Chapter

Data Mining and Machine Learning for Software Engineering

Elife Ozturk Kiyak

Abstract

Software engineering is one of the most utilizable research areas for data mining. Developers have attempted to improve software quality by mining and analyzing software data. In any phase of software development life cycle (SDLC), while huge amount of data is produced, some design, security, or software problems may occur. In the early phases of software development, analyzing software data helps to handle these problems and lead to more accurate and timely delivery of software projects. Various data mining and machine learning studies have been conducted to deal with software engineering tasks such as defect prediction, effort estimation, etc. This study shows the open issues and presents related solutions and recommendations in software engineering, applying data mining and machine learning techniques.

Keywords: software engineering tasks, data mining, text mining, classification, clustering

1. Introduction

In recent years, researchers in the software engineering (SE) field have turned their interest to data mining (DM) and machine learning (ML)-based studies since collected SE data can be helpful in obtaining new and significant information. Software engineering presents many subjects for research, and data mining can give further insight to support decision-making related to these subjects.

Figure 1 shows the intersection of three main areas: data mining, software engineering, and statistics/math. A large amount of data is collected from organizations during software development and maintenance activities, such as requirement specifications, design diagrams, source codes, bug reports, program versions, and so on. Data mining enables the discovery of useful knowledge and hidden patterns from SE data. Math provides the elementary functions, and statistics determines probability, relationships, and correlation within collected data. Data science, in the center of the diagram, covers different disciplines such as DM, SE, and statistics.

This study presents a comprehensive literature review of existing research and offers an overview of how to approach SE problems using different mining techniques. Up to now, review studies either introduce SE data descriptions [1], explain tools and techniques mostly used by researchers for SE data analysis [2], discuss the role of software engineers [3], or focus only on a specific problem in SE such as defect prediction [4], design pattern [5], or effort estimation [6]. Some existing review articles having the same target [7] are former, and some of them are not

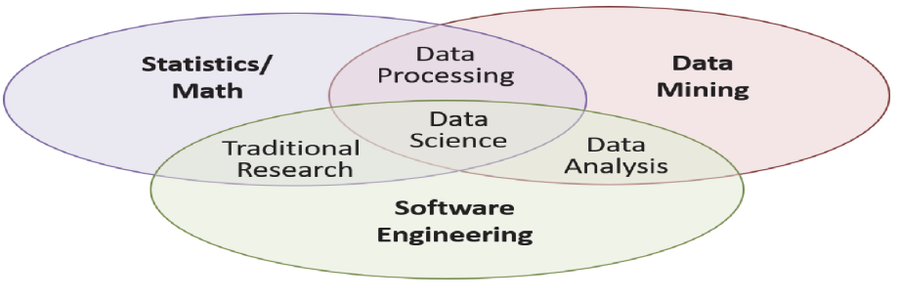

Figure 1.
The intersection of data mining and software engineering with other areas of the field.

comprehensive. In contrast to the previous studies, this article provides a systematic review of several SE tasks, gives a comprehensive list of available studies in the field, clearly states the advantages of mining SE data, and answers "how" and "why" questions in the research area.

The novelties and main contributions of this review paper are fivefold.

- First, it provides a general overview of several SE tasks that have been the focus of studies using DM and ML, namely, defect prediction, effort estimation, vulnerability analysis, refactoring, and design pattern mining.

- Second, it comprehensively discusses existing data mining solutions in software engineering according to various aspects, including methods (clustering, classification, association rule mining, etc.), algorithms (k-nearest neighbor (KNN), neural network (NN), etc.), and performance metrics (accuracy, mean absolute error, etc.).

- Third, it points to several significant research questions that are unanswered in the recent literature as a whole or the answers to which have changed with the technological developments in the field.

- Fourth, some statistics related to the studies between the years of 2010 and 2019 are given from different perspectives: according to their subjects and according to their methods.

- Five, it focuses on different machine learning types: supervised and unsupervised learning, especially on ensemble learning and deep learning.

This paper addresses the following research questions:

RQ1. What kinds of SE problems can ML and DM techniques help to solve?

RQ2. What are the advantages of using DM techniques in SE?

RQ3. Which DM methods and algorithms are commonly used to handle SE tasks?

RQ4. Which performance metrics are generally used to evaluate DM models constructed in SE studies?

RQ5. Which types of machine learning techniques (e.g., ensemble learning, deep learning) are generally preferred for SE problems?

RQ6. Which SE datasets are popular in DM studies?

The remainder of this paper is organized as follows. Section 2 explains the knowledge discovery process that aims to extract interesting, potentially useful, and nontrivial information from software engineering data. Section 3 provides an overview of current work on data mining for software engineering grouped under five tasks: defect prediction, effort estimation, vulnerability analysis, refactoring, and

design pattern mining. In addition, some machine learning studies are divided into subgroups, including ensemble learning- and deep learning-based studies. Section 4 gives statistical information about the number of highly validated research conducted in the last decade. Related works considered as fundamental by journals with a highly positive reputation are listed, and the specific methods they used and their categories and purposes are clearly expressed. In addition, widely used datasets related to SE are given. Finally, Section 5 offers concluding remarks and suggests future scientific and practical efforts that might improve the efficiency of SE actions.

2. Knowledge discovery from software engineering data

This section basically explains the consecutive critical steps that should be followed to discover beneficial knowledge from software engineering data. It outlines the order of necessary operations in this process and explains how related data flows among them.

Software development life cycle (SDLC) describes a process to improve the quality of a product in project management. The main phases of SDCL are planning, requirement analysis, designing, coding, testing, and maintenance of a project. In every phase of software development, some software problems (e.g., software bugs, security, or design problems) may occur. Correcting these problems in the early phases leads to more accurate and timely delivery of the project. Therefore, software engineers broadly apply data mining techniques for different SE tasks to solve SE problems and to enhance programming efficiency and quality.

Figure 2 presents the data mining and knowledge discovery process of SE tasks including data collection, data preprocessing, data mining, and evaluation. In the data collection phase, data are obtained from software projects such as bug reports, historical data, version control data, and mailing lists that include various information about the project's versions, status, or improvement. In the data preprocessing phase, the data are preprocessed after collection by using different methods such as feature selection (dimensionality reduction), feature extraction, missing data elimination, class imbalance analysis, normalization, discretization, and so on. In the next phase, DM techniques such as classification, clustering, and association rule mining are applied to discover useful patterns and relationships in software

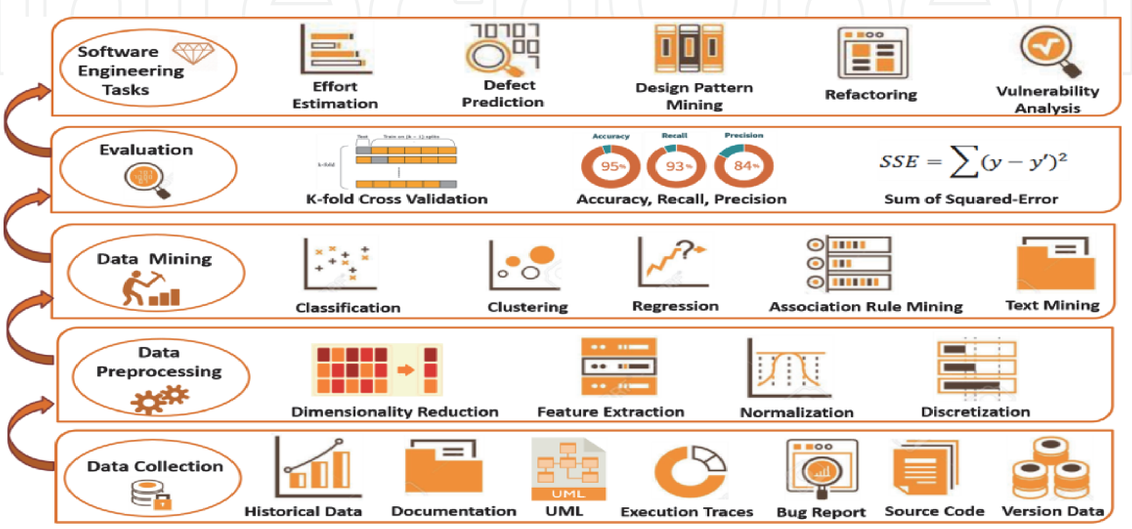

Figure 2.
KDD process for software engineering.

engineering data and therefore to solve a software engineering problem such as defected or vulnerable systems, reused patterns, or parts of code changes. Mining and obtaining valuable knowledge from such data prevents errors and allows software engineers to deliver the project on time. Finally, in the evaluation phase, validation techniques are used to assess the data mining results such as k-fold cross validation for classification. The commonly used evaluation measures are accuracy, precision, recall, F-score, area under the curve (AUC) for classification, and sum of squared errors (SSE) for clustering.

3. Data mining in software engineering

In this review, we examine data mining studies in various SE tasks and evaluate commonly used algorithms and datasets.

3.1 Data mining in defect prediction

A defect means an error, failure, flaw, or bug that causes incorrect or unexpected results in a system [8]. A software system is expected to be without any defects since software quality represents a capacity of the defect-free percentage of the product [9]. However, software projects often do not have enough time or people working on them to extract errors before a product is released. In such a situation, defect prediction methods can help to detect and remove defects in the initial stages of the SDLC and to improve the quality of the software product. In other words, the goal of defect prediction is to produce robust and effective software systems. Hence, software defect prediction (SDP) is an important topic for software engineering because early prediction of software defects could help to reduce development costs and produce more stable software systems.

Various studies have been conducted on defect prediction using different metrics such as code complexity, history-based metrics, object-oriented metrics, and process metrics to construct prediction models [10, 11]. These models can be considered on a cross-project or within-project basis. In within-project defect prediction (WPDP), a model is constructed and applied on the same project [12]. For within-project strategy, a large amount of historical defect data is needed. Hence, in new projects that do not have enough data to train, cross-project strategy may be preferred [13]. Cross-project defect prediction (CPDP) is a method that involves applying a prediction model from one project to another, meaning that models are prepared by utilizing historical data from other projects [14, 15]. Studies in the field of CPDP have increased in recent years [10, 16]. However, there are some deficiencies in comparisons of prior studies since they cannot be replicated because of the difference in utilizing evaluation metrics or preparation way of training data. Therefore, Herbold et al. [16] tried to replicate different CPDP methods previously proposed and find which approach performed best in terms of metrics such as F-score, area under the curve (AUC), and Matthews correlation coefficient (MCC). Results showed that 7- or 8-year approaches may perform better. Another study [17] replicated prior work to demonstrate whether the determination of classification techniques is important. Both noisy and cleaned datasets were used, and the same results were obtained from the two datasets. However, new dataset gave better results for some classification algorithms. For this reason, authors claimed that the selection of classification techniques affects the performance of the model.

