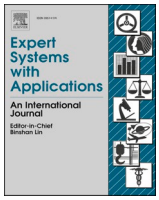
Expert Systems With Applications 253 (2024) 124266

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa



Depth linear discrimination-oriented feature selection method based on adaptive sine cosine algorithm for software defect prediction

Abdullah B. Nasser [a,e,*], Waheed Ali H.M. Ghanem [b,f], Abdul-Malik H.Y. Saad [c,e], Antar Shaddad Hamed Abdul-Qawy [g], Sanaa A.A. Ghaleb [d,f], Nayef Abdulwahab Mohammed Alduais [h], Fakhrud Din [i], Mohamed Ghetas [j]

[a] School of Technology and Innovation, University of Vaasa, Vaasa, Finland
[b] Faculty of Computer Science and Mathematics, Universiti Malaysia Terengganu, Kuala Terengganu, Malaysia
[c] College of Engineering, University of Buraimi, Buraimi, Oman
[d] Faculty of Informatics and Computing, Universiti Sultan Zainal Abidin, Terengganu, Malaysia
[e] Faculty of Computer Science and Engineering, Hodeidah University, Al Hudaydah, Yemen
[f] Faculty of Engineering University of Aden, Yemen
[g] Faculty of Science, Abdulrahman Al-Sumait University, Zanzibar, Tanzania
[h] Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia, Parit Raja, Batu Pahat, Johor, Malaysia
[i] Department of Computer Science & IT, University of Malakand, Pakistan
[j] Department of Computer Science, Galala University, Suez, Egypt

ARTICLE INFO

Keywords:
Software defect prediction
Machine learning
Feature selection
Metaheuristic algorithms
Sine cosine algorithm
Linear discriminant analysis

ABSTRACT

Software Defect Prediction (SDP) plays a vital role in the software development life cycle as it helps identify and fix software defects. However, predicting software defects with irrelevant features and overlapping classes is challenging and can lead to lengthy training and low model accuracy. To address these challenges, this research introduces a novel Depth Linear Discrimination-Oriented Feature Selection Method based on Adaptive Sine Cosine Algorithm, named Depth Adaptive Sine Cosine Feature Selection (DASC-FS). DASC-FS integrates the Adaptive Sine Cosine Algorithm (ASCA) as a search algorithm to determine the relevant features and adopts Depth Linear Discriminant Analysis (D-LDA) to identify the discriminative features that maximize class separation. The paper proposes ASCA which is a metaheuristic algorithm meticulously designed to enhance the search capabilities of the standard Sine Cosine Algorithm (SCA). Combining the simplicity of the SCA with the efficiency of multiple mutation operators inspired by Genetic Algorithms (GA), ASCA enhances the diversity of the solutions and imparts remarkable adaptability to various situations. Furthermore, this study introduces a novel linear discriminant method, called Depth Linear Discriminant Analysis (D-LDA) to enhance the robustness of the original LDA. D-LDA systematically integrates the matrix depth concept into LDA, offering a systematic approach to address the challenges associated with scatter matrix estimation. As matrix depth measures how central or deep a particular matrix is within a distribution with respect to different directions, it is an efficient tool for computing a robust scatter matrix estimator that can handle outliers and complex data structures. The experimental results showed that DASC-FS consistently obtains the highest accuracy compared to most existing methods by integrating ASCA and D-LDA, thereby considering both accuracy optimization and class separation. The results also show that the use of multiple mutation operators in ASCA improves the search process capabilities. The results also show that the capacity of D-LDA to reduce data dimensionality and increase class separation yields highly competitive results compared to other LDAs. Finally, features related to code size and complexity have emerged as key factors for SDP because they consistently rank as important features across different classifiers and datasets. DASC-FS offers a valuable solution in domain knowledge for enhancing

* Corresponding author.
E-mail addresses: Nasser.abdullah@uwasa.fi (A.B. Nasser), waheedghanem@umt.edu.my (W.A.H.M. Ghanem), abdulmalik.h@uob.edu.om (A.-M.H.Y. Saad), antarabdulqawy@sumait.ac.tz (A.S.H. Abdul-Qawy), sanaaabduljabbar@unisza.edu.my (S.A.A. Ghaleb), nayef@uthm.edu.my (N.A.M. Alduais), fakhruddin@uom.edu.pk (F. Din), Mohamed.Ghetas@gu.edu.eg (M. Ghetas).

https://doi.org/10.1016/j.eswa.2024.124266
Received 4 February 2024; Received in revised form 19 April 2024; Accepted 15 May 2024
Available online 20 May 2024

predictive accuracy and understanding factors contributing to software defects through enhanced search capabilities, robust scatter matrix estimation, and the ability to reduce data dimensionality.

1. Introduction

Software quality can be influenced by several factors, such as the number of lines in the code, length of functions and methods, number of operations, number of logical errors, complexity measurements, testability, maintainability, and number of comments. Organizations use these factors to assess the correctness of software before releasing it into the market (Curcio, Malucelli, Reinehr, & Paludo, 2016; Maxim & Pressman, 2014). In software prediction, specifically SDP, these factors are the core components in predicting and representing the features or attributes of the software modules. SDP helps software engineers prioritize their limited resources by predicting which software modules are more likely to have defects (Menzies, Greenwald, & Frank, 2006), using historical data and machine learning algorithms, to classify new software modules as normal or abnormal with a certain probability (Shepperd, Bowes, & Hall, 2014). However, the increasing volume of data in recent years has posed a challenge for data-based machine learning algorithms because redundant and irrelevant features can affect the SDP performance. To address this issue, many researchers have focused on designing and developing feature selection methods that can reduce redundant and irrelevant features, to improve SDP performance (Balogun et al., 2020; Mohammad, Imtiaz, Shakya, Almadhor, & Anwar, 2022).

Feature Selection (FS) is a machine learning process in which a subset of relevant features is selected to design the model (Mastery, 2018). The goal of FS is to determine the best feature combination that maximize the performance of the model. FS techniques are often used to simplify learning models, make them more interpretable, and reduce training time. In the context of SDP, the selection of the most relevant features that can accurately identify potential software system defects before they occur is a key challenge. The features in SDP encompass various software metrics and include code complexity measurements, such as the number of code lines, blank lines, operators, total operands, and branches of the flow graph, as well as development effort and time, to name a few. By selecting only appropriate features, FS methods improve the accuracy of SDP models by reducing the irrelevant features that introduce noise into the model and affect prediction performance. FS can mitigate the challenges posed by large and complex datasets and streamline the software development process. There are several approaches to FS, including filter, wrapper, and embedded methods (Benala & Tantati, 2022; Pandey & Tripathi, 2021; Wahono, 2015). Filter methods are based on the characteristics of individual features such as their correlations with the target variable or their variances. These methods, although simple and efficient, do not consider the interactions between features or the impact of features on model accuracy. In contrast, wrapper methods that select features based on the accuracy of the model can maximize testing accuracy and provide a more accurate measure of feature importance but incur computational expenses. Embedded methods combine the benefits of both the filter and wrapper methods by learning the feature importance during the model training process. However, these methods are also computationally expensive (Ullah et al., 2021).

Consequently, metaheuristic algorithms can be used to search through the space of possible feature subsets to identify a set of features that maximize the performance and accuracy of the model (Yu & Liu, 2004). One advantage of using metaheuristic algorithms in FS is that they can find relevant features in a reasonable time, even if the search space is significantly large or the optimization problem is non-convex (Agrawal, Abutarboush, Ganesh, & Mohamed, 2021; J. Yang & Honavar, 1998). Many different metaheuristic algorithms can be used for FS, including Simulated Annealing (SA), Genetic Algorithm (GA), Ant Colony Algorithm (ACA), particle swarm optimization, Harmony Search (HS), Firefly Algorithm (FA), Cuckoo Search (CS), and Flower Pollination Algorithm (FPA) (Anbu & Anandha Mala, 2019; Dokeroglu, Deniz, & Kiziloz, 2022; Goyal, 2022a; Lee, Choi, Ryu, & Kim, 2022; Malhotra, Nishant, Gurha, & Rathi, 2021; Sharma & Kaur, 2021; Wahono & Herman, 2014), to name a few.

The SCA is a metaheuristic algorithm based on the trigonometric sine and cosine functions (Mirjalili, 2016). SCA is relatively straightforward and only requires that common parameters such as population size and maximum iterations be tuned. Furthermore, SCA is a relatively new optimization algorithm; however, it has obtained promising results in several applications. In contrast, GA (Mitchell, 1998) is one of the most well-known metaheuristic algorithms. The performance of the GA depends on the quality of the genetic representation, the choice of crossover and mutation operators, and the specific problem being solved. The GA operators, including mutation operators, introduce new genetic material into the population for the search process. This can help prevent the algorithm from being trapped in a local optimum, and a larger search space can be explored. Hence, the choice of the mutation operator can affect the performance of the GA. For example, some mutation operators may be more effective in exploring the search space, whereas others may be more effective in escaping from local optima (De, 1989; Deb, 2011). Therefore, this study proposes a new metaheuristic algorithm based on SCA and multiple mutation operators of GA, called adaptive SCA (ASCA). ASCA can adapt itself by switching between mutation operators based on performance. ASCA was used as a core implementation search algorithm for developing the new FS method called DASC-FS, which can be employed to build a machine-learning model for SDP. The DASC-FS aims to select the most relevant features using ASCA, to maximize model accuracy.

However, the overlap between classes and presence of outliers are common problems in SDP, leading to inaccurate results (Gong, Zhang, Zhang, Wei, & Huang, 2022) (L. Chen, Fang, Shang, & Tang, 2018). To address these challenges, this research proposes a new linear discriminant method called Depth Linear Discriminant Analysis (D-LDA) to enhance the robustness of the original LDA. Unlike other linear discriminant methods, such as Fisher Discriminant Analysis (FDA), Principal Component Analysis (PCA), and LDA, D-LDA attempts to address the challenges associated with scatter matrix estimation, which is the core of discriminant methods, by utilizing robust scatter estimation based on matrix depth computation, inspired by a recently published article (M. Chen, Gao, & Ren, 2018). The matrix depth measures how central or deep a particular matrix is within a distribution in different directions. The D-LDA generates the initial empirical depth matrix and then iteratively updates this matrix by moving in different directions to determine the smallest matrix depth. D-LDA incorporates the concepts of empirically initializing the depth matrix and computing the matrix depth into traditional linear discriminant methods, thereby enhancing their robustness by assigning lower weights to potential outliers during scatter matrix estimation. Moving the estimator towards the deepest point helps capture the most accurate scatter matrix estimation. Integrating the new D-LDA into the feature selection process can improve the accuracy of the classification process. D-LDA finds a linear combination of features that maximizes the separation of classes and projects the feature space onto a lower-dimensional space. Integrating ASCA and D-LDA can leverage both techniques by identifying the most relevant features using ASCA and exploring the discriminative features for the classification process.

The broader goal of this research is to develop an efficient FS method for SDP that is capable of maximizing model accuracy while mitigating the influence of irrelevant features. In pursuing this goal, this research

aims to address the following research questions:

1. Can the integrated FS method, DASC-FS, enhance the performance of SDP classification algorithms by combining ASCA and D-LDA?
2. How can DASC-FS which integrates ASCA and D-LDA be developed to improve feature selection and classification accuracy in SDP?
3. How can an enhanced ASCA be developed to effectively select relevant features for SDP while mitigating the influence of irrelevant features?
4. How can the robustness of linear discriminant methods be enhanced to address the overlapping classes and data outlier challenges in SDP?
5. How does the performance of DASC-FS compare with existing FS methods for SDP in terms of classification accuracy and mitigating irrelevant features?

In terms of software quality, this study also aims to identify and analyse the important features that maximize SDP performance. Consistently selecting important features of different datasets can help identify the subsets of features that have a high impact on software quality. This analysis will provide valuable insights into the characteristics of the selected features and help software engineers identify software features that are likely to cause defects. The contributions of this study are summarized as follows:

- The proposal of a new feature selection method called a depth linear discrimination-oriented feature selection method based on an adaptive sine cosine algorithm (DASC-FS) for SDP integrates ASCA and LDA to identify the relevant and discriminative features for SDP models.
- The development of a new metaheuristic algorithm called Adaptive SCA (ASCA) incorporates the simplicity of the SCA and the efficiency of multiple mutation operators of the GA to improve the search capabilities of the SCA.
- A new linear discriminant method, called Depth Linear Discriminant Analysis (D-LDA), is proposed to enhance the robustness of the original LDA.
- An experimental study was conducted to investigate the effectiveness of DASC-FS in selecting software features that maximize model accuracy using various datasets and classifiers.
- Identify and analyse the important features among different software datasets that can maximize the accuracy of the SDP model and influence software quality.

The remainder of this paper is organized as follows: Section 2 reviews previous research on FS and LDA methods. Section 3 presents the designs of the proposed ASCA, D-LDA, and DASC-FS, along with describing the methodology and dataset used in the experiments. It also presents the experimental findings, including comparisons of DASC-FS with other FS methods, ASCA with the original SCA, and D-LDA with other LDAs. Finally, in the conclusion, the main contributions and limitations of the proposed method are outlined, suggesting directions for future work.

2. Related works

To acquire insight into the limitations of existing studies, this section reviews related works in two subsections. The first section reviews existing SDP feature selection methods, whereas the second section reviews the improvements made to LDA and its variants.

2.1. SDP feature selection methods

Defects in software systems for improved performance and accuracy. Another filter-based FS method combines chi-square, information gain, and association filters to identify relevant features and improve the performance of the software defect model (Jia, 2018). An example of a correlation-based feature selection technique is presented in (Elish & Elish, 2008), using a corporate support vector machine (SVM) as a classifier. However, in general, the performance of the filter approach is highly dependent on the selected datasets and classifiers (Balogun et al., 2021). Furthermore, filter methods do not model feature dependencies because they neither consider the relationships between features nor their effect on each other (Pudjihartono, Fadason, Kempa-Liehr, & O'Sullivan, 2022).

The second approach in FS methods uses the wrapper as a predictive model to evaluate the effectiveness of different feature subsets. This approach typically involves iterative addition or removal of features based on their impact on the predictive performance of the model. For example, the Recursive Feature Elimination (RFE) used for SDP along with SVM (Adorada, Wirawan, & Kurniawan, 2020), can be considered a wrapper method in FS. To learn the model, the RFE method assigns a weight or importance score to each feature based on the contribution to the target variable. The least important feature with the lowest coefficient is then removed. This process is repeated until a certain number of features or stopping criteria are reached. Sequential Forward Selection (SFS) and Sequential Backward Selection (SBS) are two popular machine learning FS wrapper methods that have also been used in SDP (Singh & Haider, 2022). SFS begins with an empty feature set and adds features individually based on a performance metric. At each iteration, the algorithm evaluates the performance of the model trained on the current feature set, thereby adding the feature set that results in the largest improvement. This process is repeated until a stopping criterion is reached or all features have been added to the feature set. In contrast, SBS starts with the full feature set and removes features one at a time based on a performance metric.

Several optimization algorithms based on the FS wrapper approach have been used for SDP, including GAs, Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), Firefly Algorithms (FA), HS, and Artificial Bee Colony (ABC). A study conducted in (Wahono & Herman, 2014) proposed a wrapper-based FS method using GA in combination with a bagging technique, to enhance SDP model performance. The study by (Malhotra et al., 2021) used the Synthetic Minority Oversampling Technique (SMOTE) to address the imbalance challenge, employing PSO as the FS method and SVM as the classifier. Another study (Goyal, 2022a) proposed a new method using FS and decision trees, referring to it as a cost-sensitive decision tree based on harmony search (HS-CSDT) (Lee et al., 2022), whereby HS was used to find the subset of features as well as for parameter tuning. FA has also been used as a search method for the same problem (Anbu & Anandha Mala, 2019). In general, the results show that the use of a single optimization algorithm can produce highly competitive results. However, some studies have adopted hybridization algorithms as alternative approaches, combining the advantages of multiple algorithms to create a robust solution. One such study proposed a hybrid FS method that combines GAs and ACO (Nemati, Basiri, Ghasem-Aghaee, & Aghdam, 2009), first using GAs to select a subset of features and then using ACO to further optimise the selected subset. Hybridization based on PSO, GA, and SVM classifiers has been demonstrated to improve the performance of software fault prediction (Alsghaier & Akour, 2020). Another study used WOA along with GA operators as a search algorithm to identify a relevant subset of features for SDP (Hassouneh et al., 2021). In (Goyal, 2022b), the authors introduced a new feature selection method called Feature Selection Evolving Populations With Mathematical Diversification (FS-EPwMD) to improve the accuracy of SDP models. FS-EPwMD uses mathematical diversification for genetic evolution, to select the most relevant features and avoid local optima.

Within the same framework as hybridization, some scientists have combined filter and wrapper methods. The hybrid multi-filter wrapper FS method is an example of this approach (Balogun et al., 2021). After filtering features using the filter method, the set of selected features is reduced using the wrapper method. Another study adopted Rao

optimization and multi-criteria decision-making (MCDM) for defect prediction (Thirumoorthy, 2022). Convolutional Neural Networks (CNNs) and Deep Neural Networks (DNNs) have also been widely used in recent years for various tasks including SDP, employing CNNs and DNNs to extract high-level features from software metrics and source code to make accurate predictions (Khan et al., 2022; Zhu, Ying, Zhang, & Zhu, 2021). These neural networks have been effective in learning the complex relationships between features and target variables. Kakkar and Jain (2016) combined two feature selection approaches in their work, wrappers and filters, to propose a framework for SDP, conducting an exhaustive search using the wrapper-based method and subsequently ranking the search using the filter-based feature selection method, employing five classifiers in the framework: IBk, KStar, LWL, Random Tree, and Random Forest (RF). A recent study (Iqbal & Aftab, 2020) introduced a classification framework that leveraged the multi-filter feature selection technique and Multi-Layer Perceptron (MLP). This framework incorporates an over-sampling technique to address the imbalance problem. For the same problem, (Iqbal, Aftab, Ullah, Bashir, & Saeed, 2019) proposed a feature-selection-based ensemble framework that utilizes the bagging and boosting techniques along with RF in the classification process. These types of studies, particularly those combining and utilizing different feature selection methods with different classifiers, as well as class-balancing techniques, contribute to the growing body of literature on SDP, by providing an effective mechanism for identifying defect-prone modules.

More recent studies propose a comprehensive framework to enhance accuracy, reduce computational costs, and improve interpretability (Ali et al., 2024), integrating GA and classifiers including RF, SVM, and naïve Bayes. These classifiers are used individually to compute model accuracy. Subsequently, an ensemble voting technique is applied to select the most accurate classifier. A hybrid approach called GWOFS-MLP that integrates Gray Wolf Optimization (GWO) and MLP has also been proposed to optimize feature selection (Mustaqeem, Mustajab, & Alam, 2024). The GWO selects the relevant features, whereas MLP serves as the predictive engine to model and classify software defects. A new approach that integrates the Mutation Boosted Salp Swarm Optimizer (MB-SSO) has been proposed as a search engine to select the relevant features, using rough set theory to analyse feature relevance and dependency (Sekaran & Lawrence, 2024). The study provides new insights into the dynamic interplay between feature selection and analysis methods, opening a venue for more robust and efficient software defect prediction models. Another study investigated the effect of combining feature selection and data resampling to address imbalanced classification. Comprehensive empirical studies have been conducted on a large number of datasets including SDP. Acquiring a greater understanding of the combined effects of feature selection and data resampling on imbalanced classification, may be useful for developing effective methods in related research (Zhang et al., 2023).

The third method is the embedded approach, which ranks feature importance as part of the training process of a predictive model. This approach has been used in various machine learning algorithms, such as RF and SVM. A study by (K. Wang, Liu, Yuan, & Wang, 2021) used an embedded FS method based on the Least Absolute Shrinkage and Selection Operator (LASSO) in combination with an SVM to predict defects in software systems. The results showed that the method effectively identifies relevant features and improves the predictive performance of the model. A study by (Osman, Ghafari, & Nierstrasz, 2017) proposed an embedded FS method based on Ridge, LASSO, and ElasticNet regularization in combination with linear and Poisson regression to predict defects in software systems. The results demonstrated that the method effectively identified relevant features and improved the predictive performance of the model.

Although these studies use various techniques to address different problems related to defect prediction, the general objective is to improve the prediction accuracy of software defect models. In addition, no single algorithm can be deemed optimal for all problems in the quest for a solution, particularly when working on optimization algorithms. This also applies to FS methods, each having unique strengths and limitations depending on the various metrics and underlying principles. Table 1 presents a comparative analysis of the relevant works in the field of feature selection, highlighting the advantages and disadvantages of methods. The table reviews a wide range of FS methods, emphasising their computational complexity and the contribution of each to advancements.

