RES MILITARIS
REVUE EUROPEENNE D ETUDES EUROPEAN JOURNAL OF MILITARY STUDIES
Social Science Journal

Software Defect Prediction using Dimensionality Reduction Technique

Bhanu Pratap Rai, C. S. Raghuvanshi, Hari Om Sharan

Faculty of Engineering & Technology, Rama University, Mandhana, Kanpur, 209217, U.P., India.

*Corresponding author(s). E-mail(s): bhanurai0@gmail.com;

Abstract

The software system has a huge amount of programming codes, several procedures, and modules. Due to this, it is too complex to understand. For the better experience of the user, software must be running efficiently. If any module gets any type of defect, it may harm the field of health care, defense, education, and so on. It can also be expensive in terms of money, effort, and reputation for a company. It is very tedious work to identify the defects in the module. To allot resources efficiently, reduce costs, and enhance the performance of software, the prediction of defective software is very necessary and it may help developers to save time and reduce the development cost. In the testing phase of software, we get most of the defects which reduces the quality of the software. According to the report (Herb Krasner), the testing phase takes more time as compared to other phases and takes more than 50 percent cost of the software where finding and fixing the defect takes place. This paper uses five datasets "MC1, JM1, KC1, CM1, and PC1" of the NASA repository for analysis. This repository contains 13 datasets with different instances from range 127 to 17001. Firstly, we use a Support Vector Machine (SVM), Random Forest (RF), and two Naïve Bays (NB) algorithms namely Gaussian Naive Bayes (GNB) and Bernoulli Naive Bayes (BNB) to calculate the results for each dataset. Secondly, we use PCA for dimensionality reduction and calculate the results before and after applying PCA to all aforementioned algorithms. Finally, we observe that after using PCA, the results of all the algorithms have been improved.

Keywords: Machine Learning, Software Defect Prediction, Dimensionality Reduction, Principle Component Analysis

1 Introduction

It is a challenging process to develop quality software that can fulfill users' requirements. Removing errors or flaws in software is the primary factor that determines its dependability and quality[11][27]. Software defect prediction(SDP) is a technique used in software engineering to anticipate or predict the presence of defects, bugs, or issues in a software system before it is released. SDP aims to oblige software professionals to allocate test resources more efficiently[28]. Using SDP approaches, errors are found early in the "software development life cycle" (SDLC)[4][23]. It involves the use of historical data, metrics, and various algorithms to create models that can identify areas of code or modules that are more likely to contain defects. This process requires a combined effort of all software development teams for analyzing, planning, testing, and execution. In the

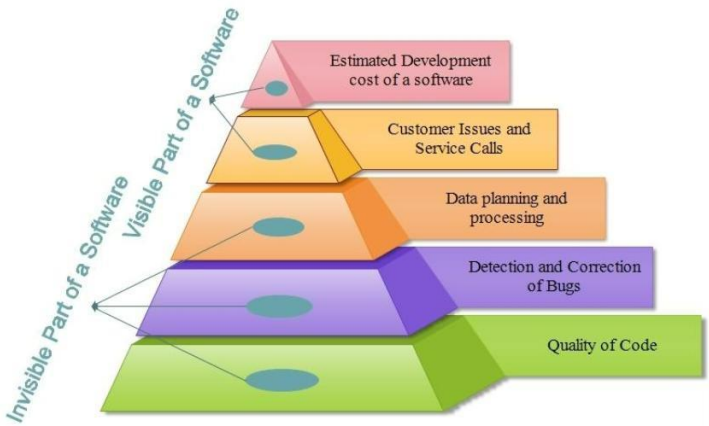

Fig. 1 Iceberg Model for SDP

testing phase, we get most of the defects which reduce the quality of the software. According to [16][26], the testing phase takes more time and more than 50 percent cost of the software, where finding and fixing the defect takes place. However, there is a correlation between the Iceberg model [16] and the overall cost of software delivery, which is not always visible. Another paper of Anon 2018a (Software fails watch) which was published in Tricentis, examined 606 famous software failures. This paper reported that 314 companies were involved in this analysis, assets worth $1.7 trillion, and 3.8 billion people were affected.

Several old and new methods have been used by researchers in the last few years for SDP like manual testing, automation, static analysis, and AI approaches. Some of the techniques take lots of time, but some of them are very fast, some of them do not require historical datasets but in some approaches, we need historical datasets. We can see the differences among all of them in Figure 5. In this field of SDP, researchers have found several issues and attempted to provide solutions. However, no solution by the researchers has been accepted universally. Several problems are still unanswered [2]. The problems of the SDP are as follows:

- Absence of a general framework
- There are no standard performance measures
- The connection between fault and attributes
- Problems in predicting cross-project
- The Economics of Predicting Software Defects

Several experiences have shown that the use of defective software hurts the business. Suppose we take an example of a defective life-critical system software of any hospital. It is very dangerous to use. This could even kill someone. Critical business applications need reliable software, and it is the main challenge for the development team of the software industry. It can lead to various issues and challenges throughout the software development lifecycle. Here are some common problems associated with defects in software:

Functional Issues: Defects can cause the software to behave unexpectedly or fail to perform specific functions as intended. This can result in incorrect calculations, data processing errors, or other functional issues.

User Experience Problems: Defects can impact the user interface, making it difficult for users to interact with the software. This may lead to frustration, confusion, and a poor overall user experience.

Security Vulnerabilities: Some defects can create security vulnerabilities, exposing the software to potential cyber threats. This may include

issues like input validation errors, buffer overflows, or other vulnerabilities that could be exploited by malicious actors.

Data Loss or Corruption: Defects in data processing or storage components can lead to data loss or corruption. This is particularly problematic when dealing with sensitive or critical information.

System Crashes: Severe defects can cause the entire software system to crash. This disrupts the normal operation of the software and can lead to data loss or downtime.

Delayed Timelines: Identifying and fixing defects can consume a significant amount of time and resources. This may lead to project delays and impact the overall timeline for software development.

