrics	DIT, LCOM, RFC, NOC, CBO, CF, AIF, MIF, AHF	Measures Depth of Inheritance Tree,	N/A	[20], [8], [10], [12], [14], [16], [12], [15],[23]
Miscellaneous	branch_count, call_pairs, condition_count,decision_count, decision_density, design_density,edge_count, essential_density, parametercount,maintenance_severity, modified_condition_count,multiple_condition_count, global_data_density,global_data_complexity, percent_comments,normalized_cyclomatic_complexity and node_count	Different metrics	N/A	[45], [21], [22], [14], [19]

Table 7. Performance measures

Measure	Description	Definition	Studies
Precision	The proportion of the predicted positive cases that were correct	$\frac{TP}{TP+FP}$	[8], [14], [20], [22], [25], [44]
Recall, pd	The proportion of positive cases that were correctly identified	$\frac{TP}{TP+FN}$	[8], [26], [10], [14], [15], [17], [20], [22], [25], [44]
Probability of False Alarm, pf	Proportion of non-faulty units incorrectly classified as fault-prone	$\frac{FP}{FP+TN}$	[9], [26], [10], [14], [15], [17]
AUC, Area Under the Curve	The area under the receiver operating characteristics curve. Independent of the cutoff value	N/A	[26], [12], [14], [19], 20], [21], [24], [44]
F-measure	Harmonic mean of precision and recall	$\frac{2 \times Precision \times Recall}{Precision + Recall}$	[8], [14], [15], [16], [22], [25], [44], [45]
G-measure	Harmonic mean of pd and (1-pf)	$\frac{2 \times pd \times (1-pf)}{pd + (1-pf)}$	[10], [14]
Balance	The euclidean distance from the (pd, pf) point to (pd=1, pf=0) in the ROC curve	$1 - \sqrt{\frac{(pf)^2 + (1-pd)^2}{2}}$	[17]
MCC	A compound measure considering all true and false positives and negatives	$\frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$	[9], [14], [15]
Specificity	Is defined as the proportion of actual negatives, which got predicted as the negative	$\frac{TN}{FP+TN}$	[20]
Accuracy	a ratio of correctly predicted observation to the total observations	$\frac{TN+TP}{TP+FP+FN+TN}$	[20], [25]
FP rate	refers to the expectancy of the false positive ratio	$\frac{FP}{FP+TN}$	[23]
FN rate	The false negative rate is the proportion of the individuals with a known positive condition for which the test result is negative.	$\frac{FN}{FN+TP}$	[23]
G-mean	Is a metric that measures the balance between classification performances on both the majority and minority classes. A low G-Mean is an indication of a poor performance.	N/A	[23]
Miscellaneous	H-measure, Cost, Hubert Stat. Procedure	N/A	[26]

Change the algorithm: While in every machine learning problem, it's a good rule of thumb to try a variety of algorithms, it can be especially beneficial with imbalanced datasets. Decision trees frequently perform well on imbalanced data [32].

Oversample minority class: Oversampling can be defined as adding more copies of the minority class. Oversampling can be a good choice when you don't have a ton of data to work with [43].

Undersample majority class: Undersampling can be defined as removing some observations of the majority class. Undersampling can be a good choice when you have a ton of data - think millions of rows [43]. But a drawback is that we are removing information that may be valuable. This could lead to underfitting and poor generalization to the test set.

Generate synthetic samples: A technique similar to upsampling is to create synthetic samples. For example, some methods use a nearest neighbors algorithm to generate new and synthetic data we can use for training our model [43]. Again, it's important to generate the new samples only in the training set to ensure our model generalizes well to unseen data.

5.3 Cross-Project Defect Prediction

It's not always possible for companies to train a model based on their local data to predict the fault in their programs. The lack of local data or the difficulty to collect it is one important reason why companies are reluctant to use defect prediction. Another issue is that companies change their practice in the course of time and current local data might not be representative in the near future so, they are not interested in organizing all these local data. One alternative is to use models that are trained based on similar projects and then use it to predict the defects in another project. This approach produces fairly satisfying results and it is financially reasonable which makes it very popular especially in the last recent years. Researchers have been interested in this topic recently. He et al. [8] showed that simple classifiers like NB can do as well as the other modeling method in CPDP. Also, they showed that NB and BN are quiet stable and produce reasonable results for different metric sets in source and target data. Chen et al. [13] tried a regression model on six-open-source projects and concluded that a good regression model can perform well in many cases in comparison to more complicated models like random forests. Chen et al. [9] proposed a novel approach (Double Transfer Boosting DTB) to enhance the prediction models in CCDP. This approach involves data gravitation re-weight and transfer boosting that removes negative samples. They also offered a model (VCB-SVM) which is basically a model based on SVB and boosting that can address class imbalance issues. Peters et al. [10] presented LACE2 which is a multiparty sharing protocol that allows companies to predict the defect in their codes even when they don't have enough local data to build a model based on it, Nam et al. [12] proposed a heterogeneous defect prediction model (HDP) based on metric matching that can predict defects in software as good as other approaches.