Finding a better FS method for SDP motivated the ongoing research because the no-free-lunch theorem highlights the limitations of individual algorithms and the need for an optimal solution that fits different problem settings. The DASC-FS offers a unique combination of algorithmic innovation and methodological improvements to address the main limitations of existing feature selection methods in SDP. DASC-FS integrates ASCA and D-LDA, offering a novel approach that optimizes the FS process considering both accuracy optimization and class separation. The use of multiple mutation operators in ASCA allows DASC-FS to efficiently explore the search space, increasing the diversity of solutions and adaptability to various situations. Furthermore, the use of D-LDA provides a robust scatter matrix estimator that can handle outliers and complex data structures.

2.2. Linear discriminant analysis methods

Linear discriminant analysis is a Dimensionality Reduction (DR) technique that has been widely used in classification to find a projection of data points that maximize the ratio of between-class and within-class scatter. By maximizing the separability between data points, LDA can improve the accuracy of the classification process, effectively identifying and distinguishing different samples based on their features. Notably, LDA and its counterparts are not FS methods. In DR techniques, the objective is to find a set of features equivalent to the original features instead of finding the top set of features. LDA is a powerful tool for classification and dimensionality reduction, with several LDA variants and alternatives proposed; however, it has some performance limitations. In this review, we present existing works and categorize them based on the specific problems they aim to address.

One of the main problems with LDA is its sensitivity to outliers that are significantly different from the rest of the data. To address this problem, numerous studies have proposed methods, including robust measures into LDA, using the L1-norm of the projection matrix and its extension 2DLDA-T $\ell 1$ (Yang, Zheng, & Liu, 2023), integrating two-dimensional LDA and the T $\ell 1$ −norm, and using robust distance metrics such as the Mahalanobis distance, which are less sensitive to outliers. A recent example of an LDA-based approach is the Wasserstein Discriminant Analysis (WDA) (Flamary, Cuturi, Courty, & Rakotomamonjy, 2018; Kuhn, Esfahani, Nguyen, & Shafieezadeh-Abadeh, 2019), which is based on the Wasserstein distance. Similar to other discriminant algorithms, it aims to maximize the separation between classes while minimizing it within classes. Furthermore, WDA utilizes optimal transport theory to find an optimal transportation matrix that aligns with the original data.

One of the most common algorithms for discrimination is PCA (Abdi & Williams, 2010), which is used to reduce the dimensionality of data feature representations while retaining as much important information as possible. Unlike LDA, PCA is an unsupervised learning algorithm for finding a set of features or orthogonal axes called principal components that are equal to the original features, to explain most of the variance in the data. Regarding the non-Gaussian data distribution problem associated with LDA, Adaptive Discriminative Analysis (ADA) (Luo, Hou, Nie, & Yi, 2018) has been proposed to handle data with diverse distributions. ADA combines sample measurement magnitude and subspace learning to preserve the within-class local structure and simultaneously learn discriminative transformation. Fisher discriminant analysis (FLD) is a specific type of LDA designed for binary classification tasks and can be considered a method for handling diverse data distributions (Durrant

Table 1
Analysis of feature selection methods in SDP, including advantages and disadvantages, compared with DASC-FS.

Category	Reference	Description	Advantages	Disadvantages
Filter Method	(Ghotra et al., 2017)	Filter methods use statistical measures (e.g., correlation coefficients, chi-squared tests, etc.) to evaluate feature importance independently of predictive models.	- Fast computation - Independent of predictive models	- May overlook feature dependency
Filter Method	(P. Wang, 2012)	Filter FS method based on mutual information.	- Improved performance and accuracy in defect prediction.	- Highly dependent on datasets and classifiers.
Hybrid Method	(Jia, 2018)	Combines chi-square, information gain, and association filters for feature selection.	- Combines advantages of multiple technologies.	- Performance highly dependent on dataset and classifier selection.
Filter Method	(Elish & Elish, 2008)	Correlation-based feature selection technique using a SVM classifier.	- Utilises correlation information to select features. - SVMs are capable of handling high-dimensional data.	- Highly dependent on the selection of the kernel and its parameters. - Highly computational.
Wrapper Method	(Balogun et al., 2021)	A two-stage FS method; after ranking features, enhanced wrapper FS methods use predictive models to iteratively evaluate feature subsets.	- Addresses filter rank selection problem. - Reduces number of evaluation cycles. - Achieves good performance.	- Can be computationally expensive.
Filter Method	(Adorada et al., 2020)	Filter FS method that evaluates the importance of features based on internal attributes, rather than assuming independent features.	- Feature independence not assumed, making it suitable for datasets with strong feature dependencies. - Generally works well in different environments. - efficient computer programming.	- May not perform optimally in situations where features are truly independent.
Embedded Method	(Adorada et al., 2020)	SVM-RFE incorporates feature selection into the SVM classification algorithm, using coefficients from the SVM model to iteratively eliminate irrelevant features.	- By integrating feature selection in the SVM algorithm, obviates the need for separate feature selection steps. - Uses SVM coefficients to estimate feature importance, which can improve selection accuracy.	- High computational complexity
Wrapper Method	(Singh & Haider, 2022)	Popular wrapper methods include Sequential Forward Selection (SFS) and Sequential Backward Selection (SBS).	- Iteratively adds/removes features based on performance.	- Risk of overfitting if not properly controlled.
Hybrid Method	(Wahono & Herman, 2014)	Utilises GA in combination with bagging for feature selection.	- Improved predictive performance of SDP models.	- Genetic algorithms can be computationally expensive.
Hybrid Method	(Malhotra et al., 2021)	Uses PSO and Synthetic Minority Oversampling Technique (SMOTE) for feature selection with SVM classifier.	- Addresses the challenges of imbalanced data. - Improves feature selection effectiveness.	- High computational complexity - Sensitivity to parameter settings.
Hybrid Method	(Lee et al., 2022)	HS method proposed for feature selection with decision trees.	- Comprehensive parameter optimisation. - Effective in selecting relevant features.	- High computational complexity - Performance may be sensitive to algorithm parameters.
Hybrid Method	(Nemati et al., 2009)	Hybrid approach combining GAs and ACO for feature selection.	- Combines advantages of multiple algorithms. - Improved defect prediction accuracy.	- Increased complexity due to hybridization.
Hybrid Method	(Alsghaier & Akour, 2020)	Utilizes hybridization of PSO, GA, and SVM classifiers for improved performance.	- Combines advantages of multiple algorithms. - Improves defect prediction accuracy.	- Complexity in parameter tuning.
Hybrid Method	(Hassouneh et al., 2021)	Uses WOA along with GA for feature selection.	- Effective in selecting relevant feature subsets.	- Complexity in setting algorithm parameters.
Hybrid Method	(Goyal, 2022b)	Introduces the Feature Selection Evolving Populations with Mathematical Diversification (FS-EPwMD) method.	- Improved accuracy for SDP models. - Leverages metaheuristics to efficiently explore search space	- Complexity in mathematical diversification process.
Hybrid Method	(Iqbal & Aftab, 2020)	Leverages multi-filter feature selection technique with MLP.	- Combines advantages of multi-Filter FS. - Addressed class imbalance issue in SDP.	- May require extensive computational resources.
Hybrid Method	(Iqbal et al., 2019)	Proposes ensemble framework using bagging, boosting, and RF with feature selection.	- Addresses class imbalance for improved performance.	- Increased complexity due to ensemble methods.
Hybrid Method	(Ali et al., 2024)	Integrates GA, ML classifiers (e.g. RF, SVM, and naïve Bayes) and voting technique.	- Optimizes feature selection. - Enhances accuracy.	- Complexity in optimization and ensemble voting technique.
Hybrid Method	(Mustaqeem et al., 2024)	Introduces GWOFS-MLP hybrid approach integrating GWO and MLP.	- Enhanced accuracy - Optimizes feature selection with improved model performance.	- Increased complexity - Parameter sensitivity in optimization algorithms.
Hybrid Method	(Sekaran & Lawrence, 2024)	Integrates MB-SSO with rough set theory for feature selection.	- Combines the strengths of MB-SSO and rough set theory. - Provided insights into feature relevance and dependency.	- Complexity in integrating multiple techniques.
Hybrid Method	(Zhang et al., 2023)	Investigates the combined effects of feature selection and data resampling on imbalanced classification.	- Improved imbalance classification.	- Complexity in comprehensive empirical studies.
Embedded Method	(K. Wang et al., 2021)	Uses embedded FS method based on LASSO with SVM for defect prediction.	- Fast, effective, and low model complexity - Identifies relevant features within training process.	- Limited to linear feature relationships as LASSO is typically applied to linear models.

(*continued on next page*)

Table 1 (*continued*)

Category	Reference	Description	Advantages	Disadvantages
Embedded Method	(Osman et al., 2017)	Proposed embedded FS method based on ridge, Lasso, and ElasticNet regularizations with linear and Poisson regression.	- Fast, effective, and low model complexity - Identifies relevant features effectively.	- Limited to specific regression models as regularization methods are not compatible with all types of ML algorithms.
Hybrid Method	- This research - (DASC-FS FS)	- Integrates ASCA as a search algorithm to find relevant features and D-LDA to identify the discriminative features that maximize class separation.	- Integrates ASCA and D-LDA to optimize feature selection. - Considers both accuracy and class separation. - Efficient search process with multiple mutation operators. - D-LDA serves as a robust scatter matrix estimator that can handle outliers and complex data structures.	- Increased complexity - Requires fine-tuning of parameters.

& Kabán, 2010). Both FLD and LDA share the same procedure for dimensionality reduction and classification. However, FLD focuses on finding a single linear discriminant to maximize class separation, making it valuable for handling diverse data distributions. More recently, Robust Sparse Linear Discriminant Analysis (RSLDA) (Wen et al., 2018) was developed based on sparse LDA, to reduce the impact of undesirable features on linear feature combinations by applying a sparsity constraint to the linear discriminant vectors. This approach promotes feature selection by minimizing the impact of less important features, setting their coefficients to zero. RSLDA utilizes a sparse matrix, orthogonal matrix, and l2,1 norm simultaneously to improve robustness to noise and guarantee that the extracted features retain the original information of the data. Kernel LDA is an extension of LDA that was proposed to address the challenge of classifying nonlinear data, whereby kernel functions are applied to transform data into higher-dimensional spaces, to make them linearly separable. Examples of common kernel functions are the polynomial, Radial Basis Function (RBF), and sigmoid kernels, with each function being suitable for different types of data. DeepLDA, which combines a traditional DNN and LDA, has been found to be very efficient for dimensionality reduction. DeepLDA is used to project the data into a discriminative subspace, and the main task of the DNN is to learn a nonlinear mapping from the enhanced feature space to the class labels (Dorfer, Kelz, & Widmer, 2015).

3. Proposed works

This section describes the design of the proposed methods. First, the design of the Adaptive Sine Cosine Algorithm (ASCA) is presented, followed by the design of Depth Linear Discriminant Analysis (D-LDA), and finally the methodology of the DASC-FS for SDP, which is based on ASCA and D-LDA. Therefore, this section also provides an overview of SCA and LDA.

3.1. The proposed Adaptive Sine cosine algorithm

In response to the question regarding developing an enhanced ASCA that effectively selects relevant features for SDP while mitigating the influence of irrelevant features, this section presents ASCA, which is a metaheuristic algorithm that incorporates the simplicity of SCA and efficiency of GA mutation operators. SCA (Mirjalili, 2016) is a metaheuristic algorithm based on the trigonometric sine and cosine functions. It is relatively straightforward and only requires tuning common parameters, such as population size and maximum iterations. Furthermore, although SCA is a relatively new metaheuristic algorithm, it has shown promising results in several applications (Abualigah & Diabat, 2021; Guo, Wang, Dai, & Xu, 2020), generating a population of solutions, which are then subjected to improvement iterations. SCA mainly relies on two equations to improve the solution:

$$X_i^{t+1} = X_i^t + r_1 \times sin(r_2) \times \left| r_3 P_i^t - X_i^t \right| \quad (1)$$

$$X_i^{t+1} = X_i^t + r_1 \times cos(r_2) \times \left| r_3 P_i^t - X_i^t \right| \quad (2)$$

where X_i^{t+1} is the new solution generated based on the previous solution X_i^t and the best solution obtained thus far, P_i^t, withr_1, r_2 and $r_3$denoting three random numbers. The new solution is randomly generated using either Eq. 1 or 2.

However, GA relies heavily on crossover, mutation, and selection operators. The mutation operator plays a crucial role in introducing new genetic material into the population, thereby preventing the GA from becoming trapped in local optima (De Falco, Della Cioppa, & Tarantino, 2002; Salgotra & Singh, 2017).

The sine cosine algorithm is relatively simple and efficient in terms of computation, but it may get trapped in local optima because of a lack of exploration. However, by incorporating multiple mutation operators, ASCA can explore the search space more effectively. The adaptation of different mutations can have different effects on the population, allowing ASCA to dynamically adjust its behaviour based on the improvement rate. This prevents ASCA from becoming trapped in local optima, leading to better solutions. Moreover, the use of multiple mutation operators increases the diversity of the population, which can help ASCA escape from local optima and converge to global optima more efficiently and quickly. Consequently, ASCA can provide more accurate and robust results than SCA. ASCA incorporates five different mutation operators: point, inversion, scramble, swap, and Gaussian.

1. *Point mutation* randomly changes one or more genes on a chromosome by replacing them with different values.
2. *Inversion mutation* selects a subset of genes on the chromosome and reverses their order.
3. *Scramble mutation* randomly selects a subset of genes on a chromosome and shuffles their order.
4. *Swap mutation* randomly selects two genes on a chromosome and swaps their positions.
5. *Gaussian mutation* adds a small random value to each gene in the chromosome following a Gaussian distribution.

Fig. 1 illustrates the design of ASCA. First, the initial population is generated and evaluated. This population is then subjected to improvement, which includes updating the population using the SCA equations. Then, a fraction of the population is subjected to mutation operators based on Eq. (3): Given a number of mutations n, ASCA can modify itself over time, based on the following formula:

$$M_i = \begin{cases} M_i, P_i > P_{i-1} \\ M_{i+1}, P_i \le P_{i-1} \end{cases} \quad (3)$$

where m is the list of mutations i [1 … n]; n is the number of mutations; and p is the rate of improvement. At each iteration, the rate of improvement is quantified to achieve optimal or near-optimal solutions. Using this equation, ASCA can switch between mutations based on the improvement rate of the entire population. If the improvement rate of

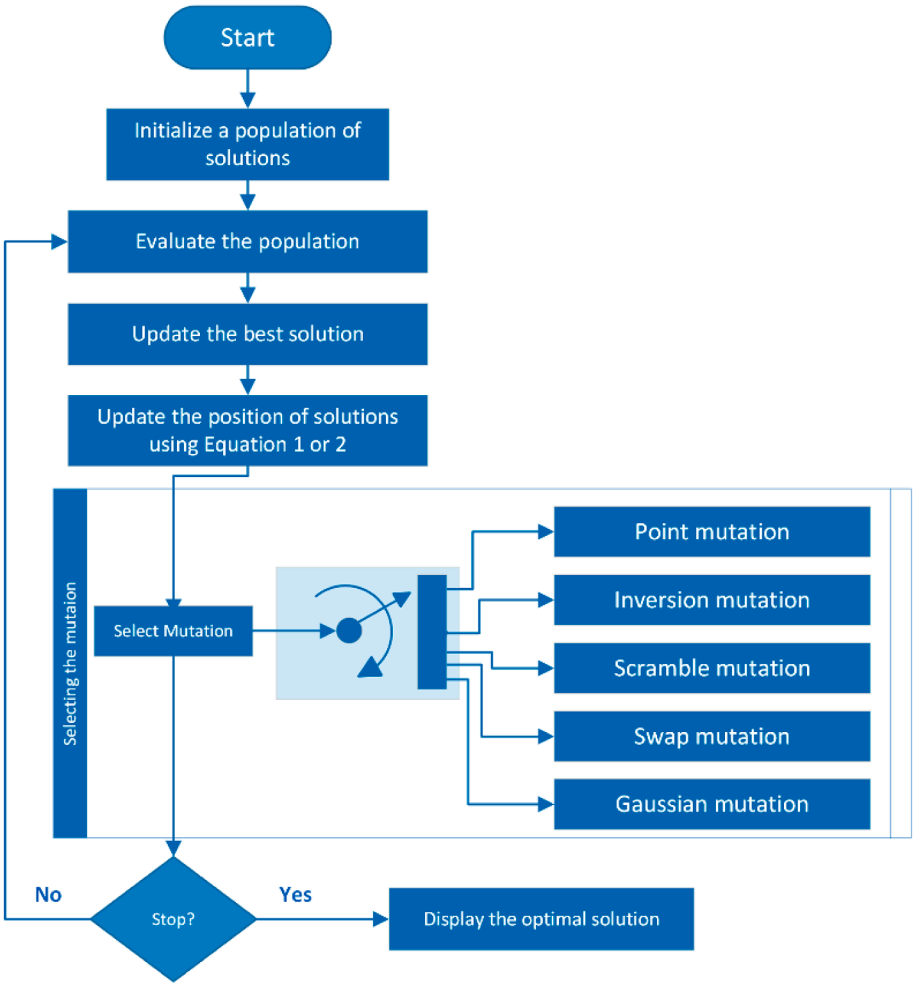


Fig. 1. Adaptive sine cosine algorithm.

the current population is better than that of the previous population, ASCA continues to use the same mutation operator; otherwise, it switches to the next mutation operator.

3.2. The proposed depth linear discriminant analysis method

To enhance the robustness of linear discriminant methods, D-LDA improves the accuracy and reliability of LDA by addressing overlapping classes, and data outliers, employing the concept of matrix depth for robust scatter matrix estimation. Matrix depth measures centrality or how far out matrix outliers are within a given distribution. The depth matrix can be used to develop a robust covariance and scatter matrix estimator for handling outliers and complex data structures. Matrix depth D(Σ, P) quantifies how well the estimator Σ captures the structure in the given data points by moving the estimator towards the deepest point to obtain the most accurate estimation of the scatter matrix that can enhance the performance of LDA. A lower matrix depth implies a greater centrality within the distribution.

For a p-variate distribution X ∼ P, the matrix depth D(Σ, P) is mathematically defined as:

$$D(\Sigma, P) = min\{u^T \Sigma u | u \in R^p, u^T \Sigma u \leq u^T X u forall X \sim P\} \quad (4)$$

In this equation, matrix depth represents the minimum value of $u^T \Sigma u$ across different vector u directions R^p that satisfy the conditions $u^T \Sigma u \geq 0$ and $u^T \Sigma u \leq u^T X u$. The condition $u^T \Sigma u \geq 0$ imposed on u in the definition ensures that the matrix Σ is positive semidefinite while the condition $u^T \Sigma u \leq u^T X u$ ensures that the matrix Σ is not deeper than any other positive semidefinite matrix X with respect to the distribution P.

Computing the scatter matrix within and between classes is an essential step in standard LDA and is the key to dimensionality reduction and class separability assessment. D-LDA systematically integrates matrix depth into LDA to compute the scatter matrix within classes. Similar to LDA, D-LDA focuses on dimensionality reduction, with the primary goal of finding a linear combination of features that best separates classes in a dataset, whereby D-LDA receives X and its class labels and generates the transformed X. The algorithm starts by calculating the scatter matrix between the classes by accumulating the outer products of the differences between the class means (μ_i) and overall mean (μ), weighted by the number of data points in each class (N_i). Subsequently, the scatter matrix between classes Sw is computed using the concept of matrix depth. Starting with the initial Sw matrix generated based on empirical data X, the algorithm computes the matrix depth over different vectors u using Eq. (3) and selects the vector with the lowest depth to update the Sw matrix by moving towards the u direction using Eq. (5), where α_t is the step size. The updating of Sw is repeated T times.

$$Sw = Sw + \alpha_t * u_t * u_t^T \quad (5)$$

The algorithm then finds the eigenvectors and eigenvalues by solving Swˆ(−1) * Sb, where the eigenvectors represent the linear combinations of features that maximise class separability and are used to transform the

data. Algorithm 1 summarises the steps of D-LDA.

Algorithm 1: Depth Linear Discriminant Analysis (D-LDA)

Input:
- Input data matrix X, where each row represents a data point, and each column represents a feature.
- Class labels y for each data point in X.
- New dimensionality number k

Output:
- Transformed data matrix X_

Algorithm Steps:

Step 1: Compute Scatter Matrices Sb and Sw
- Step 1.1: Compute Between-Class Scatter Matrix Sb:
- Initialize Sb to zero.
- For each class i from 1 to C (number of classes):
- Calculate Ni (the number of data points in class i).
- Compute the difference between the class means μi and overall mean μ of the data.
- Update Sb as Sb += Ni * (μi − μ)(μi − μ)ˆT.
- Step 1.2: Compute Within-Class Scatter Matrix Sw
- Initialize the current scatter matrix estimator Sw (as obtained from robust depth scatter matrices).
- Define a set of directions U, which can be predefined or generated based on data characteristics.
- Initialize an iteration counter t = 0.
- The following steps are repeated for t = 1 to T (a predefined number of iterations):
- For each unit vector u in the direction set U:
- **Depth Calculation (Du):** Compute the matrix depth Du(Sw, Ui) for direction u using (4).
- **Direction Selection (u_t):** Select the direction u_t that achieves the smallest matrix depth, Du. This direction points towards the deepest matrix estimator.
- **Estimator Update:** Update the current scatter matrix estimator Sw using (5)

Step 2: Eigenvectors and Eigenvalues Computation
- Solve the generalized eigenvalue problem using the final within-class scatter matrix and between-class scatter matrix Sb, to find the top k eigenvectors corresponding to the largest eigenvalues.