Numerous defect prediction studies have been conducted using DM techniques. In the following subsections, we will explain these studies in terms of whether they apply ensemble learning or not. Some defect prediction studies in SE are compared

Ref.	Year	Task	Objective	Algorithms	Ensemble learning	Dataset	Evaluation metrics and results
[18]	2011	Classification	Comparative study of various ensemble methods to find the most effective one	NB	Bagging, boosting, RT, **RF**, RS, AdaBoost, Stacking, and **Voting**	NASA datasets: CM1 JM1 KC1 KC2 KC3 KC4 MC1 MC2 MW1 PC1 PC2 PC3 PC4 PC5	10-fold CV, ACC, and AUC *Vote* 88.48% *random forest* 87.90%
[19]	2013	Classification	Comparative study of class imbalance learning methods and proposed dynamic version of AdaBoost.NC	NB, RUS, RUS-bal, THM, SMB, BNC	RF, SMB, BNC, **AdaBoost.NC**	NASA and PROMISE repository: MC2, KC2, JM1, KC1, PC4, PC3, CM1, KC3, MW1, PC1	10-fold CV Balance, G-mean and AUC, PD, PF
[20]	2014	Classification	Comparative study to deal with imbalanced data	Base Classifiers: C4.5, NB Sampling: ROS, RUS, SMOTE	AdaBoost, Bagging, boosting, RF	NASA datasets: CM1, JM1, KC1, KC2, KC3, MC1, MC2, MW1, PC1, PC2, PC3, PC4, PC5	5 × 5 CV, MCC, ROC, results change according to characteristics of datasets
[17]	2015	Clustering/ classification	To show that the selection of classification technique has an impact on the performance of software defect prediction models	Statistical: NB, **Simple Logistic** Clustering: KM, EM Rule based: Ripper, Ridor NNs: RBF Nearest neighbor: KNN DTs: J48, **LMT**	Bagging, AdaBoost, rotation forest, random subspace	NASA: CM1, JM1, KC1, KC3, KC4, MW1, PC1, PC2, PC3, PC4 PROMISE: Ant 1.7, Camel 1.6, Ivy 1.4, Jedit 4, Log4j 1, Lucene 2.4, Poi 3, Tomcat 6, Xalan 2.6, Xerces 1.3	10 × 10-fold CV AUC > 0.5 Scott-Knott test α = 0.05, simple logistic, LMT, and RF + base learner outperforms KNN and RBF
[21]	2015	Classification	Average probability ensemble (APE) learning module is proposed by combining feature selection and ensemble learning	**APE** system combines seven classifiers: SGD, weighted SVMs (W-SVMs), LR, MNB and Bernoulli naive Bayes (BNB)	RF, GB	NASA: CM1, JM1, KC1, KC3, KC4, MW1, PC1, PC2, PC3, PC4 PROMISE (RQ2): Ant 1.7, Camel 1.6, Ivy 1.4, Jedit 4, Log4j 1, Lucene 2.4, Poi 3, Tomcat 6, Xalan 2.6, Xerces 1.3	10 × 10-fold CV, AUC > 0.5 Scott-Knott test α = 0.05, simple logistic, LMT, and RF + base learner outperforms KNN and RBF
[22, 23]	2016	Classification	Comparative study of 18 ML techniques using OO metrics on six releases of Android operating system	LR, **NB**, BN, **MLP**, RBF SVM, VP, CART, J48, ADT, Nnge, DTNB	Bagging, random forest, Logistic model trees, **Logit Boost**, Ada Boost	6 releases of Android app: Android 2.3.2, Android 2.3.7, Android 4.0.4, Android 4.1.2, Android 4.2.2, Android 4.3.1	10-fold, inter-release validation AUC for NB, LB, MLP is >0.7
[24]	2016	Classification	Caret has been applied whether parameter settings can have a	NB, KNN, LR, partial least squares, NN, LDA, rule based, DT, SVM	Bagging, boosting	Cleaned NASA JM1, PC5 Proprietary from Prop-1 to Prop-5	Out-of-sample bootstrap validation technique, AUC

Ref.	Year	Task	Objective	Algorithms	Ensemble learning	Dataset	Evaluation metrics and results
			large impact on the performance of defect prediction models			Apache Camel 1.2, Xalan 2.5–2.6 Eclipse Platform 2.0–2.1–3.0, Debug 3.4, SWT 3.4, JDT, Mylyn, PDE	Caret AUC performance up to 40 percentage points
[25]	2017	Regression	Aim is to validate the source code metrics and identify a suitable set of source code metrics	5 training algorithms: GD, GDM, GDX, NM, LM	Heterogeneous linear and nonlinear ensemble methods	56 open-source Java projects from PROMISE Repository	10-fold CV, t-test, ULR analysis Neural network with Levenberg Marquardt (LM) is the best
[16]	2017	Classification	Replicate 24 CDPD approaches, and compare on 5 different datasets	DT, LR, NB, SVM	LE, RF, BAG-DT, BAG-NB, BOOST-DT, BOOST-NB	5 available datasets: JURECZKO, NASA MDP, AEEEM, NETGENE, RELINK	Recall, PR, ACC, G-measure, F-score, MCC, AUC
[26]	2017	Classification	Just-in-time defect prediction (TLEL)	NB, SVM, DT, LDA, NN	Bagging, stacking	Bugzilla, Columba, JDT, Platform, Mozilla, and PostgreSQL	10-fold CV, F-score
[13]	2017	Classification	Adaptive Selection of Classifiers in bug prediction (ASCI) method is proposed.	Base classifiers: LOG (binary logistic regression), NB, RBF, MLP, DT	Voting	Ginger Bread (2.3.2 and 2.3.7), Ice Cream Sandwich (4.0.2 and 4.0.4), and JellyBean (4.1.2, 4.2.2 and 4.3.1)	10-fold, inter-release validation AUC for NB, LB, MLP is >0.7
[27]	2018	Classification	MULTI method for JIT-SDP (just in time software defect prediction)	EALR, SL, RBFNet Unsupervised: LT, AGE	Bagging, AdaBoost, Rotation Forest, RS	Bugzilla, Columba, Eclipse JDT, Eclipse Platform, Mozilla, PostgreSQ	CV, timewise-CV, ACC, and P_{OPT} MULTI performs significantly better than all the baselines
[28]	2007	Classification	To found pre- and post-release defects for every package and file	LR	—	Eclipse 2.0, 2.1, 3.0	PR, recall, ACC
[8]	2014	Clustering	Cluster ensemble with PSO for clustering the software modules (fault-prone or not fault-prone)	PSO clustering algorithm	KM-E, KM-M, PSO-E, PSO-M and EM	Nasa MDP, PROMISE	

Ref.	Year	Task	Objective	Algorithms	Ensemble learning	Dataset	Evaluation metrics and results
[29]	2015	Classification	Defect identification by applying DM algorithms	NB, J48, MLP	—	PROMISE, NASA MDP dataset: CM1, JM1, KC1, KC3, MC1, MC2, MW1, PC1, PC2, PC3	10-fold CV, ACC, PR, FMLP is the best
[30]	2015	Classification	To show the attributes that predict the defective state of software modules	NB, NN, association rules, DT	Weighted voting rule of the four algorithms	NASA datasets: CM1, JM1, KC1, KC2, PC1	PR, recall, ACC, F-score NB > NN > DT
[31]	2016	Classification	Authors proposed a model that finds fault-proneness	NB, LR, LivSVM, MLP, SGD, SMO, VP, LR Logit Boost, Decision Stamp, RT, REP Tree	RF	Camel1.6, Tomcat 6.0, Ant 1.7, jEdit4.3, Ivy 2.0, arc, e-learning, berek, forrest 0.8, zuzel, Intercafe, and Nieruchomosci	10-fold CV, AUC AUC = 0.661
[32]	2016	Classification	GA to select suitable source code metrics	LR, ELM, SVML, SVMR, SVMP	—	30 open-source software projects from PROMISE repository from DS1 to DS30	5-fold CV, F-score, ACC, pairwise t-test
[33]	2016	—	Weighted least-squares twin support vector machine (WLSTSVM) to find misclassification cost of DP	SVM, NB, RF, LR, KNN, BN, cost-sensitive neural network	—	PROMISE repository: CM1, KC1, PC1, PC3, PC4, MC2, KC2, KC3	10-fold CV, PR, recall, F-score, G-mean Wilcoxon signed rank test
[34]	2016	—	A multi-objective naive Bayes learning techniques MONB, MOBNN	NB, LR, DT, MODT, MOLR, MONB	—	Jureczko datasets obtained from PROMISE repository	AUC, Wilcoxon rank test CP MO NB (0.72) produces the highest value
[35]	2016	Classification	A software defect prediction model to find faulty components of a software	Hybrid filter approaches FISHER, MR, ANNIGMA.	—	KC1, KC2, JM1, PC1, PC2, PC3, and PC4 datasets	ACC, ent filters, ACC 90%
[36]	2017	Classification	Propose an hybrid method called TSC-RUS + S	A random undersampling based on two-step cluster (TSC)	Stacking: DT, LR, kNN, NB	NASA MDP: i.e., CM1, KC1, KC3, MC2, MW1, PC1, PC2, PC3, PC4	10-fold CV, AUC, (TSC-RUS + S) is the best
[37]	2017	Classification	Analyze five popular ML algorithms for software defect prediction	ANN, PSO, DT, NB, LC	—	Nasa and PROMISE datasets: CM1, JM1, KC1, KC2, PC1, KC1-LC	10-fold CV ANN < DT

Ref.	Year	Task	Objective	Algorithms	Ensemble learning	Dataset	Evaluation metrics and results
[38]	2018	Classification	Three well-known ML techniques are compared.	NB, DT, ANN	—	Three different datasets DS1, DS2, DS3	ACC, PR, recall, F, ROC ACC 97% DT > ANN > NB
[10]	2018	Classification	ML algorithms are compared with CODEP	LR, BN, RBF, MLP, alternating decision tree (ADTree), and DT	Max, CODEP, Bagging J48, Bagging NB, Boosting J48, Boosting NB, RF	PROMISE: Ant, Camel, ivy, Jedit, Log4j, Lucene, Poi, Prop, Tomcat, Xalan	F-score, PR, AUC ROC Max performs better than CODEP

Table 1.
Data mining and machine learning studies on the subject "defect prediction."

in **Table 1**. The objective of the studies, the year they were conducted, algorithms, ensemble learning techniques and datasets in the studies, and the type of data mining tasks are shown in this table. The bold entries in **Table 1** have better performance than other algorithms in that study.

3.1.1 Defect prediction using ensemble learning techniques

Ensemble learning combines several base learning models to obtain better performance than individual models. These base learners can be acquired with:

i. Different learning algorithms

ii. Different parameters of the same algorithm

iii. Different training sets

The commonly used ensemble techniques bagging, boosting, and stacking are shown in **Figure 3** and briefly explained in this part. Bagging (which stands for bootstrap aggregating) is a kind of parallel ensemble. In this method, each model is built independently, and multiple training datasets are generated from the original dataset through random selection of different feature subsets; thus, it aims to decrease variance. It combines the outputs of each ensemble member by a voting mechanism. Boosting can be described as sequential ensemble. First, the same weights are assigned to data instances; after training, the weight of wrong predictions is increased, and this process is repeated as the ensemble size. Finally, it uses a weighted voting scheme, and in this way, it aims to decrease bias. Stacking is a technique that uses predictions from multiple models via a meta-classifier.