6 Future work and limitations

This review is based on 23 prior studies that are published later than 2015 and does not cover the whole literature. Indeed, more fine-grained research should be done to make a general conclusion about challenges and popular methods and metrics in SDP. A potential future work for this review would be making a comparison between the performance results of different machine learning models in SDP when they are used by the same set of features and databases. Also, the same comparison can be made for different databases and different sets of features.

7 Conclusion

This paper presents a review of different machine learning models, datasets, and features used for software defect prediction. Machine learning models in SDP are categorized into supervised and unsupervised models, and each category and related prior studies are explained. An overview of software engineering metrics as features of SDP is provided.

Different categories of code metrics in SDP are depicted alongside the research works that based their model on them. Furthermore, different performance evaluation metrics for SDP machine learning models have been introduced as well as the papers that use them. Finally, recent challenges and trends in SDP are described, and some examples for each of them are provided.

References

[1] Shepperd, Martin, David Bowes, and Tracy Hall. "Researcher bias: The use of machine learning in software defect prediction." IEEE Transactions on Software Engineering 40.6 (2014): 603-616.

[2] Malhotra, Ruchika. "A systematic review of machine learning techniques for software fault prediction." *Applied Soft Computing* 27 (2015): 504-518.

[3] Li, Ning, Martin Shepperd, and Yuchen Guo. "A systematic review of unsupervised learning techniques for software defect prediction." arXiv preprint arXiv:1907.12027 (2019).

[4] Ghotra, Baljinder, Shane McIntosh, and Ahmed E. Hassan. "Revisiting the impact of classification techniques on the performance of defect prediction models." Proceedings of the 37th International Conference on Software Engineering-Volume 1. IEEE Press, 2015.

[5] Wang, Song, Taiyue Liu, and Lin Tan. "Automatically learning semantic features for defect prediction." 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE). IEEE, 2016.

[6] Wang, Song, et al. "Deep semantic feature learning for software defect prediction." IEEE Transactions on Software Engineering (2018).

[7] Hosseini, Seyed rebvar, Burak Turhan, and Dimuthu Gunarathna. "A systematic literature review and meta-analysis on cross project defect prediction." IEEE Transactions on Software Engineering 45.2 (2017): 111-147.

[8] P. He, B. Li, X. Liu, J. Chen, and Y. Ma, “An empirical study on software defect prediction with a simplified metric set,”Inf. Softw.Technol., vol. 59, pp. 170–190, 2015.

[9] L. Chen, B. Fang, Z. Shang, and Y. Tang, “Negative samples reduction in cross-company software defects prediction,”Inf.Softw. Technol., vol. 62, pp. 67–77, 2015.

[10] F. Peters, T. Menzies, and L. Layman, “Lace2: Better privacy preserving data sharing for cross project defect prediction,” inProc.37th Int. Conf. Softw. Eng.-Vol. 1, 2015, pp. 801–811.

[11] Yan, Meng, et al. "File-level defect prediction: Unsupervised vs. supervised models." *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 2017.

[12] J. Nam and S. Kim, “Heterogeneous defect prediction,” inProc.10th Joint Meeting Found. Softw. Eng., 2015, pp. 508–519.

[13] M. Chen and Y. Ma, “An empirical study on predicting defect numbers,” inProc. Int. Conf. Softw. Eng. Knowl. Eng., 2015, pp. 397–402.

[14] Zhang, Feng, et al. "Towards building a universal defect prediction model with rank transformed predictors." *Empirical Software Engineering* 21.5 (2016): 2107-2145.

[15] Bennin, Kwabena Ebo, et al. "Mahakil: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction." IEEE Transactions on Software Engineering 44.6 (2017): 534-550.

[16] Y. Zhang, D. Lo, X. Xia, and J. Sun, “An empirical study of classifier combination for cross-project defect prediction,” inProc. IEEE39th Ann. Comput. Softw. Appl. Conf., 2015, pp. 264–269.

[17] Ryu, Duksan, Jong-In Jang, and Jongmoon Baik. "A hybrid instance selection using nearest-neighbor for cross-project defect prediction." *Journal of Computer Science and Technology* 30.5 (2015): 969-980.

[18] Zhang, Feng, et al. "The use of summation to aggregate software metrics hinders the performance of defect prediction models." IEEE Transactions on Software Engineering 43.5 (2016): 476-491

[19] Singh, Pradeep, and Shrish Verma. "Cross project software fault prediction at design phase." Int. J. Comput. Electr. Automat. Control Inf. Eng 9.3 (2015): 800-805.

[20] Singh, Satwinder, and Rozy Singla. "Comparative performance of fault-prone prediction classes

with k-means clustering and MLP." Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies. ACM, 2016.

[21] Zhang, Feng, et al. "Cross-project defect prediction using a connectivity-based unsupervised classifier." Proceedings of the 38th International Conference on Software Engineering. ACM, 2016.