Increased Costs: The process of identifying, fixing, and retesting defects can contribute to increased development costs. Additionally, defects discovered later in the development cycle or after release can be more expensive to address.

Reputation Damage: Defects that affect the user experience or result in critical issues can harm the reputation of the software and the development team. Users may lose trust in the product, and negative reviews can impact future adoption.

Difficulty in Reproduction: Some defects may be challenging to reproduce consistently, making it difficult for developers to identify the root cause and implement a fix.

Communication Challenges: Defects can lead to misunderstandings and communication challenges between development teams, quality assurance (QA) teams, and stakeholders. Clear communication is crucial for effective defect resolution.

To mitigate these problems, it's essential to implement robust testing processes, conduct thorough code reviews, and prioritize continuous integration and continuous testing practices throughout the software development lifecycle. Additionally, fostering a culture of quality and collaboration within the development team can help address and prevent defects more effectively.

1.1 Section Introduction

we divided all our work into six sections in this paper which are as follows:

- The first section is the introduction section. In this section we have discussed about importance of software, what is defects, the reason behind writing this paper (problems with defects, benefits of defect prediction), and the background of both the forecast & the finding of defects. The problems with SDPs are also discussed in this section.
- The second section is the background section for the dataset and used algorithms. In this section, we have discussed all the datasets and algorithms (SVM, RB, NB, and PCA) we use for SDP analysis. The List of all available algorithms of ML and the general flow of SDP is also given in this section.
- The third section is for SLR. Based on this SLR which process we have used for SDP is also discussed in this third section. Based on SLR we saw the difference between conventional and predictive approaches of SDP. These differences are also discussed in this section.
- In the Fourth section, we discussed why we use SDP, the flow of SDP, the process, and the used steps for the SDP. We can see the pictorial representation of the used process of SDP in Figure 6 in this section.
- The Fifth section is for results and discussion. The result set of the algorithms on the used datasets is also mentioned in this section which we can see in "Table-2, Table-3, Table-4, Table-5, Table-6, and Table-7" with bar charts of all the tables. In this section, we discussed result sets and compared all the datasets and saw that after using PCA results of all the algorithms have been increased.
- The sixth section is for the conclusion.

2 Background

In modern times, the software industry is growing and becoming more advanced. As we know we are highly dependent on the software. This software must be defect-free, effective, highly secure, and reliable. It should have the capability of maintenance according to the user's requirement and should be small in size. According to the definition of ANSI, "The possibility that a piece of software will operate without errors for a predetermined amount of time in a predetermined environment is known as software reliability.

A bibliometric review [20] indicates that researchers worldwide are very interested in the topic of SDP. We realize that machine learning

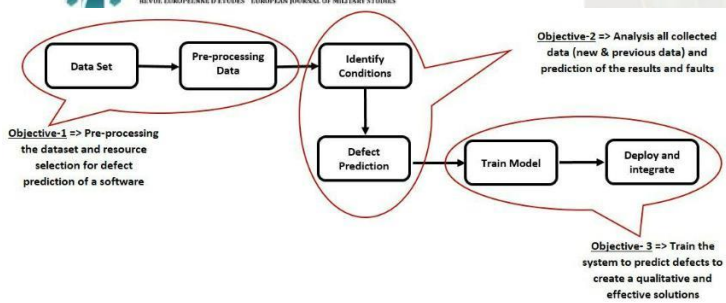

Fig. 2 Overview of SDP Process

(ML) techniques are winning nowadays. ML is a subset of AI where we provide a large dataset to the machine and train it with this dataset to predict the output according to the given input. As we know SDP is essential. Figure 2 shows an overview of the SDP process. To predict the defects preprocessing of the dataset is very important to analyze defects and to identify defects. We can apply different ML algorithms to train and test the model to predict defects.

2.1 Used Datasets

As of the last update in January 2022, there isn't a specific NASA dataset solely dedicated to software defect prediction publicly available and officially released by NASA. However, NASA has been involved in various research projects related to software engineering and defect prediction, and some of their datasets might be available through collaborations, publications, or research repositories. The NASA repository is used in this research paper which is available for all researchers of SDP for analysis. This repository contains 13 datasets with different instances from range 127 to 17001. All the datasets contain several attributes from the range 20 to 40. We are using five datasets "MC1, JM1, KC1, CM1, and PC1" of NASA repository. The details of the dataset are given in Table 1:

Dataset	Number of Attributes	Number of Records
CM1	39	327
JM1	23	7782
KC1	23	1183
MC1	40	1988
PC1	39	705

Table 1 Number of attributes and records of the datasets

2.2 Used Algorithms

ML is very famous among all the researchers[6],[7][22]. Large amounts of data are automatically analyzed and examined using machine learning. Without human involvement, it automates the data analysis process and instantly produces forecasts. ML comes in four varieties "Supervised, Semi-supervised, Unsupervised, and Reinforcement ML", and many algorithms are used in these types to process the data which we can see in Figure 3. Among all the algorithms we are using SVM, RF, NB, and PCA algorithms for analysis of the NASA dataset.

In this paper, after preprocessing the dataset we divided the data into training and testing and used SVM, RF, and NB (GNB, CNB) algorithms to calculate the results. After calculating the results we applied PCA to decrease the dataset's dimensionality and selected 20 principal components of each dataset for analysis. Based on principal components, we created new datasets with 20 principal components and again applied SVM, RF, and NB (GNB, CNB) algorithms to calculate the results. The results of all the algorithms increased the maximum datasets in the NASA repository. Tables 2,3,4,5,6, and 7 show the results of all the algorithms on every dataset before using PCA and after using PCA. Figure 6 shows the process of SDP used in this paper. SVM is a very famous supervised ML algorithm used for both "classification and regression" types of problems[1] and PCA is a type of unsupervised ML technique to reduce the features/dimensionality of the dataset.