Step 3: Transform Data
- Transform the input data using the learned eigenvectors. X_transformed = X @ eigenvectors

Output

Return X_transformed, i.e., the transformed data matrix.

In this study, D-LDA was used to address the problem of overlapping classes by finding a linear combination of features that maximizes class separation. D-LDA is a highly efficient technique that projects the original-dimensional feature space onto a lower-dimensional space, thereby determining a projection direction that maximizes the ratio of between-class to within-class scatter to separate the classes. Fig. 2 shows how D-LDA works. In the figure, data consist of three classes before and after applying LDA. D-LDA clearly separates the data points of different classes, grouping them for each class.

3.3. Proposed depth linear discrimination-oriented feature selection method based on Adaptive Sine cosine algorithm.

The proposed feature selection method, DASC-FS, aims to improve the accuracy of the SDP models by integrating ASCA and LDA. The DASC-FS method is based on evolutionary algorithms, iteratively evaluating different combinations of features and selecting the ones that result in the best model accuracy. DASC-FS integrates ASCA and D-LDA, with the former functioning as a search algorithm to find combinations of the most relevant features and the latter identifying the most discriminative features for the classification process. The DASC-FS method involves generating and evaluating multiple feature subsets using ASCA, thereby selecting the subset that leads to the highest model accuracy. Each candidate in the population represents a binary combination of features. ASCA attempts to improve candidate fitness, which represents the test accuracy. The method trains the model using the candidate features, and then the test accuracy of each candidate is calculated. From an optimization perspective, the objective function maximizes the test accuracy of the prediction model based on different feature combinations. In the context of the DASC-FS method, the rate of improvement of ASCA quantifies the improvement achieved at each iteration i, to maximize prediction accuracy, which in practice, is the change in prediction accuracy between consecutive iterations.

After numerous iterations of improvement, including feature selection, data transformation, and evaluation, the process is completed, or the exit criteria are met. The DASC-FS can generate the best candidate solution that obtains the best test accuracy and feature score, representing the importance of the feature. The DASC-FS is incorporated into several classification models, such as RF, Logistic Regression (LR), Gradient Boosting (GB), Gaussian Naive Bayes (GNB), and K-Nearest Neighbour (KNN), to identify the most relevant features for use in the model. The process of selecting features, training, and evaluation can be broadly summarized as follows:

- Data collection and preprocessing: The datasets (detailed in the next section) are cleaned and preprocessed to ensure that they are in a suitable format for modelling.
- Feature selection: The goal of feature selection is to identify the most relevant and predictive features. In this step, the DASC-FS is used to select the features to include in the model. ASCA iteratively selects the relevant features that maximize prediction accuracy.
- Data Transformation: After selecting features using ASCA, D-LDA is applied to obtain the transformed features as linear combinations of features designed to maximize class separability. In this step, the

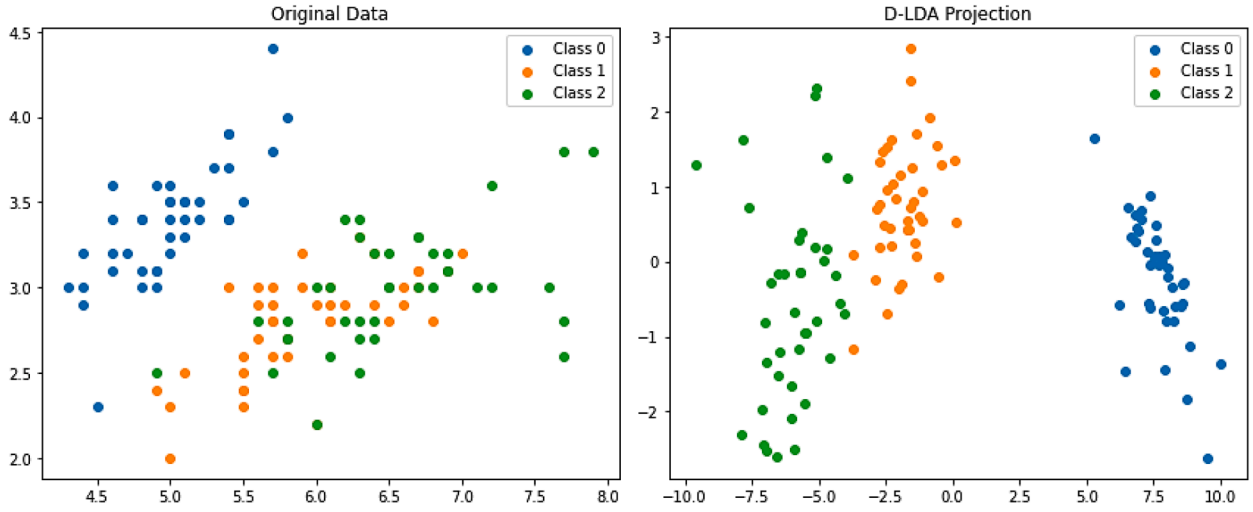


Fig. 2. Data before and after applying D-LDA.

selected features are projected onto a low-dimensional space to help accelerate the training and testing processes.

- Model training and testing: Once the features have been selected and transformed, the next step is to train the prediction model using the selected features. As this study proposes a feature selection method intended to work with different classifiers, the DASC-FS was designed to seamlessly integrate with different classifiers. Several classification methods can be used for training, such as KNN, RF, LR, GB, and GNB. The model was evaluated using a stratified k-fold cross-validation to determine its accuracy and generalizability. Stratified k-fold cross-validation was selected because of the imbalance issue in some software defect datasets.
- Model optimization and fine-tuning: Once the initial model was trained and evaluated, basic fine-tuning was performed using GridSearchCV to improve the performance of the model. This involves adjusting the hyperparameters of the model such as the learning rate, number of neighbours, number of trees, maximum depth of the trees, loss function, regularization parameters, and maximum iterations.

The proposed method, which combines the power of GA mutations, SCA, and D-LDA with the effectiveness of the wrapper approach, is expected to outperform other feature selection methods such as mutual information and RFE in SDP problems.

4. Results and discussion

To address the question of whether combining ASCA and D-LDA can enhance the performance of SDP classification algorithms, this section presents the results and the discussions conducted to evaluate the proposed methods: DASC-FS, integration of D-LDA into DASC-FS, and introduction of mutation operators into the original SCA. The evaluation consisted of four experiments.

1. The first experiment evaluated the potential of the DASC-FS method in selecting relevant features that maximize the test accuracy when applied to different datasets. In this experiment, DASC-FS was compared with other FS methods from different categories, including the original SCA (SCA-FS), GA (GA-FS), Recursive Feature Elimination (RFE-FS), embedded method (EM-FS) and Mutual Information (MI-FS). This experiment also compared DASC-FS with the existing approaches.
2. In the second experiment, the performance of the ASCA was evaluated against that of the original SCA. The results of ASCA were compared with those of the original SCA in terms of accuracy and convergence rate to demonstrate the improvement provided by ASCA over SCA.
3. The third experiment evaluated the integration of D-LDA into DASC-FS, comparing the proposed D-LDA with its counterparts, including FDA, WDA, LDA, PCA, RSLDA, K-LDA, and DeepLDA.
4. The fourth experiment aimed to statistically verify the obtained results. The Wilcoxon signed-rank test was used to determine whether there was a significant difference between the proposed method and its counterparts.
5. The final experiment analysed the selected features generated by the DASC-FS method for different datasets based on their importance. In this experiment, the features that were consistently selected across different datasets, as well as those that were only selected in certain datasets were examined. Determining the importance of features based on different datasets helps identify the subsets of features that are consistently selected and thus have a high impact on software quality.

The experiments were conducted on a machine equipped with an NVIDIA 3090 GPU, Intel Core i7 processor, and 8 GB of RAM, using Python 3.8, TensorFlow 2.5, and scikit-learn 0.24 for the implementation. For the GA, SCA-FS, and DASC-FS methods, the same parameter values were used to establish a common base. For example, the number of iterations was set to 40, and the size of the population was set to 20 based on our simple tuning (Section 4.3); the default values were used for GA and mutation, setting the crossover probability to 0.9 and mutation rate to 0.1, and the tournament selection size was 3; Abbas, Ahmad, & Jabeen, 2015). Regarding classification model parameters, the grid search technique was employed to explore different combinations of hyperparameters. After tuning, the parameters for each classifier were set as follows: KNN with $k = 5$, RF with 100 trees, LR with a penalty parameter of $C = 1.0$, GB with a learning rate of 0.1 and 100 trees, and GNB with a var_smoothing parameter of 1e-9.

In the following sections, detailed discussions accompany the results of each experiment, providing insights into the findings and offering a comprehensive understanding of the performance and contributions of the proposed methods.

4.1. Datasets

The proposed method was evaluated on three datasets widely used in SDP: JM1, KC1, and PC1. These datasets were created by researchers at NASA's Software Engineering Laboratory and have been used to evaluate SDP models. Each instance in the dataset represents a different project or module described using several features. In general, the features of these datasets are classified into McCabe metrics such as cyclomatic complexity, design complexity, and lines of code; Halstead measures such as the number of unique operators, number of unique operands, total occurrences of operators, and total occurrences of operands; and lines of code measures such as number of lines of code, lines of comments, and blank lines, to name a few. In addition, the class feature indicated whether defects were found in the module.

The JM1 dataset contains 10,885 records, each representing a single software module. Each instance contains 21 features, including size (measured in lines of code), number of defects found in the module during testing, and various attributes related to the design and implementation. The distribution of defect and non-defect classes range from 8,779 (81.05 %) to 2,106 (19.33 %) modules.

The KC1 dataset consists of the data collected from several software projects and is made available through the tera-PROMISE repository of software engineering databases. The dataset contains a total of 3,186 instances distributed to 1,783 (89.77 %) modules with 'False' labels and 326 (10.23 %) with 'True' labels.

The PC1 dataset contains information on 1,109 software modules. Each observation in the dataset comprises 22 features and class variables. These features describe various characteristics of software modules, such as the number of lines of code, unique operators, and modules. The class variable is a binary label indicating whether the module contains defects, with a value of 'False' for 6.94 % of the samples and a value of 'True' for 93.05 % of the samples.

4.2. Evaluating the DASC-FS method against existing feature selection methods

In this experiment, the proposed DASC-FS method was compared with existing methods, to assess the quality of the solution as a measure of how well it can predict defective software modules. First, the proposed method was evaluated using existing methods implemented in online libraries. This section reports on feature selection methods that were implemented using existing machine learning libraries, such as the scikit-learn library. The second part compares the proposed work with existing published works.

4.2.1. Comparison with existing implemented feature selection techniques

In this section, the performance of the DASC-FS is compared with that of well-established feature selection methods implemented in existing machine learning libraries. The existing methods involve different methods from different feature selection methods, including

RFE-FS as a wrapper method, MI-FS as a filter method, SCA-FS and GA-FS as optimization wrapper methods, and EM-FS based on supported classifiers such as RF, LR, and GB.

SCA-FS refers to the FS method implemented based on the standard SCA. It aims to select the most suitable features from the dataset by optimising the objective function. Each candidate solution presents a combination of features, and the suitability of each combination is evaluated based on prediction accuracy. Similarly, GA-FS is implemented based on GA which is inspired by natural selection. In GA-FS, the candidate combination of features evolves over generations through the operations of selection, crossover, and mutation.

RFE-FS is an FS method that recursively removes less important features from the dataset until the desired number of features is reached. Typically, this works by training a machine-learning model on a dataset, and the different features are ranked based on their importance or contribution to the performance of the model. EM-FS refers to FS methods embedded within the training process of a machine-learning model. These methods learn the feature importance during the model training process; however, they can be computationally expensive. MI-FS is an FS method that measures the statistical dependence between two variables. In MI-FS, different features are evaluated based on the mutual information they share with the target variable. Features with high levels of mutual information with the target variable are considered important and are therefore selected for inclusion in the final feature combination.

The proposed method was evaluated primarily on test accuracy to align with the aim of the research. However, other metrics such as precision, recall, and F1 scores are also reported. Comparisons were performed on several datasets using different classifiers. The comparisons are presented in Tables 2 to 4 and Figs. 3 to 5.

Table 2 and Fig. 3 present the results for the JM1 dataset. In general, the results showed that DASC-FS outperforms GA-FS and MI-FS in terms of test accuracy and surpasses ROC_AUC and F1, in terms of precision. This method also achieved competitive results compared with other methods, such as SCA-FS, RFE-FS, and EM-FS. On all classifiers, DASC-FS achieved the highest accuracy scores except for the LR and GNB classifiers, for which GA-FS achieved the highest score. In terms of the selected features, DASC-FS selected fewer features than GA-FS and SCA-FS in most cases, but more features than RFE-FS and EM-FS, as their selection was based on statistical scores.

For the RF classifier, DASC-FS obtained the highest accuracy score of 0.844432, followed by SCA-FS, with an accuracy score of 0.841561. The DASC-FS also obtained the highest F1 score of 0.362353 and a highest precision score of 0.587786, indicating that it can achieve a good balance between precision and recall. When using the LR classifier, the DASC-FS obtained an accuracy score of 0.835821, similar to that of the GA-FS. However, DASC-FS obtained the highest precision score of 0.54717, indicating that it can better identify true-positive samples. By contrast, RFE-FS and MI-FS had the highest F1 scores and precision, respectively, but low accuracy and recall.

For the GB classifier, DASC-FS obtained the highest accuracy score of 0.839265 and a highest precision score of 0.535714, indicating its capability to accurately identify true-positive samples. The DASC-FS also had the lowest F1 (0.096774) and recall scores (0.053191), demonstrating some difficulties in achieving a good balance between precision and recall. For the GNB classifier, DASC-FS obtained an accuracy score of 0.832951, which was lower than that of GA-FS and higher than that of SCA-FS. The DASC-FS outperformed the other methods with an F1 score of 0.441636 and a precision score of 0.72028, indicating that it can achieve a good balance between identifying true

Table 2
Comparison of DASC-FS with existing methods on the JM1 dataset.

	ROC_AUC	ACCURACY	F1	PRECISION	RECALL	SELECTED_FEATURES	COUNT
Classifier	Random Forest (RF)						
DASC-FS	0.612306	**0.844432**	0.362353	0.587786	0.261905	[1 1 0 1 1 0 1 1 0 1 0 1 1 0 0 1 1 0 1 0 1]	13
SCA-FS	0.605996	0.841561	0.349057	0.596774	0.246667	[0 1 1 0 1 1 1 0 0 1 1 1 0 1 1 1 0 0 1 1 1]	14
GA-FS	0.629802	0.840413	0.405983	0.641892	0.296875	[1 1 0 1 0 1 0 0 0 0 1 1 1 0 1 1 1 0 0 1 0]	15
RFE-FS	0.63788	0.781444	0.231628	0.372664	0.170465	[1 0 0 0 1 1 0 1 1 1 0 1 1 0 0 0 0 0 1 0 1]	10
EM-FS	0.649766	0.777493	0.219058	0.351322	0.163343	[1 0 0 1 0 1 0 1 1 1 0 1 1 0 1 0 0 0 0 0 1]	10
MI-FS	0.653473	0.778963	0.234592	0.36009	0.176163	[1 1 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 1 1 1 1]	10
Classifier	Logistic Regression (LR)						
DASC-FS	0.541558	0.835821	0.168605	0.54717	0.099656	[1 0 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 1 1 1 1]	10
SCA-FS	0.557702	**0.837543**	0.220386	0.555556	0.137457	[1 1 1 1 0 0 0 0 1 0 1 0 1 1 1 1 1 1 0 0 0]	12
GA-FS	0.521996	0.835821	0.100629	0.470588	0.056338	[1 0 0 1 1 1 0 1 1 0 0 1 1 0 0 0 0 1 0 1 0]	6
RFE-FS	0.670824	0.803676	0.136139	0.48638	0.079772	[0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 1 1 1 1 1 1]	10
EM-FS	0.643783	0.803951	0.092264	0.570066	0.051757	[0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 1 1 0 1 0]	6
MI-FS	0.68865	0.797154	0.179759	0.516805	0.117759	[1 1 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 1 1 1 1]	10
Classifier	Gradient Boosting (GB)						
DASC-FS	0.522144	**0.839265**	0.096774	0.535714	0.053191	[1 0 1 1 0 1 0 0 0 0 1 0 0 0 1 1 1 1 0 1 0]	10
SCA-FS	0.569965	0.835821	0.251309	0.761905	0.15047	[1 0 1 1 0 0 1 1 0 1 1 1 1 0 0 1 0 0 0 1 0]	11
GA-FS	0.553204	0.838117	0.20339	0.62069	0.121622	[1 1 1 1 1 0 1 0 0 0 0 1 0 0 0 1 1 1 0 1 0]	11
RFE-FS	0.690761	0.792009	0.142471	0.525145	0.098291	[1 1 0 1 0 0 0 1 0 0 0 0 1 0 1 0 1 1 1 1 0]	10
EM-FS	0.686704	0.786773	0.124361	0.588535	0.094017	[1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0]	3
MI-FS	0.680021	0.791182	0.140773	0.485542	0.097341	[1 1 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 1 1 1 1]	10
Classifier	Gaussian Naive Bayes (GNB)						
DASC-FS	0.56924	0.832951	0.255754	0.561798	0.165563	[0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 0 1 1 0 1]	16
SCA-FS	0.596189	0.830654	0.328018	0.5625	0.231511	[1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 1 0]	11
GA-FS	0.572969	**0.835247**	0.265985	0.547368	0.175676	[0 1 0 1 1 0 1 1 1 1 1 0 1 1 1 0 1 0 0 0 1]	13
MI-FS	0.663984	0.802023	0.307621	0.455577	0.235043	[1 1 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 1 1 1 1]	10
Classifier	K-Nearest Neighbours (KNN)						
DASC-FS	0.624435	**0.812285**	0.388785	0.472727	0.330159	[1 0 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 1 0 0 1]	14
SCA-FS	0.626059	0.811137	0.38961	0.458515	0.33871	[1 1 1 1 0 1 0 0 1 1 1 0 1 0 1 1 1 0 1 0 0 0 0 1]	11
GA-FS	0.609482	0.808266	0.357692	0.438679	0.301948	[0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 0 0 0]	9
MI-FS	0.589462	0.750299	0.262532	0.306646	0.22982	[1 1 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 1 1 1 1]	10

Table 3
Comparison of DASC-FS with the existing method on the KC1 dataset.