[22] Yang, Jun, and Hongbing Qian. "Defect prediction on unlabeled datasets by using unsupervised clustering." 2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS). IEEE, 2016

[23] Boucher, Alexandre, and Mourad Badri. "Software metrics thresholds calculation techniques to predict fault-proneness: An empirical comparison." Information and Software Technology 96 (2018): 38-67.

[24] Czibula, Istvan-Gergely, et al. "A novel approach using fuzzy self-organizing maps for detecting software faults." Studies in Informatics and Control 25.2 (2016): 207-216.

[25] Chang, Ruihua, et al. "A novel method for software defect prediction in the context of big data." 2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)(. IEEE, 2017.

[26] D. Ryu, O. Choi, and J. Baik, “Value-cognitive boosting with a support vector machine for cross-project defect prediction,”Empirical Softw. Eng., vol. 21, no. 1, pp. 43–71, 2016.

[27] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener, “Defect prediction from static code features, current results, limitations, new approaches,” Automated Software Engineering, vol. 17, no. 4, pp. 375–407, 2010

[28] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, Benchmarking classification models for software defect prediction: A proposed framework and novel findings,” IEEE Transactions on Software Engineering, vol. 34, no. 4, pp. 485–496, 2008.

[29] N. Landwehr, M. Hall, and E. Frank, “Logistic model trees,” Machine Learning, vol. 59, no. 1-2, pp. 161–205,2005.

[30] J. R. Quinlan, C4. 5: programs for machine learning.Morgan Kaufmann, 1993, vol. 1.

[31] L. Breiman, “Bagging predictors, ”Machine learning, vol. 24, no. 2, pp. 123–140, 1996.

[32] Song, Qinbao, Yuchen Guo, and Martin Shepperd. "A comprehensive investigation of the role of imbalanced learning for software defect prediction." IEEE Transactions on Software Engineering (2018).

[33] Y. Freund and R. E. Schapire, “Experiments with a New Boosting Algorithm,” in Proceedings of the International Conference on Machine Learning, 1996, pp. 148–156.

[34] J. Rodr´ıguez, L. Kuncheva, and C. Alonso, “Rotation forest: A new classifier ensemble method.”IEEE transactions on pattern analysis and machine intelligence, vol. 28, no. 10, pp. 1619–1630, 2006

[35] T. K. Ho, “The random subspace method for constructing decision forests, ”IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 20, no. 8, pp. 832–844,1998

[36] S. Chidamber and C. Kemerer, “A metrics suite for object-oriented design, ”IEEE Transactions on Software Engineering, vol. 20, no. 6, pp. 476–493, 1994.

[37] R. Moser, W. Pedrycz, and G. Succi, “A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction,” in Proceedings of the 30th international conference on Software engineering. ACM, 2008, pp. 181–190.

[38] T. Gyimothy, R. Ferenc, and I. Siket, “Empirical validation of object-oriented metrics on open source software for fault prediction, ”IEEE Transactions on Software Engineering, vol. 31, no. 10, pp. 897–910, 2005.

[39] E. Arisholm and L. C. Briand, “Predicting fault-prone components in a java legacy system,” in Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering. ACM, 2006, pp. 8–17.

[40] PROMISE data set: http://promise.site.uottawa.ca/SERepository/datasets-page.html

[41] NASA Data set: http://www.filesanywhere.com/fs/v.aspx?v=896a648c5e5e6f799b

[42] Gao, Kehan, Taghi M. Khoshgoftaar, and Amri Napolitano. "Exploring software quality classification with a wrapper-based feature ranking technique." *2009 21st IEEE International*

Conference on Tools with Artificial Intelligence. IEEE, 2009.

[43] Sohan, Md Fahimuzzman, et al. "Revisiting the Class Imbalance Issue in Software Defect Prediction." *2019 International Conference on Electrical, Computer and Communication Engineering (ECCE)*. IEEE, 2019.

[44] Nam, Jaechang, and Sunghun Kim. "Clami: Defect prediction on unlabeled datasets (t)." *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2015.

[45] Chen, Xiang & Zhao, Yingquan & Wang, Qiuping & Yuan, Zhidan. (2017). MULTI: Multi-Objective Effort-Aware Just-in-Time Software Defect Prediction. Information and Software Technology.

[46] Nam, Jaechang, and Sunghun Kim. "Clami: Defect prediction on unlabeled datasets (t)." *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2015.

[47] Chen, Xiang, et al. "MULTI: Multi-objective effort-aware just-in-time software defect prediction." *Information and Software Technology* 93 (2018): 1-13.

[48] Wang, Huanjing, Taghi M. Khoshgoftaar, and Amri Napolitano. "A comparative study of ensemble feature selection techniques for software defect prediction." *2010 Ninth International Conference on Machine Learning and Applications*. IEEE, 2010

[49] Laradji, Issam H., Mohammad Alshayeb, and Lahouari Ghouti. "Software defect prediction using ensemble learning on selected features." *Information and Software Technology* 58 (2015): 388-402.

[50] Catal, Cagatay. "Software fault prediction: A literature review and current trends." *Expert systems with applications* 38.4 (2011): 4626-4636.