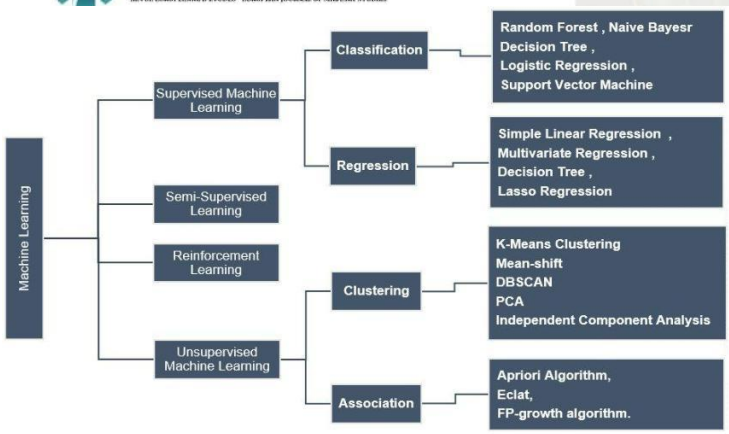

Fig. 3 Types of ML Algorithms used for SDP

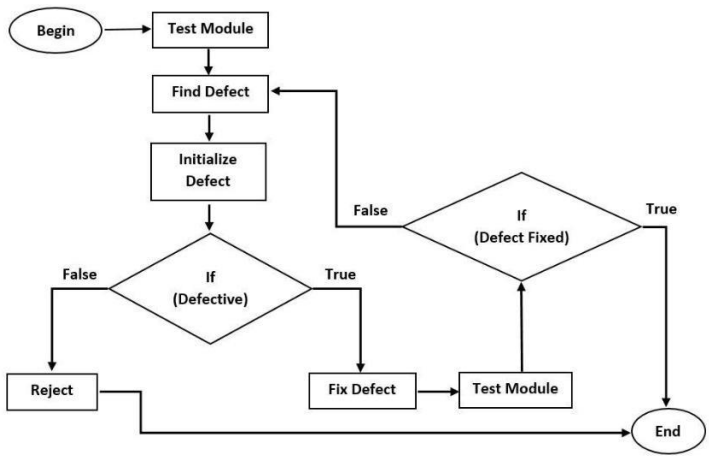

Fig. 4 Flow Chart of SDP

PCA facilitates the identification of patterns in data, based on the correlation between dimensions or features.

We used SVM with PCA to check the different scores of the dataset. We calculated the result of SVM without using PCA and after using PCA and saw that the score of SVM increased after using PCA. We can see all the scores of SVM with the datasets in Table 2.

RF is also a type of supervised ML technique that can be used for both "classification" as well as "regression"[1]. The RF algorithm is applied to the dataset to calculate the score, after this calculation, we selected twenty principal components/ features of each dataset using PCA, and again applied the RF algorithm to the selected components/ features and calculated the result. After applying PCA the results were increased which is

mentioned in the table. Table 3 shows the scores of the RF algorithm on the dataset before and after using PCA. Among all algorithms, NB is also a type of supervised ML technique which can be used for both "classification and regression". Its foundation is the "Bayes Theorem. The most efficient and straightforward classification algorithms are those that aid in the rapid construction of machine-learning models with rapid prediction capabilities. There are three types of NB model:

- Gaussian Naive Bayes – GNB: It is used when normal distribution is followed by features.
- Multinomial Naïve Bayes – MNB: When the data is distributed multinomial, It is used.
- Bernoulli Naive Bayes – BNB: Same as MNB but the independent Boolean variables are the predictor variables.

In this paper, we are using the GNB and BNB algorithms of NB to calculate the results before and after applying PCA to the dataset. Scores of the used datasets before and after applying PCA are mentioned in the (MC1, JM1, KC1, CM1, and PC1) Table-6.

3 Review of the literature

To improve the outcomes of SDP, Mahama et al. described their work in situations in [3]. They did this by utilizing a variety of strategies, including a ranker for dimensionality reduction, data sampling for unbalanced data, and an iterative partition filter for noise reduction in data. According to Bisi et al., [12], PCA for feature reduction in conjunction with ANN yields more accurate findings than Sensitivity Analysis (SA) for scale features in conjunction with ANN.

The possibility of reducing the dimensions of input space by eliminating irrelevant measures was examined by Rana et al. [24]. When opposed to PCA, which chooses features without affecting the representation of any feature variable, Information Gain may be utilized. Miao et al. [17] investigated several cost-sensitive feature selection strategies and integrated a cost matrix into FS methods. They demonstrated that their suggested work surpasses other conventional procedures in terms of cost.

The study by Garousi and Felderer [9] analyzes three industrial and two academic conferences (based on their representativeness and attractiveness) to address the issue of poor industry and academic collaboration in software testing. By creating and comparing word clouds based on presentation titles, the study contrasts the themes of certain conferences. The findings reveal considerable disparities in the academic community's interest in examining theoretical problems and looking for solutions to boost practitioners' testing efficiency. In addition, the writers give insight into the causes behind limited collaboration and advise on how to improve in the future.

To study specific projected flaws and analyze the degree of prediction uncertainty, Bowes et al. [5] analyze the performance of four ML classifiers. The article includes a thorough explanation of the background as well as a review of the research addressing the influence of dataset properties on prediction accuracy. Additionally, it takes into account how validation using in vivo datasets is crucial and that the great bulk of research is based on "NASA and PROMISE" libraries.

By examining the variables that influence software quality and enhancing software-related product and quality, Rawat and Dubey et al. [25] offer a variety of approaches for raising software quality. They examined a range of complexity and size metrics in addition to models including neural networks, evolutionary algorithms, and Bayesian belief networks, among others. Another excellent study was given by Surndha Naidu et al.[18]. The article's main goal was to determine the total number of issues to save time and money. Program "length, volume, difficulty, commitment, and time estimate" have all been used to classify the fault. For this, they used a decision tree classifier. They employed the ID3 classification technique to categorize faults. After that, they classified flawed patterns using a pattern mining technique. They implemented the suggested paradigm using Java.