	ROC_AUC	ACCURACY	F1	PRECISION	RECALL	Selected features	Count
Classifier	Random Forest (RF)						
DASC-FS	0.732687	**0.914201**	0.613333	0.851852	0.479167	[1 1 1 1 0 1 1 1 1 0 0 0 0 1 0 1 0 0 0 0 0]	10
SCA-FS	0.687505	0.905325	0.515152	0.73913	0.395349	[0 1 1 0 0 0 1 0 0 0 1 0 1 1 1 0 1 0 1 0 1]	10
GA-FS	0.650273	0.905325	0.407407	0.52381	0.333333	[0 1 1 1 0 1 1 1 0 0 1 1 0 1 0 0 0 1 0 1 1]	13
RFE-FS	0.733014	0.808923	0.28802	0.353507	0.254601	[1 0 0 0 1 1 0 1 1 1 0 1 1 0 0 0 0 0 1 1 0]	10
EM-FS	0.728264	0.80892	0.270006	0.334688	0.233129	[1 0 0 0 1 1 0 1 1 1 0 1 1 0 0 0 0 1 1 1 0]	11
MI-FS	0.725701	0.815562	0.288867	0.37729	0.245399	[1 0 0 0 1 1 0 1 1 0 0 0 1 0 0 0 1 1 1 1 0]	10
Classifier	Logistic regression (LR)						
DASC-FS	0.669031	**0.91716**	0.481481	0.764706	0.351351	[0 0 1 0 0 0 1 1 0 0 0 0 0 1 1 1 1 1 1 1 0 1]	11
SCA-FS	0.645592	0.908284	0.415094	0.611111	0.314286	[1 0 0 0 1 1 1 1 0 1 0 1 1 1 1 0 1 1 0 1 0]	13
GA-FS	0.654111	0.905325	0.448276	0.722222	0.325	[1 0 0 0 1 1 0 0 1 0 1 0 1 0 0 0 1 1 0 1 1]	10
RFE-FS	0.78691	0.853964	0.313124	0.589616	0.214724	[0 1 1 1 0 0 1 1 0 0 0 0 0 1 1 1 0 0 0 1 1]	10
EM-FS	0.778919	0.848746	0.269565	0.545624	0.180982	[0 1 1 1 0 0 1 0 1 0 0 0 1 0 0 0 1 0 0 1 1]	9
MI-FS	0.612558	0.843055	0.253339	0.483333	0.174847	[1 0 0 0 1 1 0 1 1 0 0 0 1 0 0 0 1 1 1 1 0]	10
Classifier	Gradient Boosting (GB)						
DASC-FS	0.693935	**0.911243**	0.53125	0.772727	0.404762	[0 1 1 0 0 0 0 0 1 1 0 1 1 0 0 1 1 1 0 0 1]	10
SCA-FS	0.566133	0.902367	0.232558	0.714286	0.138889	[1 1 0 0 1 1 1 0 0 1 1 1 1 0 0 1 1 0 1 1 1]	14
GA-FS	0.619966	0.902367	0.377358	0.769231	0.25	[1 1 0 0 0 1 1 0 0 0 0 0 0 1 1 1 0 1 0 1 0 0]	9
RFE-FS	0.75239	0.849219	0.241055	0.539944	0.156442	[0 1 1 0 0 1 0 1 0 0 0 0 1 1 0 1 0 1 1 1 0]	10
EM-FS	0.761844	0.846373	0.263152	0.508065	0.177914	[0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0]	3
MI-FS	0.763828	0.848746	0.184311	0.557875	0.110429	[1 0 0 0 1 1 0 1 1 0 0 0 1 0 0 0 1 1 1 1 0]	10
Classifier	Gaussian Naive Bayes (GNB)						
DASC-FS	0.741069	**0.893491**	0.571429	0.615385	0.533333	[1 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0 1 0 1 1 0]	11
SCA-FS	0.744799	**0.893491**	0.55	0.55	0.55	[1 0 0 0 0 1 0 1 0 1 1 1 1 0 1 0 1 1 1 0 0]	11
GA-FS	0.673236	0.887574	0.457143	0.551724	0.390244	[1 1 1 0 0 0 1 0 0 1 0 0 1 0 1 1 0 1 1 0 0]	8
MI-FS	0.796891	0.820769	0.410731	0.417138	0.404908	[1 0 0 0 1 1 0 1 1 0 0 0 1 0 0 0 1 1 1 1 0]	10
Classifier	K-Nearest Neighbours (KNN)						
DASC-FS	0.686734	**0.899408**	0.484848	0.592593	0.410256	[0 1 1 1 0 1 0 1 1 0 1 0 1 0 0 0 0 1 1 1 1]	12
SCA-FS	0.635452	0.887574	0.387097	0.521739	0.307692	[1 1 0 1 1 1 0 0 0 0 1 1 0 1 0 0 1 1 0 1 1]	12
GA-FS	0.649944	0.893491	0.419355	0.565217	0.333333	[1 0 0 1 1 1 0 1 1 1 0 0 0 0 1 1 1 1 0 1 0]	8
MI-FS	0.625924	0.813659	0.286194	0.352453	0.242331	[1 0 0 0 1 1 0 1 1 0 0 0 1 0 0 0 1 1 1 1 0]	10

positives and minimizing false positives. For the KNN classifier, DASC-FS had the highest score of 0.814129, which was higher than those of SCA-FS and GA-FS. The DASC-FS also obtained the highest F1 score of 0.302545 and a highest precision score of 0.525316, indicating that it can achieve a good balance between precision and recall. However, DASC-FS had the lowest recall score of 0.209537, indicating that with KNN, it was difficult to identify true-positive samples.

The selected features for each classifier are also shown in the table as a binary vector, where a value of 1 indicates that a feature is selected and a value of 0 indicates otherwise. The number of selected features for all methods ranged between 3 and 16 for the different classifiers, whereas the range of selected features for DASC-FS was between 10 and 16. The number of features selected by DASC-FS was lower than those of SCA-FS and GA-FS for all classifiers, except for GNB and KNN. The EM-FS method selected the smallest number of features, ranging from 3 to 10, whereas RFE-FS and MI-FS selected the same number of features for all classifiers (10). Notably, the objective of DASC-FS was to achieve the highest test accuracy; therefore, it can select more features than other methods with different optimization objectives.

Overall, the results of the JM1 dataset suggest that DASC-FS can produce competitive results, particularly for classifiers such as RF, LR, and GB, especially if the focus is on test accuracy.

Table 3 and Fig. 4 present the results of comparing DASC-FS with other FS methods on the KC1 dataset. Overall, DASC-FS achieved the highest test accuracy among all methods for all classifiers. The proposed method consistently outperformed SCA-FS and GA-FS in terms of accuracy, F1 score, precision, and recall for all classifiers, whereas DASC-FS achieved the highest test accuracy.

For the RF classifier, DASC-FS achieved the highest accuracy of 0.914201 and a highest F1 score of 0.613333, indicating a fine balance between precision and recall. The DASC-FS ranked second in terms of the ROC_AUC score, indicating that it is the most efficient method if the focus is on the ROC_AUC score. For the LR classifier, DASC-FS achieved the highest accuracy score of 0.91716 and a highest F1 score of 0.481481, indicating a good balance between precision and recall. Similarly, DASC-FS achieved the highest accuracies of 0.911243, 0.863242, and 0.863242 using the GB, GNB, and KNN classifiers, respectively. It also had the highest F1 score when all three classifiers were used, indicating a good balance between precision and recall. However, the ROC_AUC scores were not as high as those of the other methods although DASC-FS still achieved competitive results.

Regarding the number of selected features, for the RF classifier, DASC-FS selected 10 features, which is the same as those of SCA-FS, RFE-FS, and MI-FS, but is fewer than those of GA-FS and EM-FS. Similarly, for the LR classifier, DASC-FS selected 11 features, which is fewer than those of SCA-FS and greater than those of GA-FS, RFE-FS, EM-FS, and MI-FS. For the GB classifier, DASC-FS selected 10 features, which is the same as those of RFE-FS and MI-FS but fewer than that of SCA-FS, which selected 14 features. For the GNB and KNN classifiers, DASC-FS performed the same as SCA-FS but selected more features than GA-FS and MI-FS.

Overall, DASC-FS emerges as the best method for the KC1 dataset, especially when using the RF classifier. The RF classifier achieved the highest performance scores (accuracy, F1 score, precision, and recall) for all FS methods, including DASC-FS, making it the most suitable classifier for the KC1 dataset.

Table 4 and Fig. 5 show the results on the PC1 dataset, with DASC-FS demonstrating competitive performance compared with the other methods. Specifically, although DASC-FS outperformed or performed similarly to other methods in terms of ROC_AUC, accuracy, and

Table 4
Comparison of DASC-FS with existing methods on the PC1 dataset.

	ROC_AUC	ACCURACY	F1	PRECISION	RECALL	Selected features	Count
Classifier	Random Forest (RF)						
DASC-FS	0.6700255	**0.9492795**	0.4608187	0.7619047	0.3484848	[1 1 1 1 0 1 0 1 1 1 0 1 1 1 1 0 0 0 1 0 1]	14
SCA-FS	0.6785104	**0.9492795**	0.4712435	0.7261904	0.3666666	[0 1 0 0 0 0 0 0 1 0 0 1 0 0 1 1 1 1 0 1 0]	8
GA-FS	0.6686655	0.9481368	0.4602287	0.7611111	0.3469696	[0 1 0 1 1 0 0 1 1 1 1 0 0 1 0 1 1 1 0 1 0]	9
RFE-FS	0.7386741	0.9296630	0.2680733	0.4568181	0.193657	[1 0 0 0 0 1 0 0 1 1 0 1 1 1 1 1 0 1 0 0 0]	10
EM-FS	0.702141	0.9278612	0.239527	0.4318181	0.1680161	[1 0 0 0 1 1 0 0 1 1 0 1 1 1 1 1 0 1 0 1 0]	12
MI-FS	0.7504848	0.9233551	0.2387301	0.375	0.1808367	[1 0 0 0 1 1 0 0 0 0 1 0 1 1 1 1 0 1 1 0 0]	10
Classifier	Logistic regression (LR)						
DASC-FS	0.5922873	**0.9425188**	0.2962292	0.82	0.1893939	[1 0 1 1 1 1 1 0 0 1 1 1 1 1 1 0 0 1 0 1 1 1]	15
SCA-FS	0.5909273	0.9413889	0.2817496	0.78	0.1878787	[1 0 1 1 1 1 0 0 1 1 0 0 0 1 0 0 1 1 0 1 0]	11
GA-FS	0.5915297	0.9425125	0.2850829	0.8	0.1878787	[1 1 0 0 1 1 1 0 1 0 0 0 0 0 0 0 0 1 1 0 0]	12
RFE-FS	0.7685893	0.9215614	0.210942	0.5047619	0.1801619	[1 1 0 0 0 0 1 0 1 0 0 0 1 0 1 1 1 1 0 0 1]	10
EM-FS	0.7814922	0.9188554	0.1652353	0.3229166	0.1413630	[0 1 0 1 1 0 0 1 1 0 0 0 0 1 0 1 1 0 0 0 1]	9
MI-FS	0.5449312	0.9017204	0.1827956	0.1574074	0.2179487	[1 0 0 0 1 1 0 0 0 0 1 0 1 1 1 1 0 1 1 0 0]	10
Classifier	Gradient Boosting (GB)						
DASC-FS	0.5728933	**0.9380054**	0.2168016	0.4428571	0.1530303	[0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 0 1 1 1 0 1]	8
SCA-FS	0.5728933	**0.9380054**	0.2168016	0.4428571	0.1530303	[0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 0 1 1 1 0 1]	8
GA-FS	0.5728933	**0.9380054**	0.2168016	0.4428571	0.1530303	[0 1 0 1 1 0 0 0 0 1 0 0 0 1 1 1 0 1 0 0 1]	8
RFE-FS	0.7022205	0.9170536	0.1889472	0.3291666	0.1798245	[0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1]	10
EM-FS	0.6894497	0.9242609	0.1430343	0.3416666	0.1029014	[0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0]	2
MI-FS	0.7203456	0.9197563	0.1865524	0.3380952	0.1670040	[1 0 0 0 1 1 0 0 0 0 1 0 1 1 1 1 0 1 1 0 0]	10
Classifier	Gaussian Naive Bayes (GNB)						
DASC-FS	0.6329481	**0.9289786**	0.3460564	0.4727272	0.2924242	[0 0 1 0 0 1 1 0 0 1 0 1 0 1 1 0 0 1 0 0 0]	8
SCA-FS	0.6252172	**0.9289786**	0.3334248	0.4703463	0.2757575	[0 0 0 1 0 1 1 0 0 1 1 0 0 1 1 0 0 0 0 0 0]	7
GA-FS	0.6041821	0.91995175	0.2762322	0.3961538	0.2409090	[1 0 1 0 0 1 1 1 0 1 0 1 1 0 1 1 0 1 0 1 0]	9
MI-FS	0.7019054	0.86298825	0.2536529	0.2525488	0.348515	[1 0 0 0 1 1 0 0 0 0 1 0 1 1 1 1 0 1 1 0 0]	10
Classifier	K-Nearest Neighbours (KNN)						
DASC-FS	0.6203468	0.9357392	0.3421381	0.5388888	0.2575757	[1 0 1 1 0 0 1 1 1 0 1 0 0 1 1 1 0 0 0 1 0]	11
SCA-FS	0.6407904	**0.9436361**	0.3848077	0.68	0.2924242	[0 1 1 0 0 0 1 1 1 0 0 0 0 1 1 0 0 1 0 0 1]	9
GA-FS	0.5947480	0.9312384	0.2828041	0.5412121	0.2075757	[1 0 1 1 1 1 1 0 0 1 1 1 0 0 0 0 0 0 1 1 0]	11
MI-FS	0.5914206	0.9134435	0.0784313	0.1538461	0.0526315	[1 0 0 0 1 1 0 0 0 0 1 0 1 1 1 1 0 1 1 0 0]	10

precision, the recall and F1 scores were lower than those of the other methods, and the RFE-FS and MI-FS performances were poor compared with the other methods in most cases.

For the RF classifier, DASC-FS obtained the highest test accuracy (0.9492795) compared with SCA-FS. The ROC_AUC score of DASC-FS was 0.6700255, indicating that the classifier was quite good at distinguishing between the positive and negative classes. DASC-FS achieved the highest precision (0.7619047), indicating that the classifier predicted the positive class correctly most of the time. For the LR classifier, DASC-FS achieved the second-highest accuracy (0.9413889), with a slightly better ROC_AUC (0.592) than SCA-FS and GA-FS. However, the F1 score (0.2962) was the lowest, indicating a lower ability to balance precision and recall.

For the GB classifier, DASC-FS, SCA-FS, and GA-FS produced almost identical results. They achieved the highest accuracy (0.9380054) and ROC_AUC score (0.4428571), outperforming the other methods. Even though their F1 score (0.2168016) was the highest, it is still low indicating a lower ability to balance precision and recall despite outperforming the other methods. For the GNB classifier, GA-FS achieved the highest accuracy (0.9289786) and ROC_AUC score (0.6329481). Its F1 score (0.3460564) is the best compared to those of RFE-FS and MI-FS; however, this low F1 score indicates a lower ability to balance precision and recall. For the KNN classifier, DASC-FS achieved the second-highest accuracy (93.52 %) among all methods, with a slightly better ROC_AUC (0.750) than those of the other methods. However, the F1 score (0.3421381) and precision (0.5388888) were the lowest compared to that of SCA-FS.

Regarding the number of selected features, for the GB classifier, DASC-FS had the lowest number of selected features with 8, tied with SCA-FS. For the GNB and KNN classifiers, DASC-FS selected 8 and 11 features, respectively. However, DASC-FS had the highest number of selected features for the RF and LR models, with 14 and 15 selected features, respectively.

Overall, the results of the PC1 dataset showed that DASC-FS outperformed the other methods in terms of accuracy in three out of five classifiers. The ROC_AUC was slightly better or comparable to those of the other methods. However, in most cases, it exhibited a lower ability to balance precision and recall, selecting a relatively large number of features.

To summarise, the experiments showed that the results of DASC-FS differ depending on the dataset and classifier. However, DASC-FS outperformed the other methods in terms of accuracy and was found to be the most suitable when working with the RF classifier, obtaining the best results on the three datasets.

4.2.2. *Comparison with existing research*

This section places the proposed strategy in the context of existing research, comparing DASC-FS with those reported in relevant published works.

Table 5 provides a comparison of DASC-FS with other existing software defect prediction methods. The results of the existing methods were collected from different online publications, including (Alkhasawneh, 2022; Das et al., 2023; Goyal, 2022b) (Elish & Elish, 2008; Iqbal & Aftab, 2020; Iqbal et al., 2019; Kakkar & Jain, 2016). Three datasets (JM1, KC1, and PC1) were compared using the corresponding classifiers for each method. The results demonstrate that most existing methods are accurate based on the accuracy measurement metric. DASC-FS consistently demonstrated strong predictive capabilities compared to the other methods, achieving 84.04 %, 92.01 %, and 98.31 % on the respective datasets, outperforming all the other methods, except for the JM1

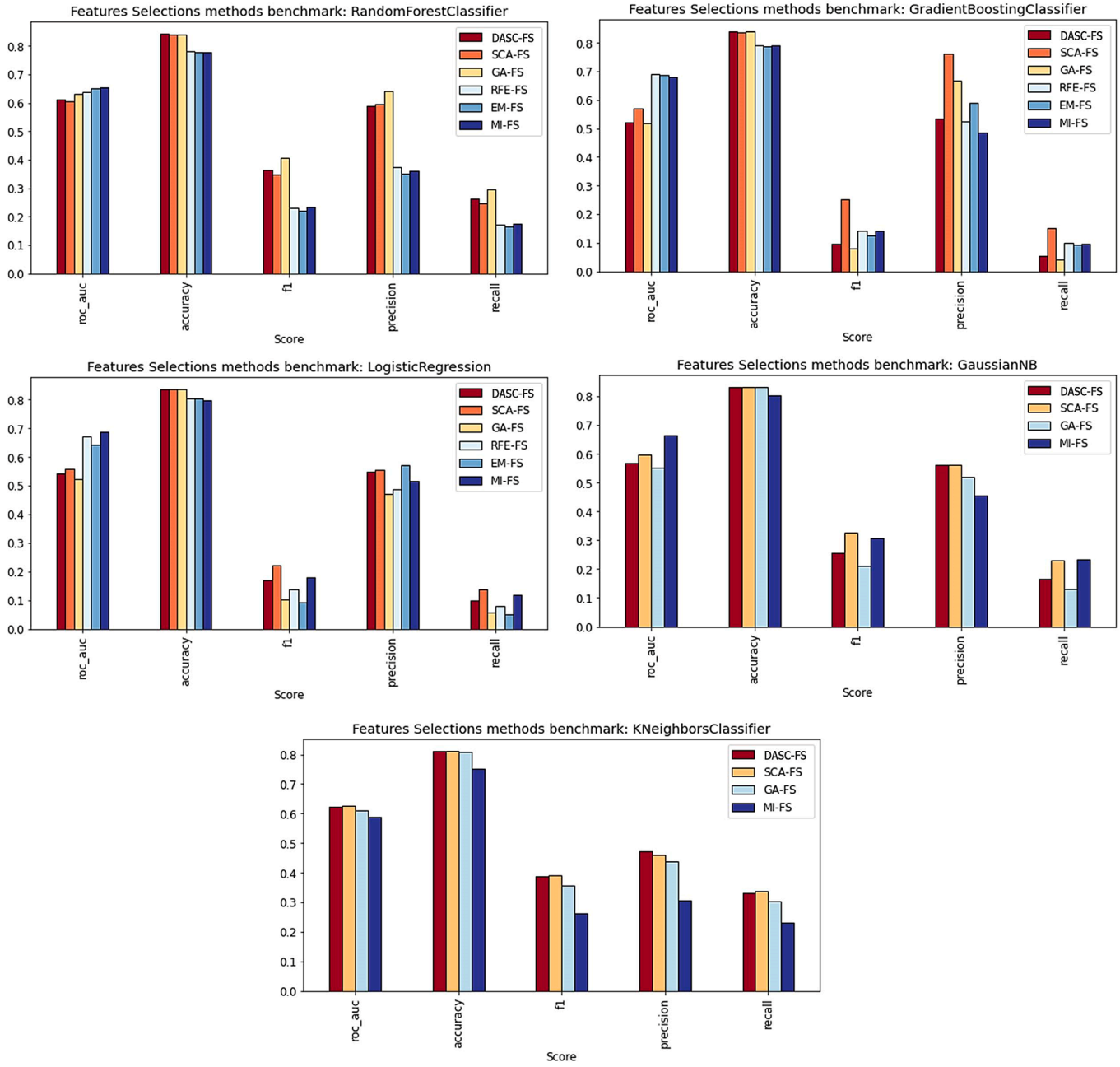


Fig. 3. Comparison of DASC-FS with existing methods on the JM1 dataset.

dataset where FS-EPwMD excelled, achieving the highest accuracy of 85.55 %. The results obtained on the KC1 dataset showed that DASC-FS achieved the highest accuracy of 92.01 % using the LR classifier, whereas FSPSO obtained the lowest accuracy of 73.21 % using the DT classifier. The results obtained for the PC1 dataset indicate that RBF achieved the highest accuracy of 98.99 %, whereas DASC-FS ranked as the second-best method with an accuracy of 98.31 %.

Delving deeper into the results, the DASC-FS outperformed most of the other methods such as FSACO, FSDE, FSGA, FSGJO, FSPSO, FS-GA, FS-PSO, FS-EPwMD, FS-12, FS-13, FS-14, and FS-15. DASC-FS consistently demonstrates superior performance across the three datasets using different classifiers, showcasing robustness and adaptability. This consistency across different datasets and classifiers confirms the effectiveness of the proposed method owing to the synergistic integration of two key components: ASCA for feature selection and D-LDA for enhancing discriminative power. This synergistic combination facilitates a comprehensive optimization approach considering enhanced accuracy and class separation, thus outperforming existing methods in which such synergistic integration is absent. However, DASC-FS was outperformed by RBF and FS-EPwMD (ANN) in some cases, whereas FS13 (MLP), FS14 (Boost-RF-FS), and FS14 (Bag-RF-FS) produced results similar to those of DASC-FS. This may be justified by the techniques used in these methods: FS13 (MLP) and FS-EPwMD (ANN) utilized neural networks capable of modelling complex and nonlinear relationships within the data; FS14 (Boost-RF-FS) and FS14 (Bag-RF-FS) employed ensemble techniques that combined multiple models.