Some software defect prediction studies have compared ensemble techniques to determine the best performing one [10, 18, 21, 39, 40]. In a study conducted by Wang et al. [18], different ensemble techniques such as bagging, boosting, random tree, random forest, random subspace, stacking, and voting were compared to each other and a single classifier (NB). According to the results, voting and random forest clearly exhibited better performance than others. In a different study [39],

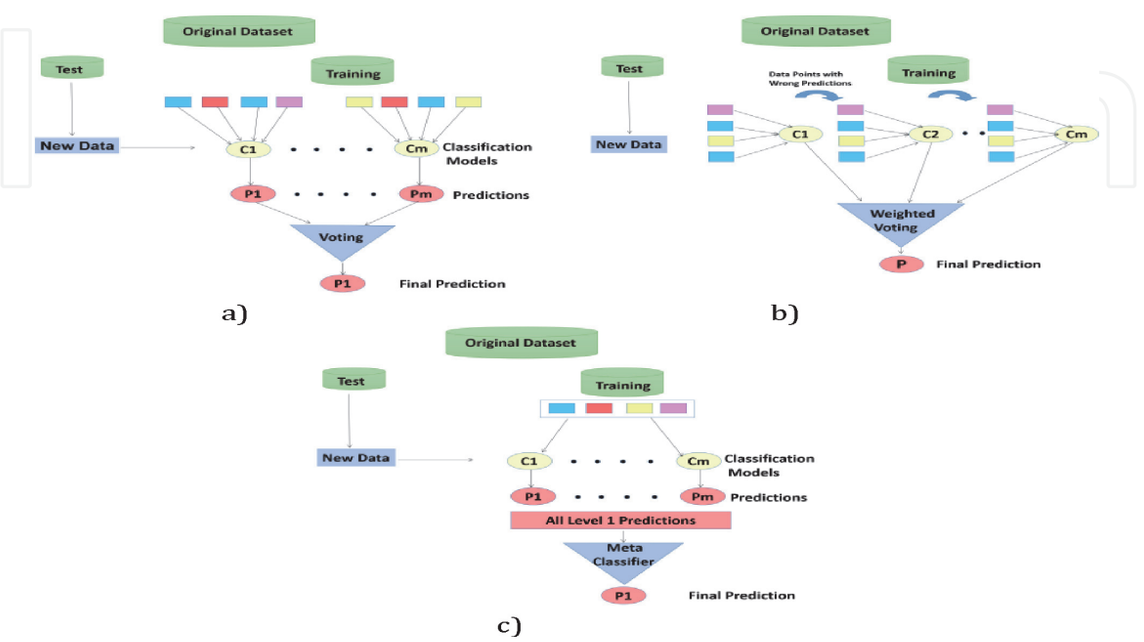

Figure 3.
Common ensemble learning methods: (a) Bagging, (b) boosting, (c) stacking.

ensemble methods were compared with more than one base learner (NB, BN, SMO, PART, J48, RF, random tree, IB1, VFI, DT, NB tree). For boosted SMO, bagging J48, and boosting and bagging RT, performance of base classifiers was lower than that of ensemble learner classifiers.

In study [21], a new method was proposed of mixing feature selection and ensemble learning for defect classification. Results showed that random forests and the proposed algorithm are not affected by poor features, and the proposed algorithm outperforms existing single and ensemble classifiers in terms of classification performance. Another comparative study [10] used seven composite algorithms (Ave, Max, Bagging C4.5, bagging naive Bayes (NB), Boosting J48, Boosting naive Bayes, and RF) and one composite state-of-the art study for cross-project defect prediction. The Max algorithm yielded the best results regarding F-score in terms of classification performance.

Bowes et al. [40] compared RF, NB, Rpart, and SVM algorithms to determine whether these classifiers obtained the same results. The results demonstrated that a unique subset of defects can be discovered by specific classifiers. However, whereas some classifiers are steady in the predictions they make, other classifiers change in their predictions. As a result, ensembles with decision-making without majority voting can perform best.

One of the main problems of SDP is the imbalance between the defect and non-defect classes of the dataset. Generally, the number of defected instances is greater than the number of non-defected instances in the collected data. This situation causes the machine learning algorithms to perform poorly. Wang and Yao [19] compared five class-imbalanced learning methods (RUS, RUS-bal, THM, BNC, SMB) and NB and RF algorithms and proposed the dynamic version of AdaBoost. NC. They utilized balance, G-mean, and AUC measures for comparison. Results showed that AdaBoost.NC and naive Bayes are better than the other seven algorithms in terms of evaluation measures. Dynamic AdaBoost.NC showed better defect detection rate and overall performance than the original AdaBoost.NC. To handle the class imbalance problem, studies [20] have compared different methods (sampling, cost sensitive, hybrid, and ensemble) by taking into account evaluation metrics such as MCC and receiver operating characteristic (ROC).

As shown in **Table 1**, the most common datasets used in the defect prediction studies [17–19, 39] are the NASA MDP dataset and PROMISE repository datasets. In addition, some studies utilized open-source projects such as Bugzilla Columba and Eclipse JDT [26, 27], and other studies used Android application data [22, 23].

3.1.2 Defect prediction studies without ensemble learning

Although use of ensemble learning techniques has dramatically increased recently, studies that do not use ensemble learning are still conducted and successful. For example, in study [32], prediction models were created using source code metrics as in ensemble studies but by using different feature selection techniques such as genetic algorithm (GA).

To overcome the class imbalance problem, Tomar and Agarwal [33] proposed a prediction system that assigns lower cost to non-defective data samples and higher cost to defective samples to balance data distribution. In the absence of enough data within a project, required data can be obtained from cross projects; however, in this case, this situation may cause class imbalance. To solve this problem, Ryu and Baik [34] proposed multi-objective naïve Bayes learning for cross-project environments. To obtain significant software metrics on cloud computing environments, Ali et al. used a combination of filter and wrapper approaches [35]. They compared different machine learning algorithms such as NB, DT, and MLP [29, 37, 38, 41].

3.2 Data mining in effort estimation

Software effort estimation (SEE) is critical for a company because hiring more employees than required will cause loss of revenue, while hiring fewer employees than necessary will result in delays in software project delivery. The estimation analysis helps to predict the amount of effort (in person hours) needed to develop a software product. Basic steps of software estimation can be itemized as follows:

- Determine project objectives and requirements.
- Design the activities.
- Estimate product size and complexity.
- Compare and repeat estimates.

SEE contains requirements and testing besides predicting effort estimation [42]. Many research and review studies have been conducted in the field of SEE. Recently, a survey [43] analyzed effort estimation studies that concentrated on ML techniques and compared them with studies focused on non-ML techniques. According to the survey, case-based reasoning (CBR) and artificial neural network (ANN) were the most widely used techniques. In 2014, Dave and Dutta [44] examined existing studies that focus only on neural network.

The current effort estimation studies using DM and ML techniques are available in **Table 2**. This table summarizes the prominent studies in terms of aspects such as year, data mining task, aim, datasets, and metrics. **Table 2** indicates that neural network is the most widely used technique for the effort estimation task.

Several studies have compared ensemble learning methods with single learning algorithms [45, 46, 48, 49, 51, 60] and examined them on cross-company (CC) and within-company (WC) datasets [50]. The authors observed that ensemble methods obtained by a proper combination of estimation methods achieved better results than single methods. Various ML techniques such as neural network, support vector machine (SVM), and k-nearest neighbor are commonly used as base classifiers for ensemble methods such as bagging and boosting in software effort estimation. Moreover, their results indicate that CC data can increase performance over WC data for estimation techniques [50].

In addition to the abovementioned studies, researchers have conducted studies without using ensemble techniques. The general approach is to investigate which DM technique has the best effect on performance in software effort estimation. For instance, Subitsha and Rajan [54] compared five different algorithms—MLP, RBFNN, SVM, ELM, and PSO-SVM—and Nassif et al. [57] investigated four neural network algorithms—MLP, RBFNN, GRNN, and CCNN. Although neural networks are widely used in this field, missing values and outliers frequently encountered in the training set adversely affect neural network results and cause inaccurate estimations. To overcome this problem, Khatibi et al. [53] split software projects into several groups based on their similarities. In their studies, the C-means clustering algorithm was used to determine the most similar projects and to decrease the impact of unrelated projects, and then analogy-based estimation (ABE) and NN were applied. Another clustering study by Azzeh and Nassif [59] combined SVM and bisecting k-medoids clustering algorithms; an estimation model was then built using RBFNN. The proposed method was trained on historical use case points (UCP).

Ref.	Year	Task	Objective	Algorithms	Ensemble learning	Dataset	Evaluation metrics and results
[45]	2008	Regression	Ensemble of neural networks with associative memory (ENNA)	NN, MLP, KNN	Bagging	NASA, NASA 93, USC, SDR, Desharnais	MMRE, MdMRE and PRED(L) For ENNA PRED(25) = 36.4 For neural network PRED(25) = 8
[46]	2009	Regression	Authors proposed the ensemble of neural networks with associative memory (ENNA)	NN, MLP, KNN	Bagging	NASA, NASA 93, USC, SDR, Desharnais	Random subsampling, t-test MMRE, MdMRE, and PRED(L) ENNA is the best
[47]	2010	Regression	To show the effectiveness of SVR for SEE	SVR, RBF	—	Tukutuku	LOOCV, MMRE, Pred(25), MEMRE, MdEMRE SVR outperforms others
[48]	2011	Regression	To evaluate whether readily available ensemble methods enhance SEE	MLP, RBF, RT	Bagging	5 datasets from PROMISE: cocomo81, nasa93, nasa, sdr, and Desharnais 8 datasets from ISBSG repository	MMRE, MdMRE, PRED(25) RTs and Bagging with MLPs perform similarly
[49]	2012	Regression	To show the measures behave in SEE and to create good ensembles	MLP, RBF, REPTree,	Bagging	cocomo81, nasa93, nasa, cocomo2, desharnais, ISBSG repository	MMRE, PRED(25), LSD, MdMRE, MAE, MdAE Pareto ensemble for all measures, except LSD.
[50]	2012	Regression	To use cross-company models to create diverse ensembles able to dynamically adapt to changes	WC RTs, CC-DWM	WC-DWM	3 datasets from ISBSG repository (ISBSG2000, ISBSG2001, ISBSG) 2 datasets from PROMISE (CocNasaCoc81 and CocNasaCoc81Nasa93)	MAE, Friedman test Only DCL could improve upon RT CC data potentially beneficial for improving SEE
[51]	2012	Regression	To generate estimates from ensembles of multiple prediction methods	CART, NN, LR, PCR, PLSR, SWR, ABE0-1NN, ABE0-5NN	Combining top M solo methods	PROMISE	MAR, MMRE, MdMRE, MMER, MBRE, MIBRE. Combinations perform better than 83%
[52]	2012	Classification/ regression	DM techniques to estimate software effort.	M5, CART, LR, MARS, MLPNN, RBFNN, SVM	—	Coc81, CSC, Desharnais, Cocnasa, Maxwell, USP05	MdMRE, Pred(25), Friedman test Log + OLS > LMS, BC + OLS, MARS, LS-SVM

Ref.	Year	Task	Objective	Algorithms	Ensemble learning	Dataset	Evaluation metrics and results
[53]	2013	Clustering/ classification	Estimation of software development effort	NN, ABE, C-means	—	Maxwell	3-fold CV and LOOCV, RE, MRE, MMRE, PRED
[54]	2014	Regression	ANNs are examined using COCOMO model	MLP, RBFNN, SVM, PSO-SVM Extreme learning Machines	—	COCOMO II Data	MMRE, PRED PSO-SVM is the best
[55]	2014	—	A hybrid model based on GA And ACO for optimization	GA, ACO	—	NASA datasets	MMRE, the proposed method is the best
[56]	2015	Regression	To display the effect of data preprocessing techniques on ML methods in SEE	CBR, ANN, CART Preprocessing rech: MDT, LD, MI, FS, CS, FSS, BSS	—	ISBSG, Desharnais, Kitchenham, USPFT	CV, MBRE, PRED (0.25), MdBRE
[57]	2016	Regression	Four neural network models are compared with each other.	MLP, RBFNN, GRNN, CCNN	—	ISBSG repository	10-fold CV, MAR The CCNN outperforms the other three models
[58]	2016	Regression	To propose a model based on Bayesian network	GA and PSO	—	COCOMO NASA Dataset	DIR, DRM The proposed model is best
[59]	2016	Classification/ regression	A hybrid model using SVM and RBNN compared against previous models	SVM, RBNN	—	Dataset1 = 45 industrial projects Dataset2 = 65 educational projects	LOOCV, MAE, MBRE, MIBRE, SA The proposed approach is the best
[60]	2017	Classification	To estimate software effort by using ML techniques	SVM, KNN	Boosting: kNN and SVM	Desharnais, Maxwell	LOOCV, k-fold CV ACC = 91.35% for Desharnais ACC = 85.48% for Maxwell

Table 2.
Data mining and machine learning studies on the subject "effort estimation."