Software metrics have been used in a variety of software defect prediction methods that have been suggested [13, 30, 31]. These methods may be separated into two categories: unsupervised defect prediction methods and supervised defect prediction methods. Approaches for supervised defect prediction that draw on past databases for a defect prediction model's training [30]. Lyu and Guo [13] used the stacking generalization technique and the pseudoinverse learning method to create a software reliability growth model. Additionally, they

used an SVM (support vector machine) to assess the quality of software [?]. An RF algorithm was utilized by Hata et al. [14] to construct a software fault prediction model at the method level using method-level stats from the past. Defect proneness may be predicted using unsupervised defect prediction techniques without the need for a fault collection [31]. These methods can be applied when a training batch of data is either unavailable or inadequate [32]. According to Yang et al. [31], an unsupervised method that uses the reciprocal of the raw value of each change to rank the change metrics in decreasing order measure.

In this work [15] author compares nine open-source Java programs from the PROMISE repository for SDP. To do this, the author uses four primary feature extraction techniques: "auto-encoders, kernel-based principal component analysis (K-PCA), PCA, and LDA" along with support vector machines (SVM) as the base classifiers for ML. Model validation is carried out using the ten-fold cross-validation approach, and model efficiency is computed using accuracy and ROC-AUC. According to the study's findings, auto-encoders are a highly useful method for successfully reducing the software defect dataset's measurements of defects. A summary of a few earlier SDP research shows the drift and trends in the literature on feature extraction and selection in SDP. Liu and colleagues [29] investigated the effects of around thirty-two feature selection techniques, including "filter-based, wrapper-based, clustering-based, and extraction-based techniques", using the NASA dataset. A novel feature reduction strategy based on Decision tree induction was proposed by Gayatri et al. [10]. It was compared to the RELIEF method and was shown to perform better.

3.1 Conventional Vs Predictive Approaches for SDP

When comparing the traditional and predictive approaches specifically within the context of software defect prediction, there are key differences in their methodologies and effectiveness:

3.1.1 Conventional Approach for SDP

- Relies on Historical Data: Often uses historical data of defects from previous projects to identify patterns and risks.
- Rule-Based Models: May employ simple rule-based models or heuristic approaches based on past experiences or expert opinions to predict potential defects.
- Manual Inspection and Experience: Relies on manual code inspection, experience, and subjective judgment of developers or testers to anticipate potential defect-prone areas.
- Limited Data Analysis: Might lack sophisticated data analysis techniques and instead relies more on domain knowledge and past occurrences.
- Limited Scalability: May not handle large datasets or complex patterns effectively due to reliance on human judgment and simpler models.

3.1.2 Predictive Approach for SDP

- Utilizes Machine Learning: Emphasizes the use of machine learning algorithms to analyze and predict defects based on various software metrics, historical data, and patterns.
- Feature Engineering: Engages in comprehensive feature engineering, selecting relevant metrics and potentially generating new features to enhance prediction accuracy.
- Quantitative Analysis: Focuses on statistical and quantitative analysis of data to identify correlations and patterns related to defect occurrence.
- Automated Prediction: Leverages automated models that continuously learn and adapt, making predictions based on updated data.
- Scalability and Accuracy: Often exhibits better scalability and accuracy, especially with large and diverse datasets, due to the capabilities of machine learning algorithms.

3.1.3 Comparison between conventional and predictive approaches

- Accuracy and Precision: Predictive approaches, especially machine learning-based

Fig. 5 Difference among Conventional and Predictive Approaches

methods, tend to offer higher accuracy and precision in identifying potential defects compared to traditional approaches.

- Data Processing: Predictive approaches involve more sophisticated data processing and analysis techniques, allowing for a deeper understanding of patterns related to defects.
- Adaptability: Predictive approaches are more adaptable and can evolve over time with new data and improved models, while traditional approaches might be limited by static rules or heuristics.
- Automation: Predictive approaches often automate the defect prediction process, reducing the reliance on manual inspection and subjective judgment.

4 Why we use SDP ?

SDP is mainly described as a back-down from the requirement or specification of software [8]. To improve the quality of software and reduce failures we can perform unit testing, code review or defect prediction, etc. These activities are also known as quality assurance activities. The cost of these activities is approximately 75 to 80 percent of the overall budget of a project [21]. If we want to reduce the cost, we must find the defective modules first. For this, SDP has been introduced [19]. If we can predict defects of software we can reduce software development costs and increase the quality of the software. So With the help of SDP, we can do the following:

- Find the software bugs in the early stage
- Allocate the test resources efficiently
- Minimize the development cost of software
- Boost the software's 'productivity and quality'

4.1 Flow of SDP

Dataset plays a crucial role in any kind of prediction. It contains all the details of the software and its modules. After preprocessing the dataset we can divide this data set for training and testing into 70 X 30,75 X 25 or 80 X 20 ratios to train and test our module of your software. We find defects in the module if the module is defective then we can try to fix this defect. After fixing this defect problem we can again test whether the module is defective or not, if it is defective again we can find the solution to the defect, after applying this solution to the module we can again test the module. We can do this process again and again until we get a defect-free module. This is the general flow to SDP which will take several time to process. This phase of testing is very costly and takes a lot of time. Figure 4 shows the general flow of SDP.

4.2 Used Process or Steps for SDP

Figure 6 shows the whole process of SDP that we are using. To calculate the result we are using the "CM1, JM1, KC1, MC1, PC1" dataset of the NASA repository. For better results, it is important to preprocess the dataset. Preprocessing in dataset preparation for ML involves a series of steps to clean, transform, and prepare the data before feeding it into a ML model. The goal is to improve the quality of the data and enhance the model's ability to learn patterns and make accurate predictions.