4.3. Evaluating the convergence rate of ASCA against standard SCA

In this section, the performance of the proposed ASCA is compared with that of the standard SCA from the optimisation perspective. This

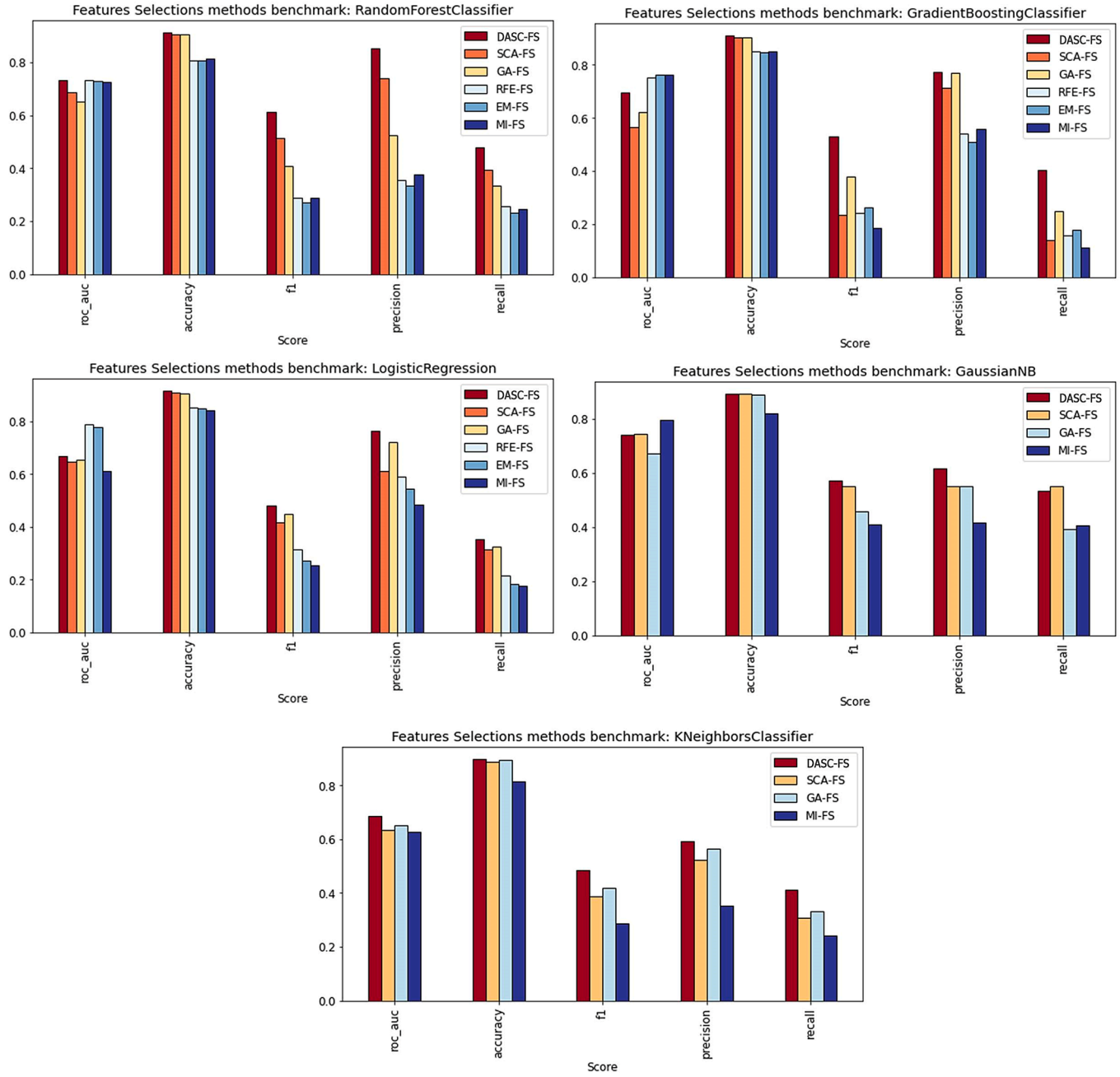


Fig. 4. Comparison of DASC-FS with existing methods on the KC1 dataset.

evaluation aims to analyse the convergence rate, which measures the speed at which the optimal solution can be obtained. To measure the convergence rate of the ASCA and SCA, we ran the algorithms for a number of iterations ranging from 0 to 40, for a population size of 20. This enabled us to measure the rate at which the algorithms converged to the optimal solution. The results of this experiment provided valuable insights into the performance of the algorithms in terms of convergence rate, helping us identify areas where the ASCA can be improved in the future.

The results are presented in Table 6 and Fig. 6. The table and figure present the optimal test accuracies of the three datasets (MJ1, KC1, and PC1) for the five classifiers. To acquire an immediate understanding of the performance variations across different iterations and classifiers, the results in Table 6 are visually represented using a graduated colour scale (Green − Yellow − Red). The shades in green indicate higher accuracy; yellow indicates moderate values; and red indicates lower accuracy.

In the results, we see that the performances of the ASCA and SCA are stable with increasing number of iterations. Moreover, ASCA's convergence to a stable high accuracy is faster than that of the SCA. This indicates the capability of ASCA to reach the optimal solution with fewer iterations.

Delving deeper into the MJ1 results, DASC-FS requires fewer iterations than SCA to reach a good or optimal solution using the RF, LR, and GB classifiers, whereas SCA-FS requires additional iterations. Moreover, the figures show that ASCA is more efficient in finding the optimal solution as the number of iterations increases. The KNN and GNB classifiers struggled to improve the solution. However, when the number of iterations increased ASCA succeeded in finding a better solution with no modifications to the ASCA. Regarding the KC1 dataset, ASCA is more consistent and displays faster convergence to higher accuracy values with all the classifiers, with the exception of GNB. In some iterations, the SCA is faster than ASCA; however, ASCA outperformed SCA in the final

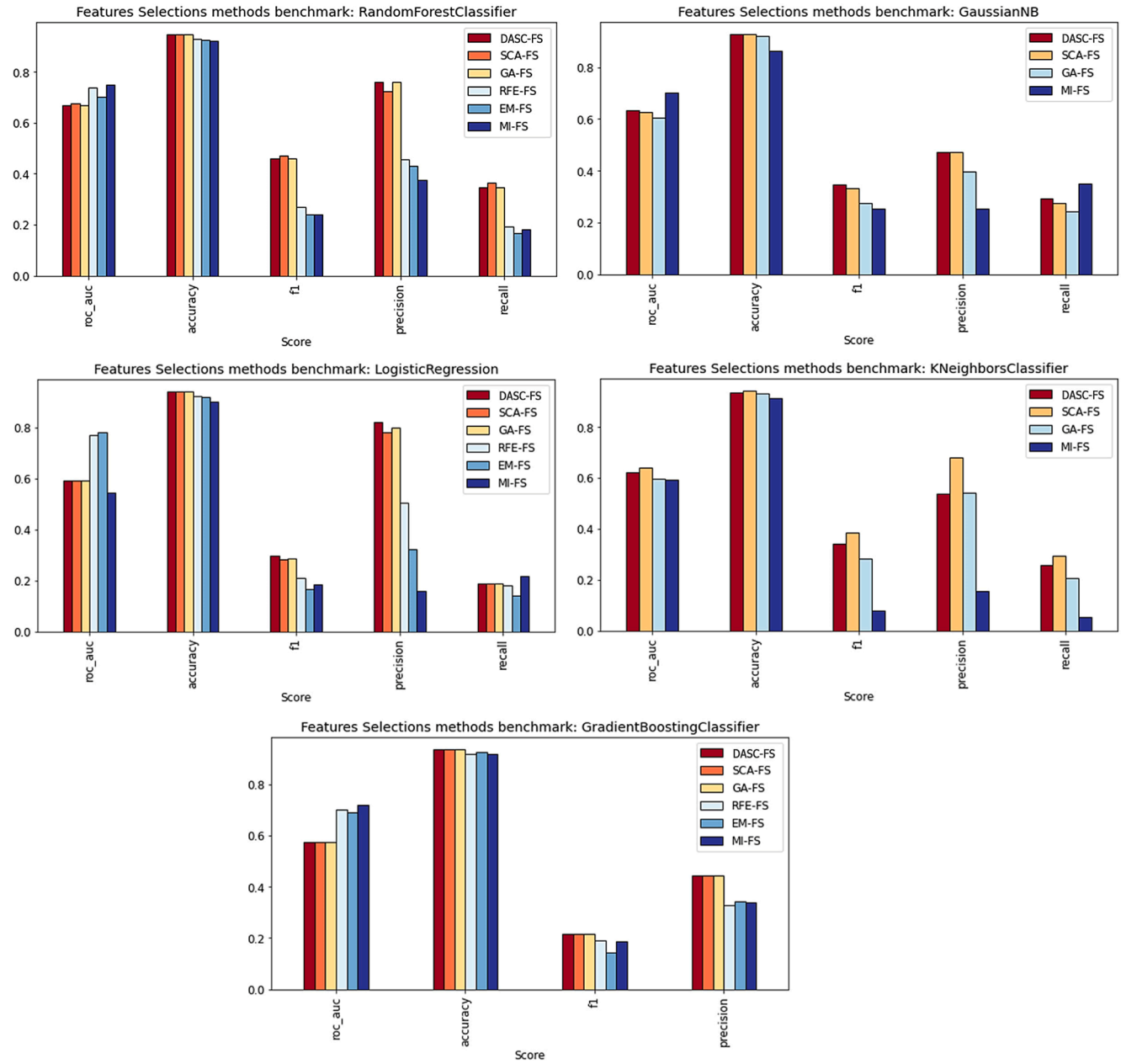


Fig. 5. Comparison of DASC-FS with the existing method on the PC1 dataset.

result. The same applies to the PC1 dataset, where the accuracy varies from one classifier to another. However, the convergence of ASCA is faster when using all classifiers.

In summary, ASCA converges to higher accuracy values faster and more consistently than SCA. The adaptation of multiple mutations into ASCA plays a crucial role in achieving more efficient and effective convergence, making it a promising choice for feature selection in a given context.

4.4. Evaluating the depth linear discrimination analysis

In this section, the integration of D-LDA into DASC-FS is evaluated to investigate the performance of the proposed DASC-FS with and without D-LDA. First, D-LDA was evaluated against existing DR, and subsequently, the performance of DASC-FS was evaluated with and without D-LDA.

4.4.1. Comparison D-LDA with other discrimination methods

This section assesses the proposed depth linear discrimination analysis method, namely D-LDA, comparing D-LDA with several other DR methods, including FDA, Wasserstein Discriminant Analysis (WDA), LDA, PCA, Robust Sparse LDA (RSLDA), Kernel LDA (KLDA), and Deep LDA (DeepLDA). All these methods can reduce the feature space; therefore, our evaluation focused on assessing the impact of dimensionality reduction on the accuracy and robustness of the methods. Table 7 presents the comparison results of D-LDA against existing methods on the software defect datasets KC1, PC1, and JM1 using the RF classifier.

When considering datasets without outliers, the tables show that D-LDA achieved both the highest and average accuracy among all the methods, on datasets MJ1 and PC1, whereas in the case of KC1, D-LDA performed well, achieving the second-highest accuracy and highest average accuracy. The results demonstrate the ability of D-LDA to effectively distinguish between classes and improve accuracy. Even in

Table 5
Comparison of DASC-FS with existing software defect prediction methods.

Dataset			JM1		KC1		PC1	
Source	Method	Classifier	Accuracy	Selected Features	Accuracy	Selected Features	Accuracy	Selected Features
This research	DASC-FS	RF	**0.8439**	9	**0.9142**	9.5	0.9775	9.3
	DASC-FS	LR	0.8404	10.13	**0.9201**	10.0	**0.9831**	11.6
	DASC-FS	GB	0.8428	9.6	**0.9142**	8.3	**0.9775**	10
	DASC-FS	GNB	0.8387	13.7	0.9112	11.4	0.9663	10.3
	DASC-FS	KNN	0.8180	11.6	0.9112	8.3	0.9831	9.4
(Das et al., 2023)	FSACO	DT	0.7962	3	7648	4.5	0.7321	8.3
	FSACO	KNN	0.7959	3	7759	4.5	0.7646	8.3
	FSACO	NB	0.7989	3	7747	4.5	0.7621	8.3
	FSACO	QDA	0.7978	3	7758	4.5	0.7692	8.3
(Das et al., 2023)	FSDE	DT	0.7663	10.3	763	9.4	0.7779	8
	FSDE	KNN	0.7339	10.3	7633	9.4	0.7805	8
	FSDE	NB	0.7962	10.3	7732	9.4	0.7763	8
	FSDE	QDA	0.7982	10.3	7739	9.4	0.7848	8
(Das et al., 2023)	FSGA	DT	0.7781	4.9	776	8.2	0.776	8.2
	FSGA	KNN	0.7863	4.9	7646	8.2	0.7646	8.2
	FSGA	NB	0.7984	4.9	7732	8.2	0.7732	8.2
	FSGA	QDA	0.7971	4.9	7801	8.2	0.7801	8.2
(Das et al., 2023)	FSGJO	DT	0.7816	6	7779	8	0.763	9.4
	FSGJO	KNN	0.796	6	7805	8	0.7633	9.4
	FSGJO	NB	0.7989	6	7763	8	0.7732	9.4
	FSGJO	QDA	0.7982	6	7848	8	0.7739	9.4
(Das et al., 2023)	FSPSO	DT	0.756	8.9	7321	8.3	0.7648	4.5
	FSPSO	KNN	0.7249	8.9	7646	8.3	0.7759	4.5
	FSPSO	NB	0.7915	8.9	7621	8.3	0.7747	4.5
	FSPSO	QDA	0.7904	8.9	7692	8.3	0.7758	4.5
(Alkhasawneh, 2022)	N/A	RBF	**0.8487**	N/A	0.8325	N/A	**0.9899**	N/A
(Goyal, 2022b)	FS-GA	Naive Bayes	0.6831	N/A	0.6513	N/A	0.6842	N/A
	FS-GA	KNN	0.5812	N/A	0.6012	N/A	0.5983	N/A
	FS-GA	DT	0.7231	N/A	0.7452	N/A	0.7152	N/A
	FS-GA	SVM	0.7231	N/A	0.7513	N/A	0.7034	N/A
	FS-GA	ANN	0.7803	N/A	0.7851	N/A	0.7251	N/A
(Goyal, 2022b)	FS-PSO	Naive Bayes	0.7034	N/A	0.7001	N/A	0.7102	N/A
	FS-PSO	KNN	0.6256	N/A	0.6091	N/A	0.6189	N/A
	FS-PSO	DT	0.7723	N/A	0.7671	N/A	0.7412	N/A
	FS-PSO	SVM	0.7721	N/A	0.76	N/A	0.752	N/A
	FS-PSO	ANN	0.791	N/A	0.7881	N/A	0.7712	N/A
(Goyal, 2022b)	FS-EPwMD	Naive Bayes	0.8133	N/A	0.8011	N/A	0.7941	N/A
	FS-EPwMD	KNN	0.6533	N/A	0.6241	N/A	0.6512	N/A
	FS-EPwMD	DT	0.8019	N/A	0.811	N/A	0.849	N/A
	FS-EPwMD	SVM	0.8243	N/A	0.8012	N/A	0.884	N/A
	FS-EPwMD	ANN	**0.8555**	N/A	0.8231	N/A	0.8994	N/A
(Kakkar & Jain, 2016)	FS-12	LWL	0.8096	N/A	0.8527	N/A	0.9108	N/A
	FS-12	J Star	0.8102	N/A	0.8682	N/A	0.8876	N/A
	FS-12	IBK	0.7526	N/A	0.8288	N/A	0.8643	N/A
	FS-12	RT	0.7468	N/A	0.8205	N/A	0.8682	N/A
	FS-12	RF	0.7971	N/A	0.8401	N/A	0.8992	N/A
(Iqbal & Aftab, 2020)	FS-13	MLP	0.8044	N/A	0.7765	N/A	0.9656	N/A
(Iqbal et al., 2019)	FS-14	Boost-RF-FS	0.8056	N/A	0.7851	N/A	0.9607	N/A
	FS-14	Bag-RF-FS	0.8061	N/A	0.7851	N/A	0.9607	N/A
(Elish & Elish, 2008)	FS-15	SVM	N/A	N/A	0.8459	N/A	0.9310	N/A
	FS-15	LR	N/A	N/A	0.8555	N/A	0.9319	N/A
	FS-15	KNN	N/A	N/A	0.8399	N/A	0.9182	N/A
	FS-15	MLP	N/A	N/A	0.8568	N/A	0.9359	N/A
	FS-15	RBF	N/A	N/A	0.8481	N/A	0.9284	N/A
	FS-15	BBN	N/A	N/A	0.7599	N/A	0.9044	N/A
	FS-15	NB	N/A	N/A	0.8286	N/A	0.8921	N/A
	FS-15	RF	N/A	N/A	0.8520	N/A	0.9366	N/A
	FS-15	DT	N/A	N/A	0.8456	N/A	0.9358	N/A

the presence of outliers (Table 8), D-LDA was superior in obtaining the best accuracy and average accuracy. D-LDA outperformed the other methods, including FDA, WDA, LDA, PCA, RSLDA, and DeepLDA, on the three datasets, except for the JM1 dataset. KLDA and RSLDA outperformed D-LDA in obtaining the best averages on the JM1 dataset. Although in the case of JM1, D-LDA was not the best, it still performed well in obtaining competitive results compared to the other methods.

To illustrate the ability of D-LDA to linearly separate classes and handle outliers, Figs. 7 and 8 present a comparison of D-LDA with existing methods using a simple dataset. These figures also show the accuracy of each method. Fig. 7 shows the results of applying the dimensionality reduction methods to normal data, whereas Fig. 8 shows the results after introducing outliers. The results show the superiority of the D-LDA method over the other methods in the presence and absence of outliers, except for the RSLDA method, which obtained results close to those of the D-LDA method. The results of KLDA and RSLDA were competitive and comparable to those of D-LDA.

4.4.2. Evaluating the integration of D-LDA into DASC-FS

The results in Table 9 present the maximum and average test scores obtained over 40 runs for each classifier. Overall, integrating D-LDA into DASC-FS improved the accuracy scores in most cases. The accuracy scores improved with the LR, GB, GNB, and KNN classifiers on the three datasets; however, with the RF classifier, the accuracies were slightly

Table 6
Convergence rate comparisons of ASCA, SCA, and GA on the MJ1, KC1 and PC1 datasets.