Zare et al. [58] and Maleki et al. [55] utilized optimization methods for accurate cost estimation. In the former study, a model was proposed based on Bayesian network with genetic algorithm and particle swarm optimization (PSO). The latter study used GA to optimize the effective factors' weight, and then trained by ant colony optimization (ACO). Besides conventional effort estimation studies, researchers have utilized machine learning techniques for web applications. Since web-based software projects are different from traditional projects, the effort estimation process for these studies is more complex.

It is observed that PRED(25) and MMRE are the most popular evaluation metrics in effort estimation. MMRE stands for the mean magnitude relative error, and PRED(25) measures prediction accuracy and provides a percentage of predictions within 25% of actual values.

3.3 Data mining in vulnerability analysis

Vulnerability analysis is becoming the focal point of system security to prevent weaknesses in the software system that can be exploited by an attacker. Description of software vulnerability is given in many different resources in different ways [61]. The most popular and widely utilized definition appears in the Common Vulnerabilities and Exposures (CVE) 2017 report as follows:

Vulnerability is a weakness in the computational logic found in software and some hardware components that, when exploited, results in a negative impact to confidentiality, integrity or availability.

Vulnerability analysis may require many different operations to identify defects and vulnerabilities in a software system. Vulnerabilities, which are a special kind of defect, are more critical than other defects because attackers exploit system vulnerabilities to perform unauthorized actions. A defect is a normal problem that can be encountered frequently in the system, easily found by users or developers and fixed promptly, whereas vulnerabilities are subtle mistakes in large codes [62, 63]. Wijayasekara et al. claim that some bugs have been identified as vulnerabilities after being publicly announced in bug databases [64]. These bugs are called "hidden impact vulnerabilities" or "hidden impact bugs." Therefore, the authors proposed a hidden impact vulnerability identification methodology that utilizes text mining techniques to determine which bugs in bug databases are vulnerabilities. According to the proposed method, a bug report was taken as input, and it produces feature vector after applying text mining. Then, classifier was applied and revealed whether it is a bug or a vulnerability. The results given in [64] demonstrate that a large proportion of discovered vulnerabilities were first described as hidden impact bugs in public bug databases. While bug reports were taken as input in that study, in many other studies, source code is taken as input. Text mining is a highly preferred technique for obtaining features directly from source codes as in the studies [65–69]. Several studies [63, 70] have compared text mining-based models and software metrics-based models.

In the security area of software systems, several studies have been conducted related to DM and ML. Some of these studies are compared in **Table 3**, which shows the data mining task and explanation of the studies, the year they were performed, the algorithms that were used, the type of vulnerability analysis, evaluation metrics, and results. In this table, the best performing algorithms according to the evaluation criteria are shown in bold.

Vulnerability analysis can be categorized into three types: static vulnerability analysis, dynamic vulnerability analysis, and hybrid analysis [61, 80]. Many studies have applied the static analysis approach, which detects vulnerabilities from source code without executing software, since it is cost-effective. Few studies have

Ref.	Year	Task	Objective	Algorithms	Type	Dataset description	Evaluation metrics and results
[71]	2011	Clustering	Obtaining software vulnerabilities based on RDBC	RDBC	Static	Database is built by RD-Entropy	FNR, FPR
[42]	2011	Classification/ regression	To predict the time to next vulnerability	LR, LMS, MLP, RBF, SMO	Static	NVD, CPE, CVSS	CC, RMSE, RRSE
[65]	2012	Text mining	Analysis of source code as text	RBF, SVM	Static	K9 email client for the Android platform	ACC, PR, recall ACC = 0.87, PR = 0.85, recall = 0.88
[64]	2012	Classification/ text mining	To identify vulnerabilities in bug databases	—	Static	Linux kernel MITRE CVE and MySQL bug databases	BDR, TPR, FPR 32% (Linux) and 62% (MySQL) of vulnerabilities
[72]	2014	Classification/ regression	Combine taint analysis and data mining to obtain vulnerabilities	ID3, C4.5/J48, RF, RT, KNN, NB, Bayes Net, MLP, SVM, LR	Hybrid	A version of WAP to collect the data	10-fold CV, TPD, ACC, PR, KAPPA ACC = 90.8%, PR = 92%, KAPPA = 81%
[73]	2014	Clustering	Identify vulnerabilities from source codes using CPG	—	Static	Neo4J and InfiniteGraph databases	—
[63]	2014	Classification	Comparison of software metrics with text mining	RF	Static	Vulnerabilities from open-source web apps (Drupal, Moodle, PHPMyAdmin)	3-fold CV, recall, IR, PR, FPR, ACC. Text mining provides benefits overall
[69]	2014	Classification	To create model in the form of a binary classifier using text mining	NB, RF	Static	Applications from the F-Droid repository and Android	10-fold CV, PR, recall PR and recall ≥ 80%
[74]	2015	Classification	A new approach (VCCFinder) to obtain potentially dangerous codes	SVM-based detection model	—	The database contains 66 GitHub projects	k-fold CV, false alarms <99% at the same level of recall
[70]	2015	Ranking/ classification	Comparison of text mining and software metrics models	RF	—	Vulnerabilities from open-source web apps (Drupal, Moodle, PHPMyAdmin)	10-fold CV Metrics: ER-BCE, ERBPP, ER-AVG
[75]	2015	Clustering	Search patterns for taint-style vulnerabilities in C code	Hierarchical clustering (complete-linkage)	Static	5 open-source projects: Linux, OpenSSL, Pidgin, VLC, Poppler (Xpdf)	Correct source, correct sanitization, number of traversals, generation time, execution time, reduction, amount of code review <95%

Ref.	Year	Task	Objective	Algorithms	Type	Dataset description	Evaluation metrics and results
[76]	2016	Classification	Static and dynamic features for classification	LR, MLP, RF	Hybrid	Dataset was created by analyzing 1039 test cases from the Debian Bug Tracker	FPR, FNR Detect 55% of vulnerable programs
[77]	2017	Classification	1. Employ a deep neural network 2. Combine N-gram analysis and feature selection	Deep neural network	—	Feature extraction from 4 applications (BoardGameGeek, Connectbot, CoolReader, AnkiDroid)	10 times using 5-fold CV ACC = 92.87%, PR = 94.71%, recall = 90.17%
[67]	2017	Text mining	To analyze characteristics of software vulnerability from source files	—	—	CVE, CWE, NVD databases	PR = 70%, recall = 60%
[68]	2017	Text mining	Deep learning (LSTM) is used to learn semantic and syntactic features in code	RNN, LSTM, DBN	—	Experiments on 18 Java applications from the Android OS platform	10-fold CV, PR, recall, and F-score Deep Belief Network PR, recall, and F-score > 80%
[66]	2018	Classification	Identify bugs by extracting text features from C source code	NB, KNN, K-means, NN, SVM, DT, RF	Static	NVD, Cat, Cp, Du, Echo, Head, Kill, Mkdir, Nl, Paste, Rm, Seq, Shuf, Sleep, Sort, Tail, Touch, Tr, Uniq, Wc, Whoami	5-fold CV ACC, TP, TN ACC = 74%
[78]	2018	Regression	A deep learning-based vulnerability detection system (VulDeePecker)	BLSTM NN	Static	NIST: NVD and SAR project	10-fold CV, PR, recall, F-score F-score = 80.8%
[79]	2018	Classification	A mapping between existing requirements and vulnerabilities	LR, SVM, NB	—	Data is gathered from Apache Tomcat, CVE, requirements from Bugzilla, and source code is collected from Github	PR, recall, F-score LSI > SVM

Table 3.
Data mining and machine learning studies on the subject "vulnerability analysis."

performed the dynamic analysis approach, in which one must execute software and check program behavior. The hybrid analysis approach [72, 76] combines these two approaches.

As revealed in **Table 3**, in addition to classification and text mining, clustering techniques are also frequently seen in software vulnerability analysis studies. To detect vulnerabilities in an unknown software data repository, entropy-based density clustering [71] and complete-linkage clustering [75] were proposed. Yamaguchi et al. [73] introduced a model to represent a large number of source codes as a graph called control flow graph (CPG), a combination of abstract syntax tree, CFG, and program dependency graph (PDG). This model enabled the discovery of previously unknown (zero-day) vulnerabilities.

To learn the time to next vulnerability, a prediction model was proposed in the study [42]. The result could be a number that refers to days or a bin representing values in a range. The authors used regression and classification techniques for the former and latter cases, respectively.

In vulnerability studies, issue tracking systems like Bugzilla, code repositories like Github, and vulnerability databases such as NVD, CVE, and CWE have been utilized [79]. In addition to these datasets, some studies have used Android [65, 68, 69] or web [63, 70, 72] (PHP source code) datasets. In recent years, researchers have concentrated on deep learning for building binary classifiers [77], obtaining vulnerability patterns [78], and learning long-term dependencies in sequential data [68] and features directly from the source code [81].

Li et al. [78] note two difficulties of vulnerability studies: demanding, intense manual labor and high false-negative rates. Thus, the widely used evaluation metrics in vulnerability analysis are false-positive rate and false-negative rate.

3.4 Data mining in design pattern mining

During the past years, software developers have used design patterns to create complex software systems. Thus, researchers have investigated the field of design patterns in many ways [82, 83]. Fowler defines a pattern as follows:

> "*A pattern is an idea that has been useful in one practical context and will probably be useful in others.*" [84]

Patterns display relationships and interactions between classes or objects. Well-designed object-oriented systems have various design patterns integrated into them. Design patterns can be highly useful for developers when they are used in the right manner and place. Thus, developers avoid recreating methods previously refined by others. The pattern approach was initially presented in 1994 by four authors—namely, Erich Gama, Richard Helm, Ralph Johnson, and John Vlissides—called the Gang of Four (GOF) in 1994 [85]. According to the authors, there are three types of design patterns:

1. Creational patterns provide an object creation mechanism to create the necessary objects based on predetermined conditions. They allow the system to call appropriate object and add flexibility to the system when objects are created. Some creational design patterns are factory method, abstract factory, builder, and singleton.

2. Structural patterns focus on the composition of classes and objects to allow the establishment of larger software groups. Some of the structural design patterns are adapter, bridge, composite, and decorator.

3. Behavioral patterns determine common communication patterns between objects and how multiple classes behave when performing a task. Some behavioral design patterns are command, interpreter, iterator, observer, and visitor.