After preprocessing we divided the dataset for training and testing (for training 80 percent and for testing 20 percent), then applied SVM, RF, and NB (GNB, CNB) algorithms to calculate the results. After calculating the results we applied PCA to decrease the dataset's dimensionality and selected 20 principal components of each dataset for analysis. Based on principal components, we created new datasets with 20 principal components and again applied SVM, RF, and NB (GNB,

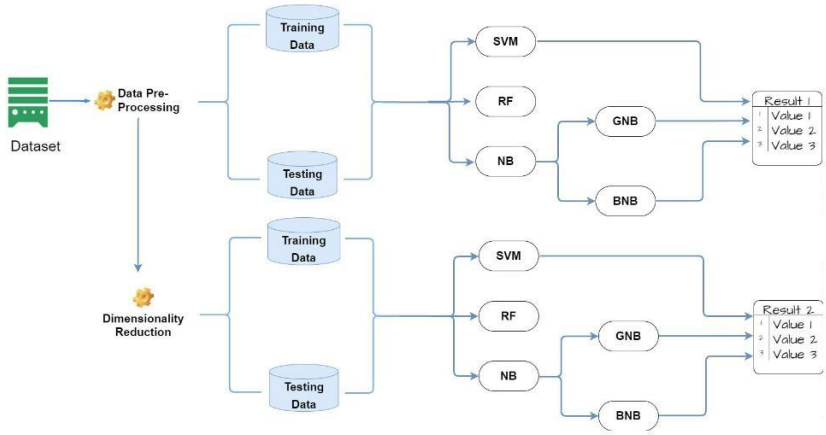

Fig. 6 SDP Process

CNB) algorithms to calculate the results. The results of all the algorithms increased on every dataset of the NASA repository. Tables 5, 6, 7, 8, 9, and 10 show the results of all the algorithms on every dataset before using PCA and after using PCA.

5 Result and Discussion

We have calculated the result of various used datasets by applying all the selected algorithms and the following results are obtained:

5.1 Result of SVM

We have calculated the results of various used datasets by applying the SVM algorithm and the above results are obtained. The detailed analysis says:

Used Algorithm	Used Dataset	Result Without Using PCA	Result After Using PCA
SVM	CM1	0.86	0.98
	JM1	0.78	1.00
	KC1	0.74	0.99
	MC1	0.97	1.00
	PC1	0.89	0.99

Table 2 Result of SVM before and after using PCA

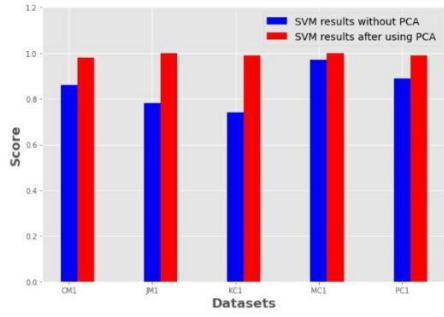

Fig. 7 Result of SVM with and without using PCA

- CM1 dataset- when we applied the SVM (Support Vector Machine) algorithm on the CM1 dataset we obtained a 0.86 result but when we used PCA (Principal component analysis) on the result, the result increased to 0.98. We have seen a growth of 0.12 in the data.
- JM1 dataset- when we applied the SVM (Support Vector Machine) algorithm on the JM1 data set we obtained a 0.78 the result but when we used PCA (Principal Component Analysis) on the result, the result increased to 1.00. We have seen a growth of 0.22 in the data.
- KC1 dataset- when we applied the SVM algorithm on the KC1 data set we obtained 0.74 as the result but when we used PCA (Principal

ResMilitaris,vol.13,n°, I1 ISSN: 2265-6294 Spring (2023)

Component Analysis) on the result, the result increased to 0.99 we saw a growth of 0.25 in the data.

- MC1 dataset- On the MC1 data set when we applied the SVM algorithm we obtained 0.97 as the result but when we used the PCA (Principal component Analysis) result, the result got increased to 1.00 which shows a growth of 0.03 in the data.
- PC1 dataset- using the SVM algorithm we obtained 0.89 as the result but when we used PCA (Principal Component Analysis) on the result, it increased to 0.99 which shows a growth of 0.10 in the result.

Based on these result sets we prepared a bar chart to visualize the result, which we can see in Figure 7.

5.2 Result of RF

We have calculated the results of various used datasets by applying the RF algorithm and the above results are obtained. The detailed analysis says:

Used Algorithm	Used Dataset	Result Without Using PCA	Result After Using PCA
RF	CM1	0.98	1.00
	JM1	1.00	1.00
	KC1	1.00	0.98
	MC1	1.00	1.00
	PC1	1.00	0.96

Table 3 Result of RF before and after using PCA

- CM1 dataset- Also applying the RF (Random Forest) algorithm on the same data set we got the result 0.98 but when we used PCA on the same it got increased to 1.00 which shows a growth of 0.02 in the result.
- JM1 dataset- Applying the RF algorithm on the given data set we got 1.00 as the result. But after applying PCA to the result it doesn't get changed, it remains constant.
- KC1 dataset- Also applying the RF algorithm on the given data set the result remains 1.00 but after using PCA it decreased to 0.98.
- MC1 dataset- Applying the RF algorithm on the given data set the result remains 1.00 before and after using PCA on the given data set.

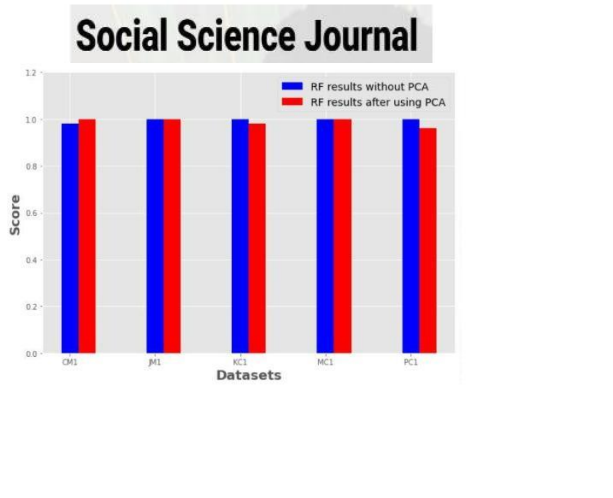

Fig. 8 Result of RF with and without using PCA

- PC1 dataset- On applying the RF algorithm on the given data set the result remains 1.00 but after using PCA on it, it decreased to 0.96 which shows a decrease of 0.04.