Iteration	MJ1										KC1										PC1									
	ASCA-RF	SCA-RF	ASCA-LR	SCA-LR	ASCA-GB	SCA-GB	ASCA-GNB	SCA-GNB	ASCA-KNN	SCA-KNN	ASCA-RF	SCA-RF	ASCA-LR	SCA-LR	ASCA-GB	SCA-GB	ASCA-GNB	SCA-GNB	ASCA-KNN	SCA-KNN	ASCA-RF	SCA-RF	ASCA-LR	SCA-LR	ASCA-GB	SCA-GB	ASCA-GNB	SCA-GNB	ASCA-KNN	SCA-KNN
0	0.8341	0.8307	0.8318	0.8318	0.8398	0.8341	0.8261	0.8226	0.8060	0.8014	0.905	0.893	0.893	0.893	0.896	0.896	0.879	0.879	0.879	0.879	0.955	0.955	0.961	0.961	0.961	0.961	0.933	0.933	0.961	0.961
1	0.8341	0.8307	0.8358	0.8318	0.8398	0.8341	0.8278	0.8301	0.8060	0.8014	0.905	0.893	0.905	0.896	0.899	0.896	0.891	0.891	0.879	0.879	0.955	0.955	0.961	0.972	0.966	0.961	0.933	0.933	0.966	0.961
2	0.8341	0.8307	0.8358	0.8330	0.8398	0.8341	0.8278	0.8301	0.8060	0.8031	0.905	0.893	0.905	0.896	0.899	0.896	0.891	0.891	0.879	0.879	0.966	0.966	0.966	0.972	0.972	0.961	0.933	0.933	0.966	0.961
3	0.8341	0.8307	0.8358	0.8330	0.8398	0.8341	0.8289	0.8301	0.8134	0.8031	0.905	0.893	0.905	0.896	0.899	0.896	0.891	0.891	0.879	0.879	0.966	0.966	0.978	0.972	0.972	0.961	0.933	0.933	0.966	0.961
4	0.8341	0.8307	0.8358	0.8330	0.8398	0.8341	0.8289	0.8301	0.8134	0.8031	0.905	0.893	0.905	0.902	0.899	0.902	0.891	0.891	0.888	0.879	0.966	0.966	0.978	0.972	0.972	0.961	0.938	0.933	0.966	0.961
5	0.8370	0.8307	0.8358	0.8330	0.8398	0.8341	0.8289	0.8301	0.8134	0.8100	0.905	0.893	0.908	0.902	0.899	0.902	0.891	0.891	0.888	0.888	0.966	0.966	0.978	0.972	0.972	0.961	0.938	0.933	0.966	0.961
6	0.8370	0.8404	0.8358	0.8330	0.8398	0.8358	0.8289	0.8301	0.8134	0.8100	0.905	0.893	0.911	0.902	0.899	0.902	0.891	0.891	0.888	0.888	0.966	0.966	0.978	0.978	0.972	0.966	0.938	0.933	0.966	0.972
7	0.8370	0.8404	0.8358	0.8330	0.8398	0.8358	0.8289	0.8301	0.8134	0.8100	0.905	0.899	0.914	0.902	0.899	0.902	0.891	0.891	0.888	0.888	0.966	0.966	0.978	0.978	0.972	0.966	0.938	0.966	0.966	0.972
8	0.8370	0.8404	0.8358	0.8330	0.8398	0.8358	0.8289	0.8301	0.8134	0.8100	0.905	0.899	0.914	0.902	0.899	0.902	0.891	0.891	0.888	0.888	0.966	0.966	0.978	0.978	0.972	0.966	0.938	0.966	0.966	0.972
9	0.8370	0.8404	0.8370	0.8330	0.8398	0.8358	0.8295	0.8301	0.8134	0.8100	0.905	0.899	0.914	0.902	0.908	0.902	0.891	0.891	0.888	0.888	0.966	0.966	0.978	0.978	0.972	0.966	0.938	0.966	0.966	0.972
10	0.8370	0.8404	0.8370	0.8330	0.8398	0.8364	0.8295	0.8335	0.8134	0.8100	0.905	0.899	0.914	0.902	0.908	0.902	0.891	0.891	0.888	0.888	0.966	0.966	0.978	0.978	0.972	0.966	0.938	0.966	0.966	0.972
11	0.8370	0.8404	0.8370	0.8330	0.8398	0.8364	0.8295	0.8335	0.8134	0.8100	0.905	0.899	0.914	0.905	0.908	0.902	0.893	0.891	0.888	0.888	0.966	0.966	0.978	0.978	0.972	0.966	0.938	0.966	0.972	0.972
12	0.8370	0.8404	0.8370	0.8330	0.8398	0.8364	0.8295	0.8335	0.8134	0.8100	0.905	0.899	0.914	0.905	0.908	0.902	0.893	0.891	0.888	0.888	0.972	0.972	0.978	0.978	0.972	0.966	0.938	0.966	0.972	0.972
13	0.8370	0.8404	0.8381	0.8330	0.8398	0.8364	0.8301	0.8335	0.8134	0.8123	0.905	0.902	0.914	0.905	0.908	0.902	0.893	0.891	0.888	0.888	0.972	0.972	0.978	0.978	0.972	0.966	0.938	0.966	0.978	0.972
14	0.8387	0.8404	0.8381	0.8330	0.8427	0.8364	0.8301	0.8335	0.8134	0.8123	0.905	0.902	0.914	0.905	0.908	0.902	0.893	0.891	0.888	0.888	0.972	0.972	0.978	0.978	0.972	0.966	0.938	0.966	0.978	0.972
15	0.8387	0.8404	0.8381	0.8364	0.8427	0.8375	0.8301	0.8335	0.8134	0.8123	0.905	0.902	0.914	0.905	0.908	0.902	0.893	0.896	0.888	0.888	0.972	0.972	0.978	0.978	0.972	0.966	0.938	0.966	0.978	0.972
16	0.8387	0.8404	0.8381	0.8364	0.8427	0.8375	0.8301	0.8335	0.8134	0.8123	0.905	0.902	0.914	0.905	0.908	0.902	0.893	0.896	0.888	0.888	0.972	0.972	0.978	0.978	0.978	0.966	0.938	0.966	0.978	0.972
17	0.8387	0.8404	0.8381	0.8364	0.8427	0.8375	0.8301	0.8335	0.8134	0.8123	0.905	0.902	0.914	0.905	0.908	0.902	0.896	0.896	0.888	0.888	0.972	0.972	0.978	0.978	0.978	0.966	0.938	0.966	0.978	0.972
18	0.8387	0.8404	0.8381	0.8364	0.8427	0.8375	0.8301	0.8335	0.8134	0.8134	0.908	0.902	0.914	0.905	0.908	0.902	0.896	0.896	0.888	0.888	0.972	0.972	0.978	0.978	0.978	0.966	0.949	0.966	0.978	0.972
19	0.8387	0.8404	0.8381	0.8364	0.8427	0.8375	0.8324	0.8335	0.8134	0.8134	0.908	0.902	0.914	0.905	0.908	0.911	0.896	0.896	0.888	0.888	0.972	0.972	0.978	0.978	0.978	0.966	0.949	0.966	0.978	0.972
20	0.8387	0.8404	0.8381	0.8364	0.8427	0.8375	0.8324	0.8335	0.8134	0.8134	0.908	0.902	0.914	0.905	0.914	0.911	0.896	0.896	0.888	0.888	0.972	0.972	0.978	0.978	0.978	0.966	0.949	0.966	0.978	0.972
21	0.8387	0.8404	0.8381	0.8364	0.8427	0.8427	0.8324	0.8370	0.8134	0.8134	0.908	0.902	0.914	0.905	0.914	0.917	0.896	0.902	0.888	0.888	0.972	0.972	0.978	0.978	0.978	0.966	0.949	0.966	0.978	0.972
22	0.8387	0.8404	0.8381	0.8364	0.8427	0.8427	0.8324	0.8370	0.8134	0.8134	0.908	0.902	0.920	0.905	0.914	0.917	0.896	0.902	0.888	0.888	0.972	0.972	0.978	0.978	0.978	0.966	0.949	0.966	0.978	0.972
23	0.8387	0.8404	0.8381	0.8364	0.8427	0.8427	0.8324	0.8370	0.8134	0.8134	0.908	0.908	0.920	0.905	0.914	0.917	0.896	0.902	0.888	0.888	0.972	0.972	0.978	0.978	0.978	0.966	0.949	0.966	0.978	0.972
24	0.8387	0.8404	0.8381	0.8364	0.8427	0.8427	0.8324	0.8370	0.8134	0.8134	0.908	0.908	0.920	0.905	0.914	0.917	0.896	0.902	0.891	0.888	0.972	0.972	0.978	0.978	0.978	0.966	0.949	0.966	0.978	0.972
25	0.8393	0.8404	0.8381	0.8370	0.8427	0.8427	0.8324	0.8370	0.8134	0.8134	0.908	0.908	0.920	0.905	0.914	0.917	0.896	0.902	0.891	0.888	0.972	0.972	0.978	0.978	0.978	0.978	0.949	0.966	0.978	0.972
26	0.8393	0.8404	0.8381	0.8370	0.8427	0.8427	0.8324	0.8370	0.8134	0.8134	0.908	0.908	0.920	0.905	0.914	0.917	0.896	0.902	0.891	0.888	0.972	0.972	0.978	0.978	0.978	0.978	0.949	0.966	0.978	0.972
27	0.8393	0.8404	0.8381	0.8370	0.8427	0.8427	0.8324	0.8370	0.8180	0.8134	0.908	0.908	0.920	0.905	0.914	0.917	0.896	0.902	0.891	0.888	0.972	0.972	0.978	0.978	0.978	0.978	0.949	0.966	0.978	0.983
28	0.8439	0.8404	0.8381	0.8370	0.8427	0.8427	0.8324	0.8370	0.8180	0.8134	0.908	0.908	0.920	0.905	0.914	0.917	0.896	0.902	0.896	0.888	0.972	0.972	0.978	0.978	0.978	0.978	0.949	0.966	0.978	0.983
29	0.8439	0.8404	0.8381	0.8370	0.8428	0.8427	0.8341	0.8370	0.8180	0.8134	0.908	0.908	0.920	0.905	0.914	0.917	0.896	0.902	0.896	0.888	0.972	0.972	0.983	0.978	0.978	0.978	0.949	0.966	0.978	0.983
30	0.8439	0.8404	0.8381	0.8370	0.8428	0.8427	0.8341	0.8370	0.8180	0.8134	0.908	0.908	0.920	0.905	0.914	0.917	0.911	0.902	0.896	0.888	0.972	0.972	0.983	0.978	0.978	0.978	0.966	0.966	0.978	0.983
31	0.8439	0.8404	0.8381	0.8370	0.8428	0.8427	0.8341	0.8370	0.8180	0.8134	0.908	0.908	0.920	0.905	0.914	0.917	0.911	0.902	0.896	0.888	0.972	0.972	0.983	0.978	0.978	0.978	0.966	0.966	0.983	0.983
32	0.8439	0.8404	0.8387	0.8370	0.8428	0.8427	0.8341	0.8370	0.8180	0.8134	0.908	0.908	0.920	0.905	0.914	0.917	0.911	0.902	0.911	0.888	0.972	0.972	0.983	0.978	0.978	0.978	0.966	0.966	0.983	0.983
33	0.8439	0.8404	0.8387	0.8370	0.8428	0.8427	0.8341	0.8370	0.8180	0.8134	0.908	0.908	0.920	0.905	0.914	0.917	0.911	0.902	0.911	0.888	0.972	0.972	0.983	0.978	0.978	0.978	0.966	0.966	0.983	0.983
34	0.8439	0.8404	0.8387	0.8370	0.8428	0.8427	0.8341	0.8370	0.8180	0.8180	0.908	0.908	0.920	0.905	0.914	0.917	0.911	0.902	0.911	0.888	0.972	0.972	0.983	0.978	0.978	0.978	0.966	0.966	0.983	0.983
35	0.8439	0.8404	0.8387	0.8370	0.8428	0.8427	0.8341	0.8370	0.8180	0.8180	0.914	0.908	0.920	0.905	0.914	0.917	0.911	0.902	0.911	0.888	0.978	0.972	0.983	0.978	0.978	0.978	0.966	0.966	0.983	0.983
36	0.8439	0.8404	0.8387	0.8370	0.8428	0.8427	0.8341	0.8370	0.8180	0.8180	0.914	0.908	0.920	0.905	0.914	0.917	0.911	0.902	0.911	0.896	0.978	0.972	0.983	0.978	0.978	0.978	0.966	0.966	0.983	0.983
37	0.8439	0.8404	0.8387	0.8370	0.8428	0.8427	0.8341	0.8370	0.8180	0.8180	0.914	0.908	0.920	0.905	0.914	0.917	0.911	0.902	0.911	0.896	0.978	0.972	0.983	0.978	0.978	0.978	0.966	0.966	0.983	0.983
38	0.8439	0.8404	0.8404	0.8370	0.8428	0.8427	0.8387	0.8370	0.8180	0.8180	0.914	0.908	0.920	0.905	0.914	0.917	0.911	0.902	0.911	0.896	0.978	0.972	0.983	0.978	0.978	0.978	0.966	0.966	0.983	0.983
39	0.8439	0.8404	0.8404	0.8370	0.8428	0.8427	0.8387	0.8370	0.8180	0.8180	0.914	0.908	0.920	0.905	0.914	0.917	0.911	0.902	0.911	0.896	0.978	0.972	0.983	0.978	0.978	0.978	0.966	0.966	0.983	0.983

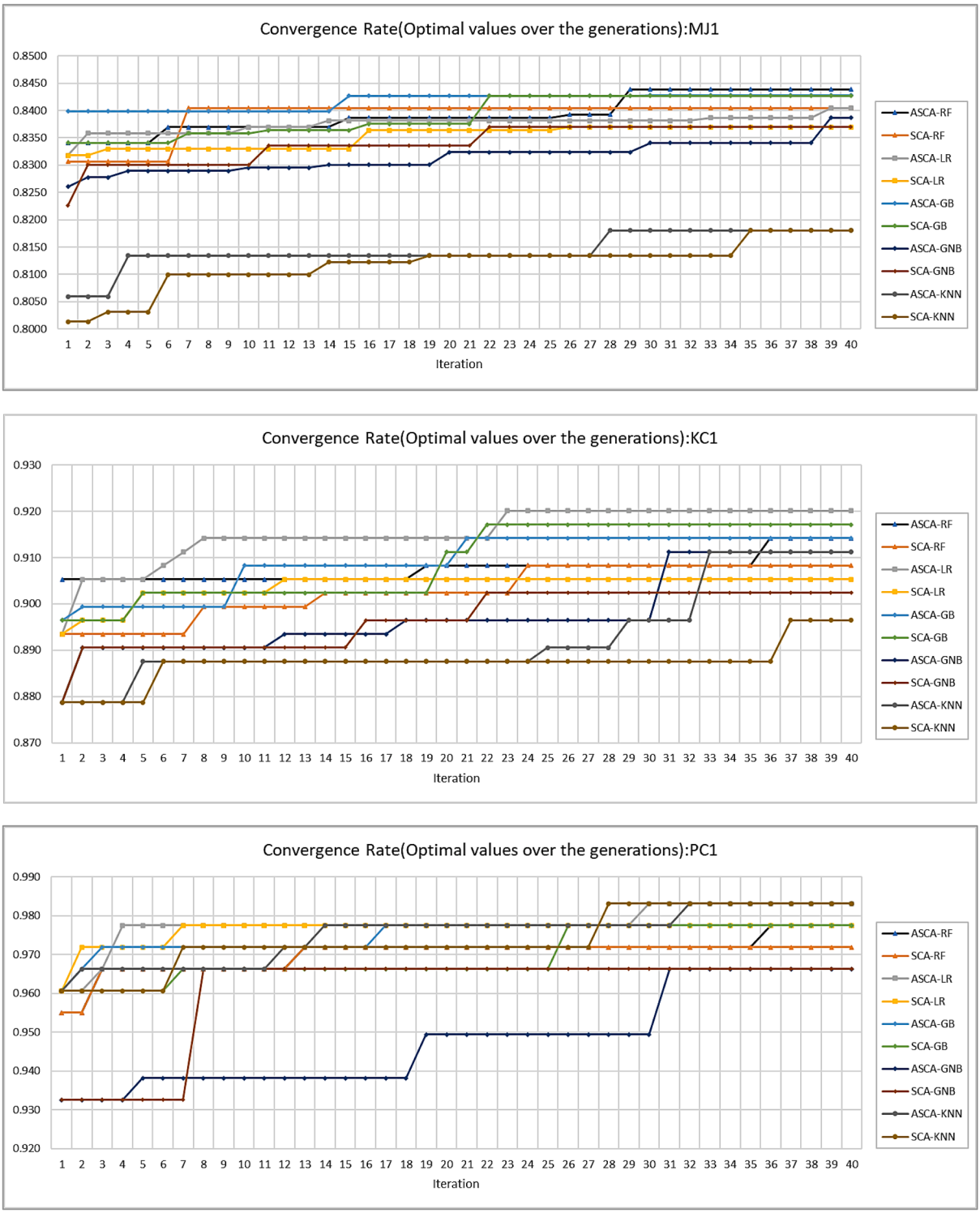


Fig. 6. Visual presentation of convergence rate. Comparisons of ASCA, and SCA on the MJ1, KC1 and PC1 datasets with different classifiers.

lower. The plot box of Fig. 9 shows the distribution of the test score accuracies obtained on the PC1 dataset. The figure shows the maximum, minimum, and mean values of 40 runs. Clearly, the maximum, minimum, and mean are higher using DASC-FS with D-LDA in most cases, except for RF, with the results remaining almost the same. RF combined with DASC-FS and D-LDA performs the best in terms of both the maximum and average scores across the three datasets (JM1, KC1, and PC1), whereas KNN exhibits the lowest performance among all classifiers.

Overall, the results of evaluating the integration of DASC-FS with D-LDA indicate that D-LDA improves the performance of DASC-FS; however, the performance depends on the selected classifier. D-LDA outperforms the other methods because of its ability to address outliers. D-LDA enhances the robustness of scatter matrix estimation by utilizing the matrix depth to find the deepest scatter matrix within classes, which enhances class separability, making it more reliable in challenging data scenarios.

Table 7
Comparisons of D-LDA against existing methods.

Dataset	JM1		KC1		PC1	
Method	Best Accuracy	Average Accuracy	Best Accuracy	Average Accuracy	Best Accuracy	Average Accuracy
FDA	0.797887	0.778663	0.862559	0.833886	0.954955	0.931081
WDA	0.799724	0.785588	0.876777	0.831398	0.963964	0.930856
D_LDA	**0.815342**	**0.795544**	0.881517	**0.853258**	**0.968468**	**0.938514**
LDA	0.802940	0.786438	**0.888626**	0.842476	0.959459	0.925901
PCA	0.808911	0.787150	0.867299	0.832642	**0.968468**	0.930856
RSLDA	0.806615	0.788838	0.862559	0.831694	**0.968468**	0.930743
KLDA	0.809830	0.791238	0.841232	0.812618	0.896396	0.847410
DeepLDA	0.802480	0.783360	**0.888626**	0.844135	0.959459	0.921847

Table 8
Comparisons of D-LDA against existing methods in the presence of outliers.

Dataset	JM1		KC1		PC1	
Method	Best Accuracy	Average Accuracy	Best Accuracy	Average Accuracy	Best Accuracy	Average Accuracy
FDA	0.790997	0.778457	0.853081	0.834123	0.923423	0.914414
WDA	0.796050	0.781902	0.857820	0.835545	0.923423	0.916216
D_LDA	**0.809184**	0.783372	**0.869668**	**0.854265**	**0.927928**	**0.922973**
LDA	0.795131	0.788379	0.860190	0.839810	**0.927928**	0.915315
PCA	0.794672	0.783050	0.855450	0.832701	0.923423	**0.922973**
RSLDA	0.796968	0.784566	0.853081	0.832701	0.923423	**0.922973**
KLDA	0.805237	**0.788470**	0.843602	0.813033	0.842342	0.820270
DeepLDA	0.790997	0.784933	0.857820	0.838152	0.923423	0.908108

4.5. Statistical analysis

In verifying the results of the comparisons, because of the nondeterministic nature of the optimisation algorithm, the ASCA results were compared against those of its counterpart algorithms (such as SCA and GA) using the Wilcoxon signed-rank test for test accuracy. In this experiment, the comparison results included three datasets and five classifiers. The ASCA was run 40 times for each dataset and classifier, and the Wilcoxon signed-rank test was used to determine significant differences between the proposed method and its counterparts. The Wilcoxon signed-rank test statistic represents the sum of the ranks of positive differences between two samples and is used to compare two dependent samples. This section also presents a statistical analysis comparing D-LDA with its counterpart methods.

Table 10 shows the results of the Wilcoxon test for the ASCA comparisons against SCA and GA. The table also presents p-values, negative and positive ranks, ties, total, asymptotic significance (two-tailed) and the Wilcoxon signed-rank test results. The test either rejects the null hypothesis, indicating that there is a significant difference between the two methods, or retains the null hypothesis, meaning that there is no significant difference between the two methods. The decision of the Wilcoxon test is based on the p-value. If the p-value is less than 0.05, the test rejects the null hypothesis; otherwise, it retains the null hypothesis.

According to the p-values, the Wilcoxon test rejected the null hypothesis, indicating a significant difference between the test accuracy of ASCA and those of the other two algorithms (SCA and GA) in all comparisons, except for the comparisons between ASCA-SCA and ASCA-GA for the GNB classifier on the JM1 dataset, the comparisons between ASCA-SCA and ASCA-GA for the GB and KNN classifier on the KC1 dataset, and the comparisons between ASCA and SCA on the PC1 dataset. The cases in which the null hypothesis was retained, indicated that there was no significant difference among the test accuracies of the ASCA, SCA, and GA. However, the positive ranks in the Wilcoxon test confirmed that ASCA outperformed the other methods in most cases.

In summary, this experiment confirms that the use of multiple mutation operators in ASCA improves the overall search process. The search mechanism of ASCA, which introduces various mutation operators, is capable of introducing new genetic material into the population. The adaptation of different mutation operators based on their performance improves the ASCA results in terms of exploration and exploitation although some mutation operators may be more effective in exploring the search space and others may be more effective in escaping from the local optima.

Table 11 presents the results of the Wilcoxon signed-rank test comparing the performance of D-LDA against other LDA methods on three datasets (JM1, KC1, and PC1). The Wilcoxon signed-rank test consistently rejected the null hypothesis, indicating that D-LDA is significantly better than the other methods in terms of accuracy across different datasets. However, in some cases, such as the results of D-LDA against PCA and RSLDA, the null hypothesis is retained, which means that the difference between the results of D-LDA against PCA and RSLDA is not significant, supporting our previous experimental results presented in Section 4.3.1, where both PCA and RSLDA perform very well compared to other existing methods. These findings emphasise the robustness and effectiveness of D-LDA in improving software defect prediction models.

4.6. Analyzing the selected features using DASC-FS

The DASC-FS was used to determine the importance of features based on test scores. While the DASC-FS finds a subset of features that achieve the maximum test scores, it can simultaneously determine the importance of each feature. The score assigned to each feature by the classifier determines feature importance, with higher scores indicating the more important features. Thus, we aimed to analyse and investigate how the selected features with high test accuracy from different software defect datasets influence software defect identification.

In this experiment, the selected features from different datasets were analysed using the DASC-FS method, obtaining the importance of features based on different datasets, which can help identify the subsets of features that are consistently selected and thus have a significant impact on software quality. This analysis provides valuable insights into the characteristics of the selected features and helps software engineers identify software features that are likely to lead to defects. By understanding the consistently selected features across different datasets, software engineers can focus their efforts on improving and optimizing them, leading to more robust and reliable software.