Many design pattern studies exist in the literature. **Table 4** shows some design pattern mining studies related to machine learning and data mining. This table contains the aim of the study, mining task, year, and design patterns selected by the study, input data, dataset, and results of the studies.

In design pattern mining, detecting the design pattern is a frequent study objective. To do so, studies have used machine learning algorithms [87, 89–91], ensemble learning [95], deep learning [97], graph theory [94], and text mining [86, 95].

In study [91], the training dataset consists of 67 object-oriented (OO) metrics extracted by using the JBuilder tool. The authors used LRNN and decision tree techniques for pattern detection. Alhusain et al. [87] generated training datasets from existing pattern detection tools. The ANN algorithm was selected for pattern instances. Chihada et al. [90] created training data from pattern instances using 45 OO metrics. The authors utilized SVM for classifying patterns accurately. Another metrics-oriented dataset was developed by Dwivedi et al. [93]. To evaluate the results, the authors benefited from three open-source software systems (JHotDraw, QuickUML, and JUnit) and applied three classifiers, SVM, ANN, and RF. The advantage of using random forest is that it does not require linear features and can manage high-dimensional spaces.

To evaluate methods and to find patterns, open-source software projects such as JHotDraw, Junit, and MapperXML have been generally preferred by researchers. For example, Zanoni et al. [89] developed a tool called MARPLE-DPD by combining graph matching and machine learning techniques. Then, to obtain five design patterns, instances were collected from 10 open-source software projects, as shown in **Table 4**.

Design patterns and code smells are related issues: Code smell refers to symptoms in code, and if there are code smells in a software, its design pattern is not well constructed. Therefore, Kaur and Singh [96] checked whether design pattern and smell pairs appear together in a code by using J48 Decision Tree. Their obtained results showed that the singleton pattern had no presence of bad smells.

According to the studies summarized in the table, the most frequently used patterns are abstract factory and adapter. It has recently been observed that studies on ensemble learning in this field are increasing.

3.5 Data mining in refactoring

One of the SE tasks most often used to improve the quality of a software system is refactoring, which Martin Fowler has described as "a technique for restructuring an existing body of code, altering its internal structure without changing its external behavior" [98]. It improves readability and maintainability of the source code and decreases complexity of a software system. Some of the refactoring types are: Add Parameter, Replace Parameter, Extract method, and Inline method [99].

Code smell and refactoring are closely related to each other: Code smells represent problems due to bad design and can be fixed during refactoring. The main challenge is to obtain which part of the code needs refactoring.

Some of data mining studies related to software refactoring are presented in **Table 5**. Some studies focus on historical data to predict refactoring [100] or to obtain both refactoring and software defects [101] using different data mining algorithms such as LMT, Rip, and J48. Results suggest that when refactoring

Ref.	Year	Task	Objective	Algorithms	EL	Selected design patterns	Input data	Dataset	Evaluation metrics and results
[86]	2012	Text classification	Two-phase method: 1—text classification to 2—learning design patterns	NB, KNN, DT, SVM	—	46 security patterns, 34 Douglass patterns, 23 GoF patterns	Documents	Security, Douglass, GoF	PR, recall, EWM PR = 0.62, recall = 0.75
[87]	2013	Regression	An approach is to find a valid instance of a DP or not	ANN	—	Adapter, command, composite, decorator, observer, and proxy	Set of candidate classes	JHotDraw 5.1 open-source application	10 fold CV, PR, recall
[88]	2014	Graph mining	Sub-graph mining-based approach	CloseGraph	—	—	Java source code	Open-source project:YARI, Zest, JUnit, JFreeChart, ArgoUML	No any empirical comparison
[89]	2015	Classification/ clustering	MARPLE-DPD is developed to classify instances whether it is a bad or good instance	SVM, DT, RF, K-means, ZeroR, OneR, NB, JRip, CLOPE.	—	Classification for singleton and adapter Classification and clustering for composite, decorator, and factory method	—	10 open-source software systems DPExample, QuickUML 2001, Lexi v0.1.1 alpha, JRefactory v2.6.24, Netbeans v1.0.x, JUnit v3.7, JHotDraw v5.1, MapperXML v1.9.7, Nutch v0.4, PMD v1.8	10-fold CV, ACC, F-score, AUC ACC > =85%
[90]	2015	Regression	A new method (SVM-PHGS) is proposed	Simple Logistic, C4.5, KNN, SVM, SVM-PHGS	—	Adapter, builder, composite, factory method, iterator, observer	Source code	P-mart repository	PR, recall, F-score, FP PR = 0.81, recall =0.81, F-score = 0.81, FP = 0.038
[91]	2016	Classification	Design pattern recognition using ML algorithms.	LRNN, DT	—	Abstract factory, adapter patterns	Source code	Dataset with 67 OO metrics, extracted by JBuilder tool	5-fold CV, ACC, PR, recall, F-score ACC = 100% by LRNN

Ref.	Year	Task	Objective	Algorithms	EL	Selected design patterns	Input data	Dataset	Evaluation metrics and results
[92]	2016	Classification	Three aspects: design patterns, software metrics, and supervised learning methods	Layer Recurrent Neural Network (LRNN)	RF	Abstract factory, adapter, bridge, singleton, and template method	Source code	Dataset with 67 OO metrics, extracted by JBuilder tool	PR, recall, F-score F-score = 100% by LRNN and RF ACC = 100% by RF
[93]	2017	Classification	1. Creation of metrics-oriented dataset 2. Detection of software design patterns	ANN, SVM	RF	Abstract factory, adapter, bridge, composite, and Template	Source code	Metrics extracted from source codes (JHotDraw, QuickUML, and Junit)	5-fold and 10-fold CV, PR, recall, F-score ANN, SVM, and RF yielded to 100% PR for JHotDraw
[94]	2017	Classification	Detection of design motifs based on a set of directed semantic graphs	Strong graph simulation, graph matching	—	All three groups: creational, structural, behavioral	UML class diagrams	—	PR, recall High accuracy by the proposed method
[95]	2017	Text categorization	Selection of more appropriate design patterns	Fuzzy c-means	Ensemble-IG	Various design patterns	Problem definitions of design patterns	DP, GoF, Douglass, Security	F-score
[96]	2018	Classification	Finding design pattern and smell pairs which coexist in the code	J48	—	Used patterns: adapter, bridge, Template, singleton	Source code	Eclipse plugin Web of Patterns The tool selected for code smell detection is iPlasma	PR, recall, F-score, PRC, ROC Singleton pattern shows no presence of bad smells

Table 4.
Data mining and machine learning studies on the subject "design pattern mining."

Ref.	Year	Task	Objective	Algorithms	EL	Dataset	Evaluation metrics and results
[100]	2007	Regression	Stages: (1) data understanding, (2) preprocessing, (3) ML, (4) post-processing, (5) analysis of the results	J48, LMT, Rip, NNge	—	ArgoUML, Spring Framework	10-fold CV, PR, recall, F-score PR and recall are 0.8 for ArgoUML
[101]	2008	Classification	Finding the relationship between refactoring and defects	C4.5, LMT, Rip, NNge	—	ArgoUML, JBoss Cache, Liferay Portal, Spring Framework, XDoclet	PR, recall, F-score
[102]	2014	Regression	Propose GA-based learning for software refactoring based on ANN	GA, ANN	—	Xerces-J, JFreeChart, GanttProject, AntApache, JHotDraw, and Rhino.	Wilcoxon test with a 99% confidence level ($\alpha = 0.01$)
[103]	2015	Regression	Removing defects with time series in a multi-objective approach	Multi-objective algorithm, based on NSGA-II, ARIMA		FindBugs, JFreeChart, Hibernate, Pixelitor, and JDI-Ford	Wilcoxon rank sum test with a 99% confidence level ($\alpha < 1\%$)
[104]	2016	Web mining/ clustering	Unsupervised learning approach to detect refactoring opportunities in service-oriented applications	PAM, K-means, COBWEB, X-Means	—	Two datasets of WSDL documents	COBWEB and K-means max. 83.33% and 0%, inter-cluster COBWEB and K-means min. 33.33% and 66.66% intra-cluster
[105]	2017	Clustering	A novel algorithm (HASP) for software refactoring at the package level	Hierarchical clustering algorithm	—	Three open-source case studies	Modularization Quality and Evaluation Metric Function
[99]	2017	Classification	A technique to predict refactoring at class level	PCA, SMOTE LS-SVM, RBF	—	From tera- PROMISE Repository seven open-source software systems	10-fold CV, AUC, and ROC curves RBF kernel outperforms linear and polynomial kernel The mean value of AUC for LS-SVM RBF kernel is 0.96

Ref.	Year	Task	Objective	Algorithms	EL	Dataset	Evaluation metrics and results
[106]	2017	Classification	Exploring the impact of clone refactoring (CR) on the test code size	LR, KNN, NB	RF	data collected from an open-source Java software system (ANT)	PR, recall, accuracy, F-score kNN and RF outperform NB ACC (fitting (98%), LOOCV (95%), and 10 FCV (95%))
[107]	2017	—	Finding refactoring opportunities in source code	J48, BayesNet, SVM, LR	RF	Ant, ArgoUML, jEdit, jFreeChart, Mylyn	10-fold CV, PR, recall 86–97% PR and 71–98% recall for proposed tech
[108]	2018	Classification	A learning-based approach (CREC) to extract refactored and non-refactored clone groups from repositories	C4.5, SMO, NB.	RF, Adaboost	Axis2, Eclipse.jdt.core, Elastic Search, JFreeChart, JRuby, and Lucene	PR, recall, F-score F-score = 83% in the within-project F-score = 76% in the cross-project
[109]	2018	Clustering	Combination of the use of multi-objective and unsupervised learning to decrease developer's effort	GMM, EM	—	ArgoUML, JHotDraw, GanttProject, UTest, Apache Ant, Azureus	One-way ANOVA with a 95% confidence level (α = 5%)

Table 5.
Data mining and machine learning studies on the subject "refactoring."

increases, the number of software defects decreases, and thus refactoring has a positive effect on software quality.

While automated refactoring does not always give the desired result, manual refactoring is time-consuming. Therefore, one study [109] proposed a clustering-based recommendation tool by combining multi-objective search and unsupervised learning algorithm to reduce the number of refactoring options. At the same time, the number of refactoring that should be selected is decreasing with the help of the developer's feedback.

4. Discussion

Since many SE studies that apply data mining approaches exist in the literature, this article presents only a few of them. However, **Figure 4** shows the current number of papers obtained from the Scopus search engine for each year from 2010 to 2019 by using queries in the title/abstract/keywords field. We extracted publications in 2020 since this year has not completed yet. Queries included ("data mining" OR "machine learning") with ("defect prediction" OR "defect detection" OR "bug prediction" OR "bug detection") for defect prediction, ("effort estimation" OR "effort prediction" OR "cost estimation") for effort estimation, ("vulnerab*" AND "software" OR "vulnerability analysis") for vulnerability analysis, and ("software" AND "refactoring") for refactoring. As seen in the figure, the number of studies using data mining in SE tasks, especially defect prediction and vulnerability analysis, has increased rapidly. The most stable area in the studies is design pattern mining.

Figure 5 shows the publications studied in classification, clustering, text mining, and association rule mining as a percentage of the total number of papers obtained by a Scopus query for each SE task. For example, in defect prediction, the number of studies is 339 in the field of classification, 64 in clustering, 8 in text mining, and 25 in the field of association rule mining. As can be seen from the pie charts, while clustering is a popular DM technique in refactoring, no study related to text mining is found in this field. In other SE tasks, the preferred technique is classification, and the second is clustering.