Based on these result sets we prepared a bar chart to visualize the result, which we can see in Figure 8.

5.3 Result of GNB

We have calculated the results of various used datasets by applying the GNB algorithm and the above results are obtained. The detailed analysis says:

Used Algorithm	Used Dataset	Result Without Using PCA	Result After Using PCA
GNB	CM1	0.85	0.97
	JM1	0.78	0.86
	KC1	0.74	0.82
	MC1	0.96	0.97
	PC1	0.84	0.95

Table 4 Result of GNB before and after using PCA

- CM1 Dataset- When we apply the GNB algorithm on the CM1 dataset the result we obtained is 0.85 and after using PCA the result we get is 0.97. Here we observed that after using PCA the result increased by 0.12.
- JM1 Dataset - When we apply the GNB algorithm on the JM1 dataset we get 0.78 as a result but after using PCA the result increased to 0.86. Here we see that after using PCA the result increased by 0.8.

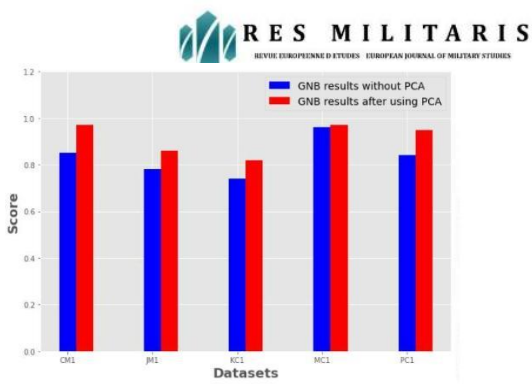

Fig. 9 Result of GNB with and without using PCA

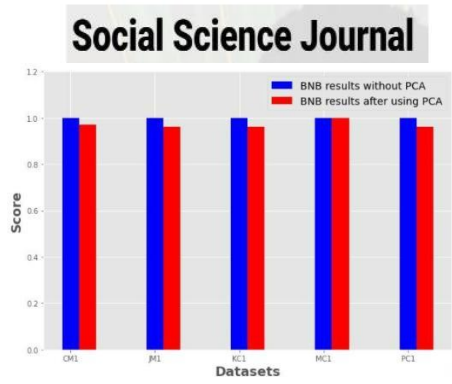

Fig. 10 Result of BNB with and without using PCA

- KC1 Dataset - When we apply the GNB algorithm on the KC1 dataset the result we obtained is 0.74 after using PCA the result we get is 0.82. Here we observed that after using PCA the result is increased by 0.8.
- MC1 Dataset - When we apply the GNB algorithm on the MC1 dataset we get 0.96 as a result but after using PCA the result increased to 0.97. Here after using PCA, we observed a small increment in the result by 0.1.
- PC1 Dataset - When we apply the GNB algorithm on the PC1 dataset the result we obtained is 0.84 after using PCA the result we get is 0.95. Here we observed that after using PCA the result increased by 0.11.

Based on these result sets we prepared a bar chart to visualize the result, which we can see in Figure 9

5.4 Result of BNB

We have calculated the results of various used datasets by applying the BNB algorithm and the above results are obtained. The detailed analysis says:

Used Algorithm	Used Dataset	Result Without Using PCA	Result After Using PCA
BNB	CM1	1.00	0.97
	JM1	1.00	0.96
	KC1	1.00	0.96
	MC1	1.00	1.00
	PC1	1.00	0.96

Table 5 Result of BNB before and after using PCA

- CM1 Dataset- When we apply the BNB algorithm on the CM1 dataset the result we get is 1.00 after using PCA the result decreased to 0.97. Here we see that after using PCA the result decreased by 0.3.
- JM1 Dataset- When the BNB algorithm is applied to the JM1 dataset the result we obtained is 1.00 but after using PCA we get 0.96. Here we see that after using PCA the result decreased by 0.4.
- KC1 Dataset - When the BNB algorithm is applied to the KC1 dataset the result we obtained is 1.00 but after using PCA, we get 0.96. This means after using PCA the result decreased by 0.4 again.
- MC1 Dataset - When the BNB algorithm is applied to the MC1 dataset the result we obtained is 1.00 after using PCA we get the same result 1.00. This means before and after using PCA the result was not changed.
- PC1 Dataset - is BNB algorithm on PC1 dataset the result we get is 1.00 after using PCA the result decreased to 0.96.

Based on the results of BNB we prepared a bar chart to visualize the result which is given in Figure 10. For better analysis of the results of all the selected algorithms over the selected dataset, we calculated bar charts that show the results of the ”SVM, RF, GNB, and BNB” algorithms over the ”CM1 dataset” in Figure 11. The results of the ”SVM, RF, GNB, and BNB” algorithms over the ”JM1 dataset” are displayed in Figure 12; the same as the results over the ”KC1 dataset” are displayed in Figure 13; the results over the ”MC1