The experiment was conducted on three datasets that shared similar

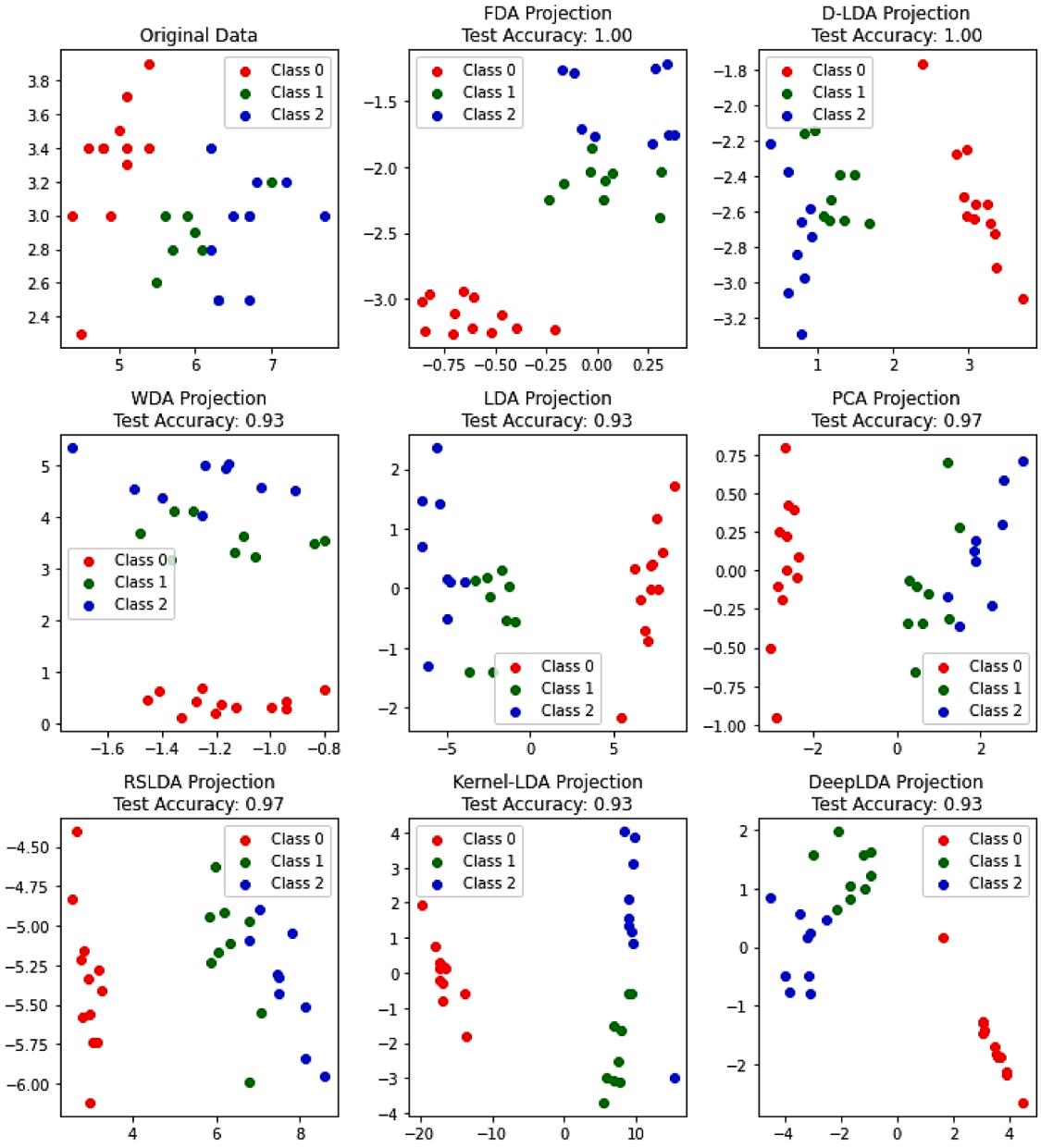


Fig. 7. Data transformation and accuracy comparisons without outliers.

features, using different classifiers, which helped gather additional data to comprehensively determine the importance of these features. DASC-FS separately generated the importance of the features for each classifier; however, Fig. 10 shows the combined results of the feature importance of the five classifiers for the JM1 dataset. The results show that different classifiers have different preferences in selecting feature importance; however, the feature 'loc', which represents the line of code, is ranked as important across all classifiers. Similarly, features such as cyclomatic complexity 'v(g)', design complexity 'iv(g)', total number of potential operands 'n', and Halstead's line count 'loCode' received high scores across most classifiers, suggesting that these features are useful in predicting the defect class. In contrast, other features such as Halstead volume 'v', Halstead's effort 'e', and Halstead's time estimator 't' were found to be less important by most classifiers, with scores ranging from 9 to 21. These features may not be as useful as others in predicting defect class labels.

Among the most important features across all classifiers, certain features received higher importance from specific classifiers, indicating the usefulness of these features for these classifiers. For example, the feature unique operator, 'uniq_Op', received a high score from the RF classifier, suggesting that it may be useful for this classifier. In contrast, the feature unique operands, 'total_Opnd', received poor ratings from most classifiers, suggesting that it may not be as useful for these classifiers.

Generally, the results of the JM1 dataset suggest that the features 'loc', 'v(g)', 'iv(g)', 'n', and 'loCode' are perhaps the most important for predicting the software defects class in the JM1 dataset, whereas features such as 'v', 'e', and 't' may be less useful.

Fig. 11 shows the feature importance scores generated by DASC-FS for the KC1 dataset. The results show that the importance of these features differs from classifier to classifier; however, there are some similar patterns. For example, line count of code 'loc', Halstead's program length 'l', McCabe's essential complexity 'ev(g)', and unique operands 'uniq_Opnd' are scored as the top important features across all five classifiers. In contrast, features such as Halstead's intelligence 'i', total operands 'total_Op', and Halstead's effort 'e' have the lowest importance

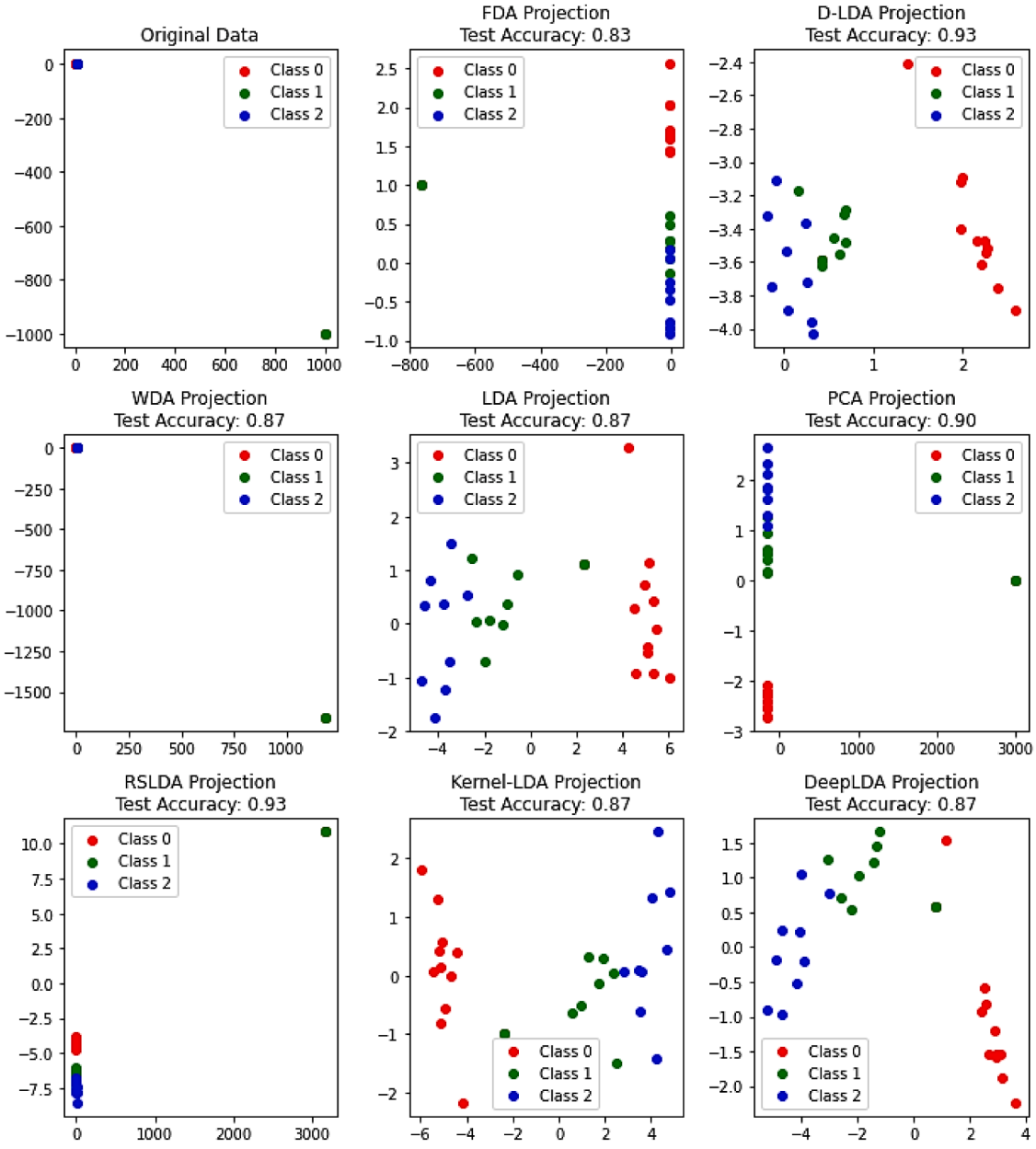


Fig. 8. Data transformation and accuracies compared in the presence of outliers.

Table 9
Comparisons after integrating DASC-FS and D-LDA on the JM1, KC1, and PC1 datasets.

Classifier	Method	JM1		KC1		PC1	
		Maximum	Average	Maximum	Average	Maximum	Average
Random Forest (RF)	DASC-FS without D-LDA	0.83524684	0.82977899	0.905325	0.892751	**0.989382**	**0.976933**
	DASC-FS with D-LDA	**0.85989667**	**0.84385763**	**0.917574**	**0.914201**	0.981916	0.977528
Logistic regression (LR)	DASC-FS without D-LDA	0.83754305	0.83105625	0.908284	0.897485	0.983146	0.973174
	DASC-FS with D-LDA	**0.84044601**	**0.84041331**	**0.926742**	**0.920118**	**0.988764**	**0.983146**
Gradient Boosting (GB)	DASC-FS without D-LDA	0.84156142	0.83306544	0.908284	0.895488	**0.973146**	0.973736
	DASC-FS with D-LDA	**0.84385763**	**0.84279952**	**0.917243**	**0.914201**	**0.978146**	**0.977528**
Gaussian Naive Bayes (GNB)	DASC-FS without D-LDA	0.83639494	0.82698048	0.89645	0.879882	0.971910	0.958567
	DASC-FS with D-LDA	**0.84270952**	**0.83869115**	**0.918284**	**0.911242**	**0.983146**	**0.966292**
K-Nearest Neighbour (KNN)	DASC-FS without D-LDA	**0.81343283**	0.80076062	0.899408	0.87818	0.956528	0.950152
	DASC-FS with D-LDA	0.81871067	**0.81802525**	**0.915325**	**0.911242**	**0.988764**	**0.983146**

scores across all classifiers. Additionally, the McCabe's cyclomatic complexity feature 'v(g)' received a high score from the RF classifier, whereas the LR classifier gives higher importance to the line count of Code and Comment 'locCodeAndComment', indicating that these features may be useful for these classifiers.

Overall, the results of the KC1 dataset suggest that the features 'loc',

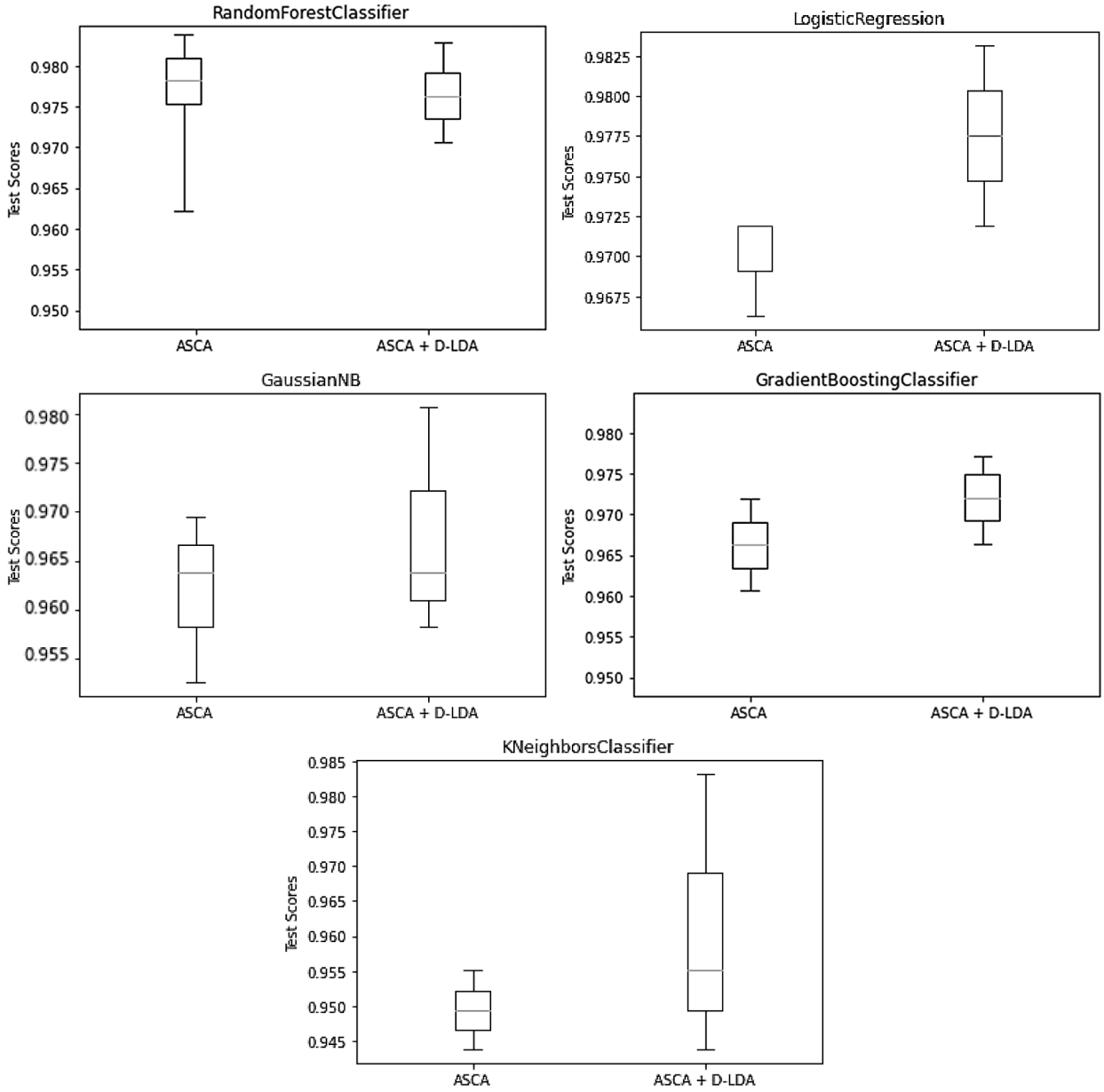


Fig. 9. Comparison of D-LDA results on the JM1, KC1, and PC1 datasets.

'l', 'ev(g)', and 'uniq_Opnd' are perhaps the most important for predicting the software defects class in the KC1 dataset, whereas features such as 'i', 'total_Op', and 'e' may be less useful.

Fig. 12 shows the feature importance scores generated by DASC-FS for the PC1 dataset. The results indicate that the rankings of the features differ across different classifiers, however, some features such as line count of code 'loc', Halstead's program length 'l', Halstead's line count 'lOCode', and McCabe's design complexity 'iv(G)' are consistently ranked as the most important across all classifiers, indicating that these features have a high impact on SDP. On the other hand, features such as Halstead's volume 'V' and Halstead's effort 'E' have the lowest importance scores across all classifiers.

Certain features received high scores from specific classifiers, indicating that they may be useful for these classifiers. For example, the total operator feature 'total_Op' had a high score for the GNB classifier but a low score for the RF, GB, and KNN classifiers. Similarly, 'locCodeAndComment', 'E', and 'D' received the highest scores from GB, KNN, and GNB classifiers, respectively.

Overall, the results of the PC1 dataset suggest that the features 'loc', 'I', 'lOCode', and 'iv(G)' may be the most important for predicting the software defects class in the PC1 dataset, whereas features such as 'locCodeAndComment', 'E', and 'D' may be less useful.

In conclusion, based on the results of the three datasets (JM1, KC1, and PC1), we can conclude that the size and complexity of code are the key factors for SDP, as suggested by the results of the three datasets. Regarding the features associated with size, total lines of code 'loc' (from JM1, KC1, PC1), total number of executable statements 'lOCode' (from JM1, PC1), total number of methods or functions 'n' from JM1, total number of lines of code and comments 'V' from PC1, and the length of the variable and function names 'I' (from KC1, PC1) rank as the highest features across all classifiers, whereas the features associated with complexity are cyclomatic complexity 'v(g)' from JM1, design complexity 'iv(g)' (from JM1, PC1), essential complexity 'ev(g)' from KC1, and design complexity 'iv(G)' from PC1. Additionally, the of unique operands feature 'uniq_Opnd' from KC1 also ranked as an important feature across classifiers.

Table 10
Wilcoxon signed-rank test statistic (ASCA vs SCA) and (ASCA vs GA).

Dataset	Comparison	Classifier	Wilcoxon test statistic	p-value	Negative Ranks	Positive Ranks	Ties	Total	Asymptotic Significance (2-tailed)	Decision
MJ1	ASCA-SCA	RF	15	1.13E-08	34	5	1	40	0.855060308	Reject the null hypothesis
	ASCA-GA		60	0.00874649	20	20	0	40	0.855060308	Reject the null hypothesis
	ASCA-SCA	LR	0	9.80E-08	20	17	3	40	0.805203965	Reject the null hypothesis
	ASCA-GA		0	3.15E-08	14	24	2	40	0.805203965	Reject the null hypothesis
	ASCA-SCA	GB	26	1.08E-07	15	2	23	40	0.899343189	Reject the null hypothesis
	ASCA-GA		23	1.06E-07	32	8	0	40	0.899343189	Reject the null hypothesis
	ASCA-SCA	GNB	341.5	0.668799076	23	15	2	40	0.46823891	Retain the null hypothesis
	ASCA-GA		310.5	0.168963676	25	14	1	40	0.46823891	Retain the null hypothesis
	ASCA-SCA	KNN	183	0.001200464	36	4	0	40	0.857266763	Reject the null hypothesis
	ASCA-GA		191.5	0.001707849	7	33	0	40	0.857266763	Reject the null hypothesis
KC1	ASCA-SCA	RF	21	9.62E-08	21	19	0	40	0.762763534	Reject the null hypothesis
	ASCA-GA		29	1.78E-07	22	18	0	40	0.762763534	Reject the null hypothesis
	ASCA-SCA	LR	0	1.21E-07	7	32	1	40	0.682809756	Reject the null hypothesis
	ASCA-GA		3	1.96E-08	13	27	0	40	0.682809756	Reject the null hypothesis
	ASCA-SCA	GB	211	0.453900595	15	25	0	40	0.558602838	Retain the null hypothesis
	ASCA-GA		20	6.50E-07	24	16	0	40	0.558602838	Reject the null hypothesis
	ASCA-SCA	GNB	0	1.52E-08	22	19	0	40	0.965152019	Reject the null hypothesis
	ASCA-GA		0	1.29E-08	21	16	4	40	0.965152019	Reject the null hypothesis
	ASCA-SCA	KNN	66	0.3829936	8	32	0	40	0.9582454	Retain the null hypothesis
	ASCA-GA		3	1.82E-08	17	23	0	40	0.958245359	Reject the null hypothesis
PC1	ASCA-SCA	RF	55	0.000136975	25	15	0	40	0.934403497	Reject the null hypothesis
	ASCA-GA		84	0.004941593	22	18	0	40	0.934403497	Reject the null hypothesis
	ASCA-SCA	LR	27	0.000118115	24	16	0	40	0.824812574	Reject the null hypothesis
	ASCA-GA		98.5	0.003108748	25	15	0	40	0.824812574	Reject the null hypothesis
	ASCA-SCA	GB	122.5	0.013080761	22	18	0	40	0.741850403	Reject the null hypothesis
	ASCA-GA		77.5	0.000681002	24	16	0	40	0.741850403	Reject the null hypothesis
	ASCA-SCA	GNB	208	0.015854247	21	19	0	40	0.936148947	Reject the null hypothesis
	ASCA-GA		149	0.079246102	18	22	0	40	0.936148947	Retain the null hypothesis
	ASCA-SCA	KNN	94	0.011018486	20	20	0	40	0.76761843	Reject the null hypothesis
	ASCA-GA		94.5	0.037792904	17	23	0	40	0.76761843	Reject the null hypothesis

5. Threats to validity

The experiments conducted in this study potentially face two main pitfalls. Threats to internal validity primarily originate from experimental deviations of the parameter settings. The source code for most of the methods compared in the experiments is not available; hence, some specific information, such as parameter settings, is unknown, and there may be some differences in the results. This threat can be overcome by using published results when available, or by finding the optimal parameter setting. The main external threat to the validity is the dataset. The proposed method was validated using three well-known datasets. However, the performance of the proposed method on other datasets remains unknown.