Defect prediction generally compares learning algorithms in terms of whether they find defects correctly using classification algorithms. Besides this approach, in some studies, clustering algorithms were used to select futures [110] or to compare supervised and unsupervised methods [27]. In the text mining area, to extract features from scripts, TF-IDF techniques were generally used [111, 112]. Although many different algorithms have been used in defect prediction, the most popular ones are NB, MLP, and RBF.

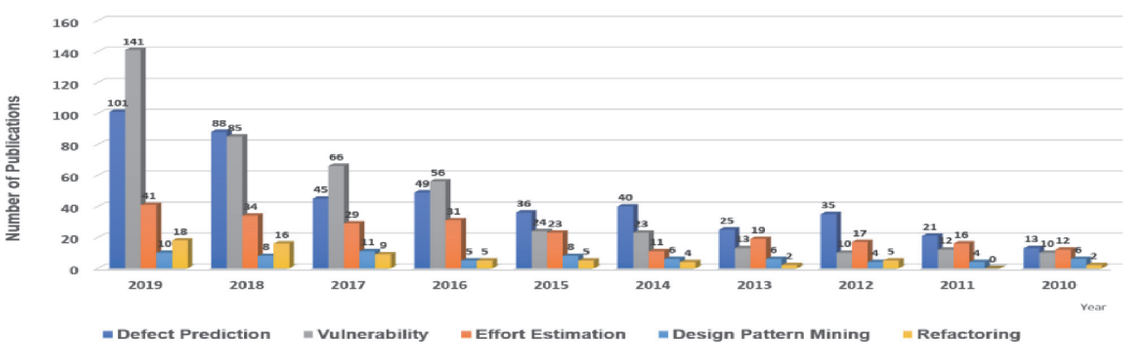

Figure 4.
Number of publications of data mining studies for SE tasks from Scopus search by their years.

Figure 6 shows the number of document types (conference paper, book chapter, article, book) published between the years of 2010 and 2019. It is clearly seen that conference papers and articles are the most preferred research study type. It is clearly seen that there is no review article about data mining studies in design pattern mining.

Table 6 shows popular repositories that contain various datasets and their descriptions, which tasks they are used for, and hyperlinks to download. For

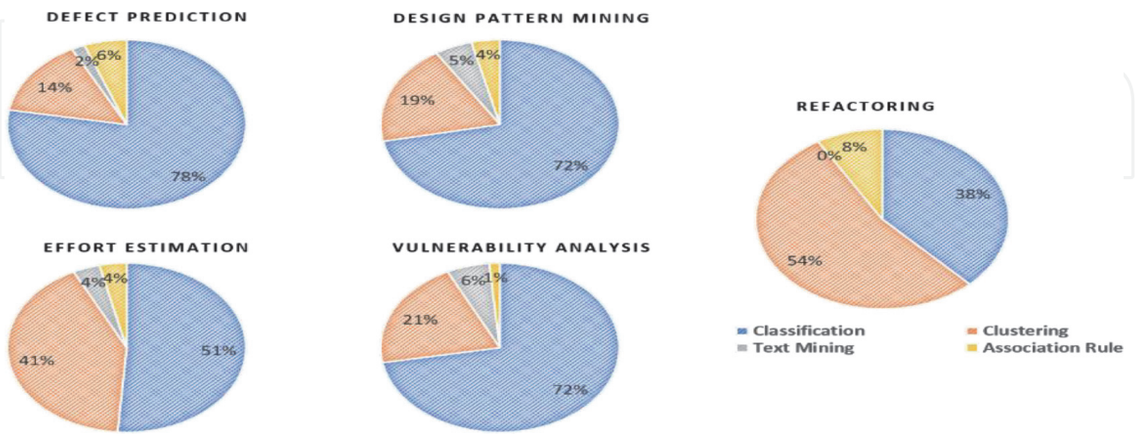

Figure 5.
Number of publications of data mining studies for SE tasks from Scopus search by their topics.

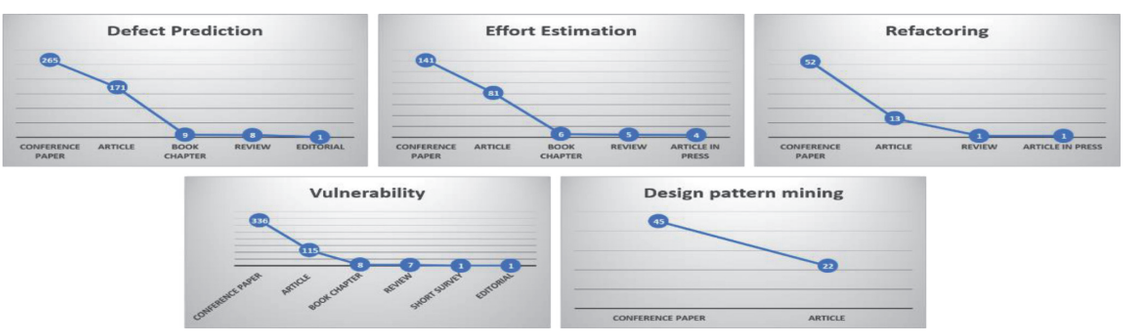

Figure 6.
The number of publications in terms of document type between 2010 and 2019.

Repository	Topic	Description	Web link
Nasa MDP	Defect Pred.	NASA's Metrics Data Program	https://github.com/opensciences/opensciences.github.io/tree/master/repo/defect/mccabehalsted/_posts
Android Git	Defect Pred.	Android version bug reports	https://android.googlesource.com/
PROMISE	Defect Pred. Effort Est.	It includes 20 datasets for defect prediction and cost estimation	http://promise.site.uottawa.ca/SERepository/datasets-page.html
Software Defect Pred. Data	Defect Pred.	It includes software metrics, # of defects, etc. Eclipse JDT: Eclipse PDE:	http://www.seiplab.riteh.uniri.hr/?page_id=834&lang=en
PMART	Design pattern mining	It has 22 patterns 9 Projects, 139 ins. Format: XML Manually detected and validated	http://www.ptidej.net/tools/designpatterns/

Table 6.
Description of popular repositories used in studies.

example, the PMART repository includes source files of java projects, and the PROMISE repository has different datasets with software metrics such as cyclomatic complexity, design complexity, and lines of code. Since these repositories contain many datasets, no detailed information about them has been provided in this article.

Refactoring can be applied at different levels; study [105] predicted refactoring at package level using hierarchical clustering, and another study [99] applied class-level refactoring using LS-SVM as learning algorithm, SMOTE for handling refactoring, and PCA for feature extraction.

5. Conclusion and future work

Data mining techniques have been applied successfully in many different domains. In software engineering, to improve the quality of a product, it is highly critical to find existing deficits such as bugs, defects, code smells, and vulnerabilities in the early phases of SDLC. Therefore, many data mining studies in the past decade have aimed to deal with such problems. The present paper aims to provide information about previous studies in the field of software engineering. This survey shows how classification, clustering, text mining, and association rule mining can be applied in five SE tasks: defect prediction, effort estimation, vulnerability analysis, design pattern mining, and refactoring. It clearly shows that classification is the most used DM technique. Therefore, new studies can focus on clustering on SE tasks.

Abbreviations

LMT	logistic model trees
Rip	repeated incremental pruning
NNge	nearest neighbor generalization
PCA	principal component analysis
PAM	partitioning around medoids
LS-SVM	least-squares support vector machines
MAE	mean absolute error
RBF	radial basis function
RUS	random undersampling
SMO	sequential minimal optimization
GMM	Gaussian mixture model
EM	expectation maximizaion
LR	logistic regression
SMB	SMOTEBoost
RUS-bal	balanced version of random undersampling
THM	threshold-moving
BNC	AdaBoost.NC
RF	random forest
RBF	radial basis function
CC	correlation coefficient
ROC	receiver operating characteristic
BayesNet	Bayesian network
SMOTE	synthetic minority over-sampling technique

Author details

Elife Ozturk Kiyak
Graduate School of Natural and Applied Sciences, Dokuz Eylul University, Turkey

*Address all correspondence to: elife.ozturk@ceng.deu.edu.tr

References

[1] Halkidi M, Spinellis D, Tsatsaronis G, Vazirgiannis M. Data mining in software engineering. Intelligent Data Analysis. 2011;**15**(3):413-441. DOI: 10.3233/IDA-2010-0475

[2] Dhamija A, Sikka S. A review paper on software engineering areas implementing data mining tools & techniques. International Journal of Computational Intelligence Research. 2017;**13**(4):559-574

[3] Minku LL, Mendes E, Turhan B. Data mining for software engineering and humans in the loop. Progress in Artificial Intelligence. 2016;**5**(4): 307-314

[4] Malhotra R. A systematic review of machine learning techniques for software fault prediction. Applied Soft Computing. 2015;**27**:504-518. DOI: 10.1016/j.asoc.2014.11.023

[5] Mayvan BB, Rasoolzadegan A, Ghavidel Yazdi Z. The state of the art on design patterns: A systematic mapping of the literature. Journal of Systems and Software. 2017;**125**:93-118. DOI: 10.1016/j.jss.2016.11.030

[6] Sehra SK, Brar YS, Kaur N, Sehra SS. Research patterns and trends in software effort estimation. Information and Software Technology. 2017;**91**:1-21. DOI: 10.1016/j.infsof.2017.06.002

[7] Taylor Q, Giraud-Carrier C, Knutson CD. Applications of data mining in software engineering. International Journal of Data Analysis Techniques and Strategies. 2010;**2**(3):243-257

[8] Coelho RA, Guimarães FRN, Esmin AA. Applying swarm ensemble clustering technique for fault prediction using software metrics. In: Machine Learning and Applications (ICMLA), 2014 13th International Conference on IEEE. 2014. pp. 356-361

[9] Prasad MC, Florence L, Arya A. A study on software metrics based software defect prediction using data mining and machine learning techniques. International Journal of Database Theory and Application. 2015;**8**(3):179-190. DOI: 10.14257/ijdta.2015.8.3.15

[10] Zhang Y, Lo D, Xia X, Sun J. Combined classifier for cross-project defect prediction: An extended empirical study. Frontiers of Computer Science. 2018;**12**(2):280-296. DOI: 10.1007/s11704-017-6015-y

[11] Yang X, Lo D, Xia X, Zhang Y, Sun J. Deep learning for just-in-time defect prediction. In: International Conference on Software Quality, Reliability and Security (QRS); 3–5 August 2015; Vancouver, Canada: IEEE; 2015. pp. 17-26

[12] Zhang F, Zheng Q, Zou Y, Hassan AE. Cross-project defect prediction using a connectivity-based unsupervised classifier. In: Proceedings of the 38th International Conference on Software Engineering ACM; 14–22 May 2016; Austin, TX, USA: IEEE; 2016. pp. 309-320

[13] Di Nucci D, Palomba F, Oliveto R, De Lucia A. Dynamic selection of classifiers in bug prediction: An adaptive method. IEEE Transactions on Emerging Topics in Computational Intelligence. 2017;**1**(3):202-212. DOI: 10.1109/TETCI.2017.2699224

[14] Zimmermann T, Nagappan N, Gall H, Giger E, Murphy B. Cross-project defect prediction: A large scale experiment on data vs. domain vs. process. In: Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering (ESEC/FSE '09); August 2009; Amsterdam, Netherlands: ACM; 2009. pp. 91-100