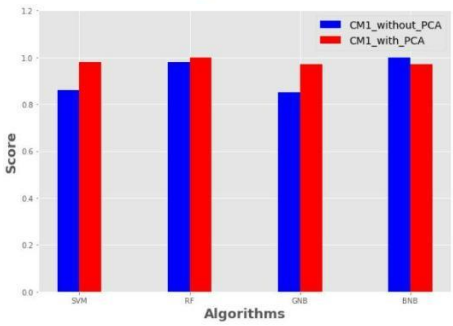

Fig. 11 Result of SVM, RB, and NB on dataset CM1 before using PCA

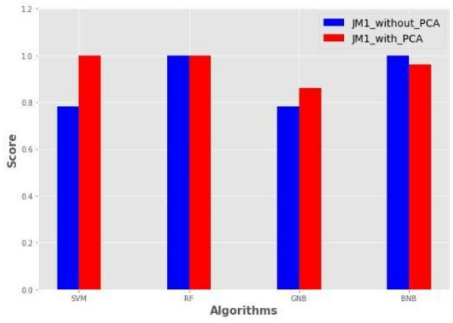

Fig. 12 Result of SVM, RB, and NB on dataset JM1 before using PCA

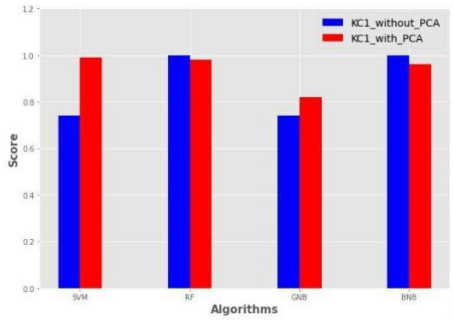

Fig. 13 Result of SVM, RB, and NB on dataset KC1 before using PCA

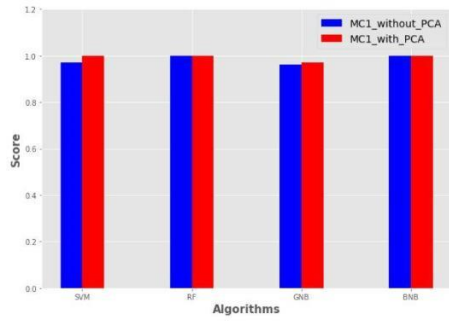

Fig. 14 Result of SVM, RB, and NB on dataset MC1 before using PCA

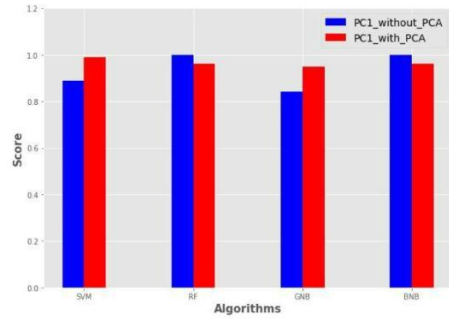

Fig. 15 Result of SVM, RB, and NB on dataset PC1 before using PCA

dataset" are displayed in Figure 14; and the results over the "MC1 dataset" are displayed in Figure 15. Figure 16, and Figure 17 show the results of selected algorithms over selected datasets before the selection of principal components and after the selection of principal components.

Based on the result we observed that the selection of a good dataset is one part, and the selection of the dependent and independent variable of the dataset is another part of a good result. In this work, the "MC1, JM1, KC1, CM1, and PC1" datasets we use from the NASA repository for analysis. And calculated results using SVM, RF, GNB, and BNB algorithms of the NB algorithm. We used the PCA algorithm for dimensionality reduction of the dataset. The process of removing variables from a training dataset to create machine

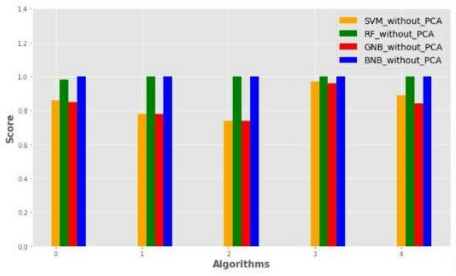

Fig. 16 Result of SVM, RB, and NB before using PCA

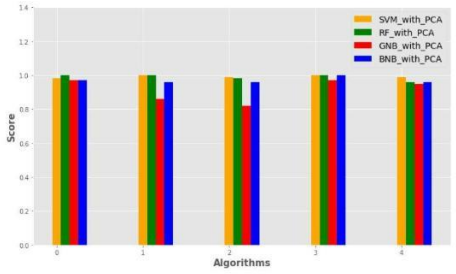

Fig. 17 Result of SVM, RB, and NB after using PCA

learning models is known as ”dimensionality reduction”. We observed that when we applied the SVM algorithm to the selected datasets results were 0.86, 0.78, 0.74, 0.97, 0.89. But when we applied the PCA algorithm to the dataset and on the calculated dimensions when we again applied the SVM algorithm, results were increased to 0.98, 1.00, 0.99, 1.00, 0.99. The same steps were with the RF algorithm and observed that the result of RF on the dataset CM1 was increased from 0.98 to 1.00, and the result of RF on the dataset JM1 and MC1 was not changed, it was 1.00 but the result on the dataset KC1, and PC1 were decreased from 1.00 to 0.98, and 0.96 respectively, But when we applied GNB and BNB algorithms of the NB algorithm on the same dataset, the results of the GNB algorithm increased from .085 to .97 on the CM1 dataset, from .78 to .86 on the JM1 dataset, .74 to .82 on the KC1 dataset, from .96 to .97 on MC1 dataset and from .84 to .95 on PC1 dataset. However, the results of the BNB algorithm decreased from 1.00 to .97 on the CM1 dataset, from 1.00 to .96 on the JM1 dataset, from 1.00 to .96 on the KC1 dataset, and from 1.00 to .96 on the PC1 dataset. Only the results of dataset MC1 were not changed it was 1.00 before applying PCA and after applying it was still 1.00.

6 Conclusion

Defect-free software is constantly desired to minimize testing costs and maintain software quality for customer satisfaction. SDP is a technique used in software engineering to anticipate or predict the presence of defects, bugs, or issues before a software system is made public. In this paper, we analyzed and compared five datasets “MC1, JM1, KC1, CM1, and PC1” of the NASA repository. At first, We used SVM, RF, GNB, and BNB algorithms of NB algorithms to calculate the result of each dataset. We used PCA for dimensionality reduction and calculated the result before and after applying PCA to all the algorithms. Based on the results we can say that if we use the PCA algorithm on the ”CM1, JM1, KC1, MC1, and PC1” dataset and again apply SVM, RF, GNB & BNB algorithms the results of GNB and SVM algorithms will be increased.