6. Conclusion and Implications

In this paper, a new FS method called Linear Discrimination-Oriented Feature Selection Method based on Depth Adaptive Sine Cosine Algorithm, namely DASC-FS, was proposed for SDP. DASC-FS utilises ASCA as a search algorithm which is an enhanced version of

Table 11
Wilcoxon signed-rank test statistic for D-LDA vs other methods.

dataset	Comparison	Wilcoxon test statistic	p-value	Negative Ranks	Positive Ranks	Ties	Total	Asymptotic Significance (2-tailed)	Decision
JM1	D_LDA vs FDA	0	5.21E-08	0	39	1	40	1	Reject the null hypothesis
	D_LDA vs WDA	76.5	1.2E-05	8	31	1	40	1.11E-33	Reject the null hypothesis
	D_LDA vs LDA	10.5	1.18E-07	2	37	1	40	0.096875	Reject the null hypothesis
	D_LDA vs PCA	47	3.95E-08	5	35	0	40	1.07E-13	Reject the null hypothesis
	D_LDA vs RSLDA	77.5	1.29E-05	9	30	1	40	1.6E-34	Reject the null hypothesis
	D_LDA vs KLDA	0	1.82E-12	0	40	0	40	1	Reject the null hypothesis
	D_LDA vs DeepLDA	2	5.46E-12	1	39	0	40	0.75183	Reject the null hypothesis
KC1	D_LDA vs FDA	71	8.33E-06	4	35	1	40	3.04E-29	Reject the null hypothesis
	D_LDA vs WDA	115	0.000122	11	28	1	40	7.03E-74	Reject the null hypothesis
	D_LDA vs LDA	187.5	0.036641	13	22	5	40	3.8E-193	Reject the null hypothesis
	D_LDA vs PCA	144.5	0.001774	10	27	3	40	1.5E-115	Reject the null hypothesis
	D_LDA vs RSLDA	186.5	0.002166	11	29	0	40	4.1E-191	Reject the null hypothesis
	D_LDA vs KLDA	3.5	6.86E-08	1	38	1	40	0.579991	Reject the null hypothesis
	D_LDA vs DeepLDA	252.5	0.086737	18	20	2	40	0	**Retain the null hypothesis**
PC1	D_LDA vs FDA	65.5	0.000115	7	26	7	40	3.91E-25	Reject the null hypothesis
	D_LDA vs WDA	31.5	2.15E-05	6	25	9	40	6.34E-07	Reject the null hypothesis
	D_LDA vs LDA	132.5	0.008027	9	24	7	40	1.87E-97	Reject the null hypothesis
	D_LDA vs PCA	205	0.112168	13	21	6	40	1.8E-230	**Retain the null hypothesis**
	D_LDA vs RSLDA	218	0.053545	18	21	1	40	2.4E-260	**Retain the null hypothesis**
	D_LDA vs KLDA	110.5	0.001365	10	24	6	40	2.36E-68	Reject the null hypothesis
	D_LDA vs DeepLDA	134.5	0.005218	8	26	6	40	2.3E-100	Reject the null hypothesis

the SCA combined with multiple mutation operators of the GA. The DASC-FS also utilises D-LDA to identify the most discriminative features and enhances the feature selection process. The proposed D-LDA aims to enhance the robustness of the original LDA by utilising the concept of matrix depth to find the deepest scatter matrix within classes. Several experiments were conducted on three datasets (JM1, KC1, and PC1) to evaluate the effectiveness of the proposed method. First, DASC-FS was evaluated against other FS methods in terms of test accuracy. Second, the performance of the ASCA was evaluated statistically as an optimisation algorithm against the original SCA and GA, to verify the effectiveness of introducing mutation operators into the original SCA in terms of convergence rate. Third, the integration of D-LDA and DASC-FS was evaluated to assess the effectiveness of introducing D-LDA into DASC-FS. The final experiment analysed the selected features generated by the DASC-FS method for different datasets based on their importance in identifying consistently selected subsets of features, which had a high impact on software quality. The experimental results showed that DASC-FS consistently achieved the highest classification accuracy among FS methods for SDP, effectively mitigating irrelevant features. DASC-FS achieved competitive results compared with other methods, such as SCA-FS, RFE-FS, and EM-FS, achieving the highest accuracy scores for all classifiers. In terms of the selected features, DASC-FS selected fewer features than GA-FS and SCA-FS in most cases but more features than RFE-FS and EM-FS. Introducing multiple mutation operators into ASCA, improved its capability for exploring the search space more efficiently. The results also showed that the integration of DASC-FS and D-LDA had a positive impact in most cases; however, the performance depended on the selected classifier. Features related to the size and complexity of the code are key factors for SDP, as they were consistently ranked as important features by DASC-FS across different classifiers and datasets.

Overall, the proposed DASC-FS showed superior performance against comparison methods, offering a comprehensive and effective approach for FS in the context of SDP. DASC-FS has several strengths and advantages, including an ability to identify the relevant features with high discriminatory power by systematically incorporating the matrix depth concept into LDA. It also utilises ASCA to efficiently identify relevant features. ASCA enhances the search process by introducing multiple mutation operators through GA, enabling the search for the best feature sets in feature space, which contributes to the accuracy of the prediction model. The combination of ASCA and D-LDA allows DASC-FS to optimize FS by considering optimal accuracy and class separation. This two-factor approach guarantees that the selected features improve the overall accuracy of the SDP model and capture complex patterns in the data. DASC-FS, with its generalizability and adaptability, has high potential for extension beyond SDP. The efficient search of ASCA and robust scatter matrix D-LDA of DASC-FS are transferrable to other domains such as finance, healthcare, and manufacturing, where accuracy prediction is critical. Additionally, the use of multiple mutation operators in ASCA allows DASC-FS to efficiently increase the diversity of the solutions, making it suitable for

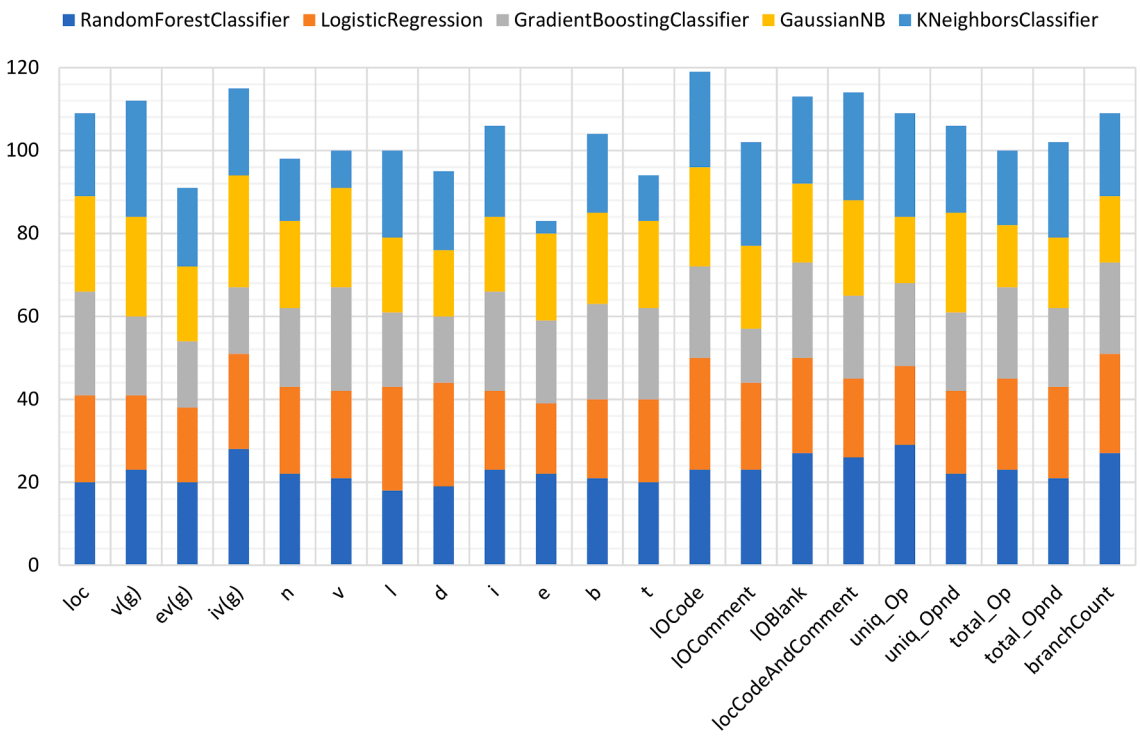


Fig. 10. Importance of features in the JM1 dataset using different classifiers.

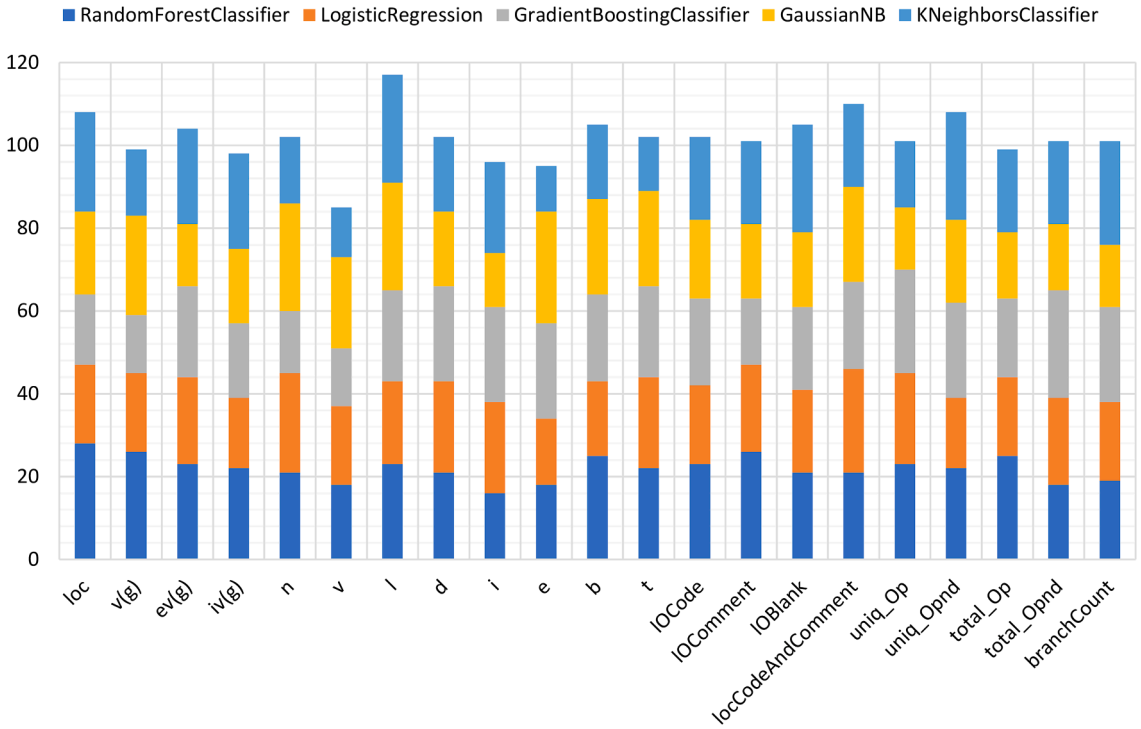


Fig. 11. Importance of features in the KC1 dataset using different classifiers.

problems in different contexts.

In terms of practical implications, as demonstrated by the results, DASC-FS consistently outperformed other feature selection methods in terms of prediction accuracy and irrelevant feature mitigation. The DASC-FS method can be utilised by software development companies to improve software quality, reduce maintenance costs, and enhance customer satisfaction before releasing their products to the market. Moreover, the mitigation of irrelevant features realised by integrating ASCA into DASC-FS reduces the complexity of feature sets and aids in model interpretability and computational efficiency. The finding that features related to the size and complexity of the code are consistently ranked as important by DASC-FS across different classifiers and datasets, is significant for software developers, making these features key factors for SDP. This insight can guide resource allocation, risk management,

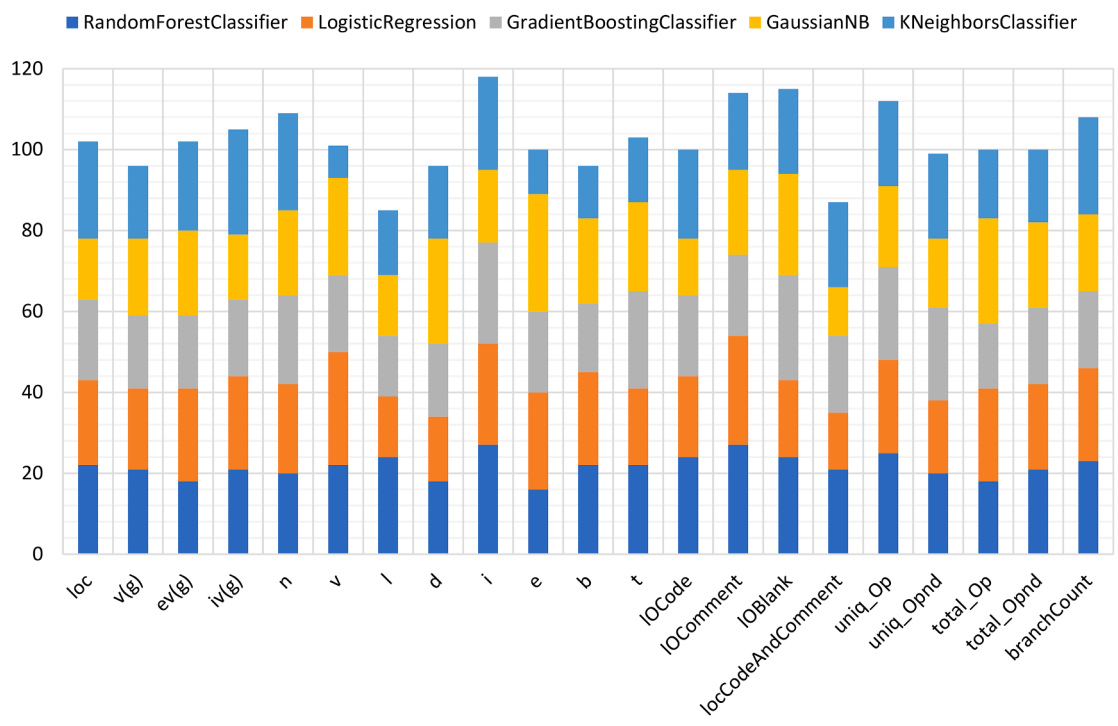


Fig. 12. Importance of features in the PC1 dataset using different classifiers.

tool development, and process improvement in software development, leading developers to prioritize testing and debugging efforts in areas of code that are larger and more complex.

From a theoretical perspective, the adaptation of multiple mutations into ASCA plays a crucial role in achieving more efficient and effective convergence, making it a promising choice for enhancing the effectiveness of optimization algorithms in feature selection tasks or other optimization problems, such as global optimization, cost minimization, resource allocation, energy efficiency, and supply chain optimization, to name a few. Additionally, the proposed D-LDA, which utilizes matrix depth to identify discriminative features, aims to reduce data dimensionality and increase class separation, yielding highly competitive results compared to the original LDA. The proposed innovation potentially leads to more accurate prediction models in several domains beyond SDP.

However, the limitations of our study must be acknowledged. First, the performance of DASC-FS was evaluated using three well-known datasets of SDP. The performance on other datasets remains unknown. Future research could explore the applicability of DASC-FS using different datasets in different domains. Second, the proposed D-LDA requires tuning of the depth parameter (D) to explore different depths, which may increase the complexity of implementation. Additionally, compared to the filter FS methods, DASC-FS is more computationally complex, necessitating that different feature combinations be explored.

In future work, the application of DASC-FS can be expanded by comparing it with other FS methods using various datasets. In another direction, the performance of the DASC-FS can be improved by hybridising ASCA with other optimization algorithms or search operators. DASC-FS not only advances the field of SDP but also introduces a flexible metaheuristic algorithm (ASCA) and a new linear discriminant method (D-LDA) with potential applications beyond the scope of SDP.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

Abbas, Q., Ahmad, J., & Jabeen, H. (2015). A novel tournament selection based differential evolution variant for continuous optimization problems. *Mathematical Problems in Engineering, 2015*.

Abdi, H., & Williams, L. J. (2010). Principal component analysis. *Wiley interdisciplinary reviews: computational statistics, 2*(4), 433–459.

Abualigah, L., & Diabat, A. (2021). Advances in sine cosine algorithm: A comprehensive survey. *Artificial Intelligence Review, 54*(4), 2567–2608.

Adorada, A., Wirawan, P. W., & Kurniawan, K. (2020). The Comparison of Feature Selection Methods in Software Defect Prediction. *Paper presented at the 2020 4th International Conference on Informatics and Computational Sciences (ICICoS)*.

Agrawal, P., Abutarboush, H. F., Ganesh, T., & Mohamed, A. W. (2021). Metaheuristic algorithms on feature selection: A survey of one decade of research (2009–2019). *IEEE Access, 9*, 26766–26791.

Ali, M., Mazhar, T., Al-Rasheed, A., Shahzad, T., Ghadi, Y. Y., & Khan, M. A. (2024). Enhancing software defect prediction: A framework with improved feature selection and ensemble machine learning. *PeerJ Computer Science, 10*, e1860.

Alkhasawneh, M. S. (2022). *2022*. Applied Computational Intelligence and Soft Computing: Software defect prediction through neural network and feature selections.

Alsghaier, H., & Akour, M. (2020). Software fault prediction using particle swarm algorithm with genetic algorithm and support vector machine classifier. *Software: Practice and Experience, 50*(4), 407–427.

Anbu, M., & Anandha Mala, G. (2019). Feature selection using firefly algorithm in software defect prediction. *Cluster Computing, 22*, 10925–10934.

Balogun, A. O., Basri, S., Mahamad, S., Abdulkadir, S. J., Almomani, M. A., Adeyemo, V. E., & Bajeh, A. O. (2020). Impact of feature selection methods on the predictive performance of software defect prediction models: An extensive empirical study. *Symmetry, 12*(7), 1147.

Balogun, A. O., Basri, S., Mahamad, S., Capretz, L. F., Imam, A. A., Almomani, M. A., … Kumar, G. (2021). A novel rank aggregation-based hybrid multifilter wrapper feature selection method in software defect prediction. *Computational Intelligence and Neuroscience, 2021*.

Benala, T. R., & Tantati, K. (2022). Efficiency of oversampling methods for enhancing software defect prediction by using imbalanced data. *Innovations in Systems and Software Engineering*, 1–17.

Chen, L., Fang, B., Shang, Z., & Tang, Y. (2018). Tackling class overlap and imbalance problems in software defect prediction. *Software Quality Journal, 26*, 97–125.

Chen, M., Gao, C., & Ren, Z. (2018). Robust covariance and scatter matrix estimation under Huber's contamination model. *Ann. Statist., 46*(5), 1932–1960.
Curcio, K., Malucelli, A., Reinehr, S., & Paludo, M. A. (2016). An analysis of the factors determining software product quality: A comparative study. *Computer Standards & Interfaces, 48*, 10–18.
Das, H., Prajapati, S., Gourisaria, M. K., Pattanayak, R. M., Alameen, A., & Kolhar, M. (2023). Feature selection using golden jackal optimization for software fault prediction. *Mathematics, 11*(11), 2438.
De Falco, I., Della Cioppa, A., & Tarantino, E. (2002). Mutation-based genetic algorithm: Performance evaluation. *Applied Soft Computing, 1*(4), 285–299.
De, G. (1989). *Genetic algorithms in search*. Optimization and Machine Learning.
Deb, K. (2011). *Multi-objective optimisation using evolutionary algorithms: An introduction*. Springer.
Dokeroglu, T., Deniz, A., & Kiziloz, H. E. (2022). A comprehensive survey on recent metaheuristics for feature selection. *Neurocomputing*.
Dorfer, M., Kelz, R., & Widmer, G. (2015). Deep linear discriminant analysis. *arXiv preprint arXiv:1511.04707*.
Durrant, R. J., & Kabán, A. (2010). Compressed Fisher linear discriminant analysis: Classification of randomly projected data. *Paper presented at the Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*.
Elish, K. O., & Elish, M. O. (2008). Predicting defect-prone software modules using support vector machines. *Journal of Systems and Software, 81*(5), 649–660.
Flamary, R., Cuturi, M., Courty, N., & Rakotomamonjy, A. (2018). Wasserstein discriminant analysis. *Machine Learning, 107*, 1923–1945.
Ghotra, B., McIntosh, S., & Hassan