[15] Turhan B, Menzies T, Bener AB, Di Stefano J. On the relative value of cross-company and within-company data for defect prediction. Empirical Software Engineering. 2009;**14**(5):540-578. DOI: 10.1007/s10664-008-9103-7

[16] Herbold S, Trautsch A, Grabowski J. A comparative study to benchmark cross-project defect prediction approaches. IEEE Transactions on Software Engineering. 2017;**44**(9): 811-833. DOI: 10.1109/TSE.2017.2724538

[17] Ghotra B, McIntosh S, Hassan AE. Revisiting the impact of classification techniques on the performance of defect prediction models. In: IEEE/ACM 37th IEEE International Conference on Software Engineering; 16–24 May 2015; Florence, Italy: IEEE; 2015. pp. 789-800

[18] Wang T, Li W, Shi H, Liu Z. Software defect prediction based on classifiers ensemble. Journal of Information & Computational Science. 2011;**8**:4241-4254

[19] Wang S, Yao X. Using class imbalance learning for software defect prediction. IEEE Transactions on Reliability. 2013;**62**:434-443. DOI: 10.1109/TR.2013.2259203

[20] Rodriguez D, Herraiz I, Harrison R, Dolado J, Riquelme JC. Preliminary comparison of techniques for dealing with imbalance in software defect prediction. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering; May 2014; London, United Kingdom: ACM; 2014. p. 43

[21] Laradji IH, Alshayeb M, Ghouti L. Software defect prediction using ensemble learning on selected features. Information and Software Technology. 2015;**58**:388-402. DOI: 10.1016/j.infsof.2014.07.005

[22] Malhotra R, Raje R. An empirical comparison of machine learning techniques for software defect prediction. In: Proceedings of the 8th International Conference on Bioinspired Information and Communications Technologies. Boston, Massachusetts; December 2014. pp. 320-327

[23] Malhotra R. An empirical framework for defect prediction using machine learning techniques with Android software. Applied Soft Computing. 2016;**49**:1034-1050. DOI: 10.1016/j.asoc.2016.04.032

[24] Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K. Automated parameter optimization of classification techniques for defect prediction models. In: Proceedings of the 38th International Conference on Software Engineering (ICSE '16). Austin, Texas; May 2016. pp. 321-332

[25] Kumar L, Misra S, Rath SK. An empirical analysis of the effectiveness of software metrics and fault prediction model for identifying faulty classes. Computer Standards & Interfaces. 2017; **53**:1-32. DOI: 10.1016/j.csi.2017.02.003

[26] Yang X, Lo D, Xia X, Sun J. TLEL: A two-layer ensemble learning approach for just-in-time defect prediction. Information and Software Technology. 2017;**87**:206-220. DOI: 10.1016/j.infsof.2017.03.007

[27] Chen X, Zhao Y, Wang Q, Yuan Z. MULTI: Multi-objective effort-aware just-in-time software defect prediction. Information and Software Technology. 2018;**93**:1-13. DOI: 10.1016/j.infsof.2017.08.004

[28] Zimmermann T, Premraj R, Zeller A. Predicting defects for eclipse. In: Third International Workshop on Predictor Models in Software Engineering (PROMISE'07); 20-26 May 2007; Minneapolis, USA: IEEE; 2007. p. 9

[29] Prakash VA, Ashoka DV, Aradya VM. Application of data mining

techniques for defect detection and classification. In: Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA); 14–15 November 2014; Odisha, India; 2014. pp. 387-395

[30] Yousef AH. Extracting software static defect models using data mining. Ain Shams Engineering Journal. 2015;**6**: 133-144. DOI: 10.1016/j.asej.2014.09.007

[31] Gupta DL, Saxena K. AUC based software defect prediction for object-oriented systems. International Journal of Current Engineering and Technology. 2016;**6**:1728-1733

[32] Kumar L, Rath SK. Application of genetic algorithm as feature selection technique in development of effective fault prediction model. In: IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics Engineering (UPCON); 9-11 December 2016; Varanasi, India: IEEE; 2016. pp. 432-437

[33] Tomar D, Agarwal S. Prediction of defective software modules using class imbalance learning. Applied Computational Intelligence and Soft Computing. 2016;**2016**:1-12. DOI: 10.1155/2016/7658207

[34] Ryu D, Baik J. Effective multi-objective naïve Bayes learning for cross-project defect prediction. Applied Soft Computing. 2016;**49**:1062-1077. DOI: 10.1016/j.asoc.2016.04.009

[35] Ali MM, Huda S, Abawajy J, Alyahya S, Al-Dossari H, Yearwood J. A parallel framework for Software Defect detection and metric selection on cloud computing. Cluster Computing. 2017; **20**:2267-2281. DOI: 10.1007/s10586-017-0892-6

[36] Wijaya A, Wahono RS. Tackling imbalanced class in software defect prediction using two-step cluster based random undersampling and stacking technique. Jurnal Teknologi. 2017;**79**: 45-50

[37] Singh PD, Chug A. Software defect prediction analysis using machine learning algorithms. In: 7th International Conference on Cloud Computing, Data Science & Engineering-Confluence; 2–13 January 2017; Noida, India: IEEE; 2017. pp. 775-781

[38] Hammouri A, Hammad M, Alnabhan M, Alsarayrah F. Software bug prediction on using machine learning approach. International Journal of Advanced Computer Science and Applications. 2018;**9**:78-83

[39] Akour M, Alsmadi I, Alazzam I. Software fault proneness prediction: A comparative study between bagging, boosting, and stacking ensemble and base learner methods. International Journal of Data Analysis Techniques and Strategies. 2017;**9**:1-16

[40] Bowes D, Hall T, Petric J. Software defect prediction: Do different classifiers find the same defects? Software Quality Journal. 2018;**26**: 525-552. DOI: 10.1007/s11219-016-9353-3

[41] Watanabe T, Monden A, Kamei Y, Morisaki S. Identifying recurring association rules in software defect prediction. In: IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS); 26–29 June 2016; Okayama, Japan: IEEE; 2016. pp. 1-6

[42] Zhang S, Caragea D, Ou X. An empirical study on using the national vulnerability database to predict software vulnerabilities. In: International Conference on Database and Expert Systems Applications. Berlin, Heidelberg: Springer; 2011. pp. 217-223

[43] Wen J, Li S, Lin Z, Hu Y, Huang C. Systematic literature review of machine

learning based software development effort estimation models. Information and Software Technology. 2012;**54**: 41-59. DOI: 10.1016/j.infsof.2011.09.002

[44] Dave VS, Dutta K. Neural network based models for software effort estimation: A review. Artificial Intelligence Review. 2014;**42**:295-307. DOI: 10.1007/s10462-012-9339-x

[45] Kultur Y, Turhan B, Bener AB. ENNA: Software effort estimation using ensemble of neural networks with associative memory. In: Proceedings of the 16th ACM SIGSOFT; November 2008; Atlanta, Georgia: ACM; 2008. pp. 330-338

[46] Kultur Y, Turhan B, Bener A. Ensemble of neural networks with associative memory (ENNA) for estimating software development costs. Knowledge-Based Systems. 2009;**22**: 395-402. DOI: 10.1016/j.knosys.2009.05.001

[47] Corazza A, Di Martino S, Ferrucci F, Gravino C, Mendes E. Investigating the use of support vector regression for web effort estimation. Empirical Software Engineering. 2011;**16**:211-243. DOI: 10.1007/s10664-010-9138-4

[48] Minku LL, Yao X. A principled evaluation of ensembles of learning machines for software effort estimation. In: Proceedings of the 7th International Conference on Predictive Models in Software Engineering; September 2011; Banff, Alberta, Canada: ACM; 2011. pp. 1-10

[49] Minku LL, Yao X. Software effort estimation as a multiobjective learning problem. ACM Transactions on Software Engineering and Methodology (TOSEM). 2013;**22**:35. DOI: 10.1145/2522920.2522928

[50] Minku LL, Yao X. Can cross-company data improve performance in software effort estimation? In: Proceedings of the 8th International Conference on Predictive Models in Software Engineering (PROMISE '12); September 2012; New York, United States: ACM; 2012. pp. 69-78

[51] Kocaguneli E, Menzies T, Keung JW. On the value of ensemble effort estimation. IEEE Transactions on Software Engineering. 2012;**38**: 1403-1416. DOI: 10.1109/TSE.2011.111

[52] Dejaeger K, Verbeke W, Martens D, Baesens B. Data mining techniques for software effort estimation. IEEE Transactions on Software Engineering. 2011;**38**:375-397. DOI: 10.1109/TSE.2011.55

[53] Khatibi V, Jawawi DN, Khatibi E. Increasing the accuracy of analogy based software development effort estimation using neural networks. International Journal of Computer and Communication Engineering. 2013;**2**:78

[54] Subitsha P, Rajan JK. Artificial neural network models for software effort estimation. International Journal of Technology Enhancements and Emerging Engineering Research. 2014;**2**:76-80

[55] Maleki I, Ghaffari A, Masdari M. A new approach for software cost estimation with hybrid genetic algorithm and ant colony optimization. International Journal of Innovation and Applied Studies. 2014;**5**:72

[56] Huang J, Li YF, Xie M. An empirical analysis of data preprocessing for machine learning-based software cost estimation. Information and Software Technology. 2015;**67**:108-127. DOI: 10.1016/j.infsof.2015.07.004

[57] Nassif AB, Azzeh M, Capretz LF, Ho D. Neural network models for software development effort estimation. Neural Computing and Applications. 2016;**27**:2369-2381. DOI: 10.1007/s00521-015-2127-1

[58] Zare F, Zare HK, Fallahnezhad MS. Software effort estimation based on the optimal Bayesian belief network. Applied Soft Computing. 2016;**49**:968-980. DOI: 10.1016/j.asoc.2016.08.004

[59] Azzeh M, Nassif AB. A hybrid model for estimating software project effort from use case points. Applied Soft Computing. 2016;**49**:981-989. DOI: 10.1016/j.asoc.2016.05.008

[60] Hidmi O, Sakar BE. Software development effort estimation using ensemble machine learning. International Journal of Computing, Communication and Instrumentation Engineering. 2017;**4**:143-147

[61] Ghaffarian SM, Shahriari HR. Software vulnerability analysis and discovery using machine-learning and data-mining techniques. ACM Computing Surveys (CSUR). 2017;**50**: 1-36. DOI: 10.1145/3092566

[62] Jimenez M, Papadakis M, Le Traon Y. Vulnerability prediction models: A case study on the linux kernel. In: IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM); 2–3 October 2016; Raleigh, NC, USA: IEEE; 2016. pp. 1-10

[63] Walden J, Stuckman J, Scandariato R. Predicting vulnerable components: Software metrics vs text mining. In: IEEE 25th International Symposium on Software Reliability Engineering; 3–6 November 2014; Naples, Italy: IEEE; 2014. pp. 23-33

[64] Wijayasekara D, Manic M, Wright JL, McQueen M. Mining bug databases for unidentified software vulnerabilities. In: 5th International Conference on Human System Interactions; 6–8 June 2012; Perth, WA, Australia: IEEE; 2013. pp. 89-96

[65] Hovsepyan A, Scandariato R, Joosen W, Walden J. Software vulnerability prediction using text analysis techniques. In: Proceedings of the 4th International Workshop on Security Measurements and Metrics (ESEM '12); September 2012; Lund Sweden: IEEE; 2012. pp. 7-10