7 Declaration of interests

The authors state that none of the work presented in this study may have been influenced by any known conflicting financial interests or personal ties.

References

[1] Abdullah Alsaeedi and Mohammad Zubair Khan. Software defect prediction using supervised machine learning and ensemble techniques: a comparative study. Journal of Software Engineering and Applications, 12 (5):85–100, 2019.

[2] Ishani Arora, Vivek Tetarwal, and Anju Saha. Open issues in software defect prediction. Procedia Computer Science, 46:906–912, 2015.

[3] Kamal Bashir, Tianrui Li, Chubato Wondaferaw Yohannese, and Yahaya Mahama. Enhancing software defect prediction using supervised-learning based

framework. In 2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE), pages 1–6. IEEE, 2017.

[4] Iqra Batool and Tamim Ahmed Khan. Software fault prediction using deep learning techniques. Software Quality Journal, pages 1–40, 2023.

[5] David Bowes, Tracy Hall, and Jean Petri´c. Software defect prediction: do different classifiers find the same defects? Software Quality Journal, 26:525–552, 2018.

[6] Mohammad Sh Daoud, Shabib Aftab, Munir Ahmad, Muhammad Adnan Khan, Ahmed Iqbal, Sagheer Abbas, Muhammad Iqbal, and Baha Ihnaini. Machine learning empowered software defect prediction system. 2022.

[7] Geanderson Esteves, Eduardo Figueiredo, Adriano Veloso, Markos Viggiato, and Nivio Ziviani. Understanding machine learning software defect predictions. Automated Software Engineering, 27(3-4):369–392, 2020.

[8] Norman E Fenton and Martin Neil. A critique of software defect prediction models. IEEE Transactions on software engineering, 25(5): 675–689, 1999.

[9] Vahid Garousi and Michael Felderer. Worlds apart: industrial and academic focus areas in software testing. IEEE software, 34(5):38–45, 2017.

[10] N Gayatri, Savarimuthu Nickolas, AV Reddy, S Reddy, and A Nickolas. Feature selection using decision tree induction in class level metrics dataset for software defect predictions. In Proceedings of the world congress on engineering and computer science, volume 1, pages 124–129, 2010.

[11] MANJUBALA BISI NEERAJ KUMAR GOYAL. Early prediction of software fault-prone module using artificial neural network. International Journal of Performability Engineering, 11(1):44, 2015.

[12] Ping Guo and Michael R Lyu. A pseudoinverse learning algorithm for feedforward neural networks with stacked generalization applications to software reliability growth data. Neurocomputing, 56:101–121, 2004.

[13] Hideaki Hata, Osamu Mizuno, and Tohru Kikuno. Bug prediction based on fine-grained module histories. In 2012 34th international conference on software engineering (ICSE), pages 200–210. IEEE, 2012.

[14] Herb Krasner. The cost of poor quality software in the us: A 2018 report. Consortium for IT Software Quality, Tech. Rep, 10, 2018.

[15] Linsong Miao, Mingxia Liu, and Daoqiang Zhang. Cost-sensitive feature selection with application in software defect prediction. In Proceedings of the 21st international conference on pattern recognition (ICPR2012), pages 967–970. IEEE, 2012.

[16] M Surendra Naidu and N Geethanjali. Classification of defects in software using decision tree algorithm. International Journal of Engineering Science and Technology, 5(6): 332, 2013.

[17] Jaechang Nam and Sunghun Kim. Heterogeneous defect prediction. In Proceedings of the 2015 10th joint meeting on foundations of software engineering, pages 508–519, 2015.

[18] Jalaj Pachouly, Swati Ahirrao, and Ketan Kotecha. A bibliometric survey on the reliable software delivery using predictive analysis. Libr. Philos. Pract, 2020:1–27, 2020.

[19] Strategic Planning. The economic impacts of inadequate infrastructure for software testing. National Institute of Standards and Technology, 1, 2002.

[20] Lei Qiao, Xuesong Li, Qasim Umer, and Ping Guo. Deep learning-based software defect prediction. Neurocomputing, 385:100–110, 2020.

[21] Bhanu Pratap Rai, CS Raghuvanshi, and Ashutosh Kumar Singh. Prediction of software defect using featureextraction technique: A study. NeuroQuantology, 20(14): 2479, 2022.

[22] Zeeshan Ali Rana, Mian M Awais, and Shafay Shamail. Impact of using information gain in software defect prediction models. In International Conference on Intelligent Computing, pages 637–648. Springer, 2014.

[23] Mrinal Singh Rawat and Sanjay Kumar Dubey. Software defect prediction models for quality improvement: a literature study. International Journal of Computer Science Issues (IJCSI), 9(5):288, 2012.

[24] Hao Wang, Weiyuan Zhuang, and Xiaofang Zhang. Software defect prediction based on gated hierarchical lstms. IEEE Transactions on Reliability, 70(2):711–727, 2021. doi: 10.109/TR.2020.3047396.

[25] Zhou Xu, Jin Liu, Zijiang Yang, Gege An, and Xiangyang Jia. The impact of feature selection on defect prediction performance: An empirical comparison. In 2016 IEEE 27th international symposium on software reliability engineering (ISSRE), pages 309–320. IEEE, 2016.

[26] Meng Yan, Yicheng Fang, David Lo, Xin Xia, and Xiaohong Zhang. File-level defect prediction: Unsupervised vs. supervised models. In 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pages 344–353. IEEE, 2017.

[27] Yibiao Yang, Yuming Zhou, Jinping Liu, Yangyang Zhao, Hongmin Lu, Lei Xu, Baowen Xu, and Hareton Leung. Effort-aware just-in-time defect prediction: simple unsupervised models could be better than supervised models. In Proceedings of the 2016 24th ACM SIGSOFT international symposium on foundations of software engineering, pages 157–168, 2016.

[28] Shi Zhong, Taghi M Khoshgoftaar, and Naeem Seliya. Unsupervised learning for expert-based software quality estimation. In HASE, pages 149–155. Citeseer, 2004.