[66] Chernis B, Verma R. Machine learning methods for software vulnerability detection. In: Proceedings of the Fourth ACM International Workshop on Security and Privacy Analytics (CODASPY '18); March 2018; Tempe, AZ, USA: 2018. pp. 31-39

[67] Li X, Chen J, Lin Z, Zhang L, Wang Z, Zhou M, et al. Mining approach to obtain the software vulnerability characteristics. In: 2017 Fifth International Conference on Advanced Cloud and Big Data (CBD); 13–16 August 2017; Shanghai, China: IEEE; 2017. pp. 296-301

[68] Dam HK, Tran T, Pham T, Ng SW, Grundy J, Ghose A. Automatic feature learning for vulnerability prediction. arXiv preprint arXiv:170802368 2017

[69] Scandariato R, Walden J, Hovsepyan A, Joosen W. Predicting vulnerable software components via text mining. IEEE Transactions on Software Engineering. 2014;**40**: 993-1006

[70] Tang Y, Zhao F, Yang Y, Lu H, Zhou Y, Xu B. Predicting vulnerable components via text mining or software metrics? An effort-aware perspective. In: IEEE International Conference on Software Quality, Reliability and Security; 3–5 August 2015; Vancouver, BC, Canada: IEEE; 2015. p. 27–36

[71] Wang Y, Wang Y, Ren J. Software vulnerabilities detection using rapid density-based clustering. Journal of Information and Computing Science. 2011;**8**:3295-3302

[72] Medeiros I, Neves NF, Correia M. Automatic detection and correction of

web application vulnerabilities using data mining to predict false positives. In: Proceedings of the 23rd International Conference on World Wide Web (WWW '14); April 2014; Seoul, Korea; 2014. pp. 63-74

[73] Yamaguchi F, Golde N, Arp D, Rieck K. Modeling and discovering vulnerabilities with code property graphs. In: 2014 IEEE Symposium on Security and Privacy; 18-21 May 2014; San Jose, CA, USA: IEEE; 2014. pp. 590-604

[74] Perl H, Dechand S, Smith M, Arp D, Yamaguchi F, Rieck K, et al. Vccfinder: Finding Potential Vulnerabilities in Open-source Projects to Assist Code Audits. In: 22nd ACM Conference on Computer and Communications Security (CCS'15). Denver, Colorado, USA; 2015. pp. 426-437

[75] Yamaguchi F, Maier A, Gascon H, Rieck K. Automatic inference of search patterns for taint-style vulnerabilities. In: 2015 IEEE Symposium on Security and Privacy; San Jose, CA, USA: IEEE; 2015. pp. 797-812

[76] Grieco G, Grinblat GL, Uzal L, Rawat S, Feist J, Mounier L. Toward large-scale vulnerability discovery using machine learning. In: Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy; March 2016; New Orleans, Louisiana, USA; 2016. pp. 85-96

[77] Pang Y, Xue X, Wang H. Predicting vulnerable software components through deep neural network. In: Proceedings of the 2017 International Conference on Deep Learning Technologies; June 2017; Chengdu, China; 2017. pp. 6-10

[78] Li Z, Zou D, Xu S, Ou X, Jin H, Wang S, et al. VulDeePecker: A Deep Learning-Based System for Vulnerability Detection. arXiv preprint arXiv:180101681. 2018

[79] Imtiaz SM, Bhowmik T. Towards data-driven vulnerability prediction for requirements. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering; November, 2018; Lake Buena Vista, FL, USA. 2018. pp. 744-748

[80] Jie G, Xiao-Hui K, Qiang L. Survey on software vulnerability analysis method based on machine learning. In: IEEE First International Conference on Data Science in Cyberspace (DSC); 13–16 June 2016; Changsha, China: IEEE; 2017. pp. 642-647

[81] Russell R, Kim L, Hamilton L, Lazovich T, Harer J, Ozdemir O, et al. Automated vulnerability detection in source code using deep representation learning. In: 17th IEEE International Conference on Machine Learning and Applications (ICMLA). Orlando, FL, USA: IEEE; 2018, 2019. pp. 757-762

[82] Mayvan BB, Rasoolzadegan A, Yazdi ZG. The state of the art on design patterns: A systematic mapping of the literature. Journal of Systems and Software. 2017;**125**:93-118. DOI: 10.1016/j.jss.2016.11.030

[83] Dong J, Zhao Y, Peng T. A review of design pattern mining techniques. International Journal of Software Engineering and Knowledge Engineering. 2009;**19**:823-855. DOI: 10.1142/S021819400900443X

[84] Fowler M. Analysis Patterns: Reusable Object Models. Boston: Addison-Wesley Professional; 1997

[85] Vlissides J, Johnson R, Gamma E, Helm R. Design Patterns-Elements of Reusable Object-Oriented Software. 1st ed. Addison-Wesley Professional; 1994

[86] Hasheminejad SMH, Jalili S. Design patterns selection: An automatic two-phase method. Journal of Systems and

Software. 2012;**85**:408-424. DOI: 10.1016/j.jss.2011.08.031

[87] Alhusain S, Coupland S, John R, Kavanagh M. Towards machine learning based design pattern recognition. In: 2013 13th UK Workshop on Computational Intelligence (UKCI); 9–11 September 2013; Guildford, UK: IEEE; 2013. pp. 244-251

[88] Tekin U. Buzluca F, A graph mining approach for detecting identical design structures in object-oriented design models. Science of Computer Programming. 2014;**95**:406-425. DOI: 10.1016/j.scico.2013.09.015

[89] Zanoni M, Fontana FA, Stella F. On applying machine learning techniques for design pattern detection. Journal of Systems and Software. 2015;**103**: 102-117. DOI: 10.1016/j.jss.2015.01.037

[90] Chihada A, Jalili S, Hasheminejad SMH, Zangooei MH. Source code and design conformance, design pattern detection from source code by classification approach. Applied Soft Computing. 2015;**26**:357-367. DOI: 10.1016/j.asoc.2014.10.027

[91] Dwivedi AK, Tirkey A, Ray RB, Rath SK. Software design pattern recognition using machine learning techniques. In: 2016 IEEE Region 10 Conference (TENCON); 22–25 November 2016; Singapore, Singapore: IEEE; 2017. pp. 222-227

[92] Dwivedi AK, Tirkey A, Rath SK. Applying software metrics for the mining of design pattern. In: IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics Engineering (UPCON); 9–11 December 2016; Varanasi, India: IEEE; 2017. pp. 426-431

[93] Dwivedi AK, Tirkey A, Rath SK. Software design pattern mining using classification-based techniques. Frontiers of Computer Science. 2018;**12**:908-922. DOI: 10.1007/s11704-017-6424-y

[94] Mayvan BB, Rasoolzadegan A. Design pattern detection based on the graph theory. Knowledge-Based Systems. 2017;**120**:211-225. DOI: 10.1016/j.knosys.2017.01.007

[95] Hussain S, Keung J, Khan AA. Software design patterns classification and selection using text categorization approach. Applied Soft Computing. 2017;**58**:225-244. DOI: 10.1016/j.asoc.2017.04.043

[96] Kaur A, Singh S. Detecting software bad smells from software design patterns using machine learning algorithms. International Journal of Applied Engineering Research. 2018;**13**: 10005-10010

[97] Hussain S, Keung J, Khan AA, Ahmad A, Cuomo S, Piccialli F. Implications of deep learning for the automation of design patterns organization. Journal of Parallel and Distributed Computing. 2018;**117**: 256-266. DOI: 10.1016/j.jpdc.2017.06.022

[98] Fowler M. Refactoring: Improving the Design of Existing Code. 2nd ed. Boston: Addison-Wesley Professional; 2018

[99] Kumar L, Sureka A. Application of LSSVM and SMOTE on seven open source projects for predicting refactoring at class level. In: 24th Asia-Pacific Software Engineering Conference (APSEC); 4–8 December 2017; Nanjing, China: IEEE; 2018. pp. 90-99

[100] Ratzinger J, Sigmund T, Vorburger P, Gall H. Mining software evolution to predict refactoring. In: First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007); 20–21 September 2007; Madrid, Spain: IEEE; 2007. pp. 354-363

[101] Ratzinger J, Sigmund T, Gall HC. On the relation of refactoring and

software defects. In: Proceedings of the 2008 International Working Conference on Mining Software Repositories; May 2008; Leipzig, Germany: ACM; 2008. pp. 35-38

[102] Amal B, Kessentini M, Bechikh S, Dea J, Said LB. On the Use of Machine Learning and Search-Based software engineering for ill-defined fitness function: A case study on software refactoring. In: International Symposium on Search Based Software Engineering; 26-29 August 2014; Fortaleza, Brazil; 2014. pp. 31-45

[103] Wang H, Kessentini M, Grosky W, Meddeb H. On the use of time series and search based software engineering for refactoring recommendation. In: Proceedings of the 7th International Conference on Management of Computational and Collective intElligence in Digital EcoSystems. Caraguatatuba, Brazil; October 2015. pp. 35-42

[104] Rodríguez G, Soria Á, Teyseyre A, Berdun L, Campo M. Unsupervised learning for detecting refactoring opportunities in service-oriented applications. In: International Conference on Database and Expert Systems Applications; 5–8 September; Porto, Portugal: Springer; 2016. pp. 335-342

[105] Marian Z, Czibula IG, Czibula G. A hierarchical clustering-based approach for software restructuring at the package level. In: 9th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC); 21–24 September 2017; Timisoara, Romania: IEEE; 2018. pp. 239-246

[106] Mourad B, Badri L, Hachemane O, Ouellet A. Exploring the impact of clone refactoring on test code size in object-oriented software. In: 16th IEEE International Conference on Machine Learning and Applications (ICMLA); 18-21 December 2017; Cancun, Mexico. 2018. pp. 586-592

[107] Imazato A, Higo Y, Hotta K, Kusumoto S. Finding extract method refactoring opportunities by analyzing development history. In: IEEE 41st Annual Computer Software and Applications Conference (COMPSAC); 4–8 July 2017; Turin, Italy: IEEE; 2018. pp. 190-195

[108] Yue R, Gao Z, Meng N, Xiong Y, Wang X. Automatic clone recommendation for refactoring based on the present and the past. In: IEEE International Conference on Software Maintenance and Evolution (ICSME); 23–29 September 2018; Madrid, Spain: IEEE; 2018. pp. 115-126

[109] Alizadeh V, Kessentini M. Reducing interactive refactoring effort via clustering-based multi-objective search. In: 33rd ACM/IEEE International Conference on Automated Software Engineering; September 2018; Montpellier, France: ACM/IEEE; 2018. pp. 464-474

[110] Ni C, Liu WS, Chen X, Gu Q, Chen DX, Huang QG. A cluster based feature selection method for cross-project software defect prediction. Journal of Computer Science and Technology. 2017;**32**:1090-1107. DOI: 10.1007/s11390-017-1785-0

[111] Rahman A, Williams L. Characterizing defective configuration scripts used for continuous deployment. In: 11th International Conference on Software Testing, Verification and Validation (ICST); 9–13 April 2018; Vasteras, Sweden: IEEE; 2018. pp. 34-45

[112] Kukkar A, Mohana R. A supervised bug report classification with incorporate and textual field knowledge. Procedia Computer Science. 2018;**132**: 352-361. DOI: 10.1016/j.procs.2018.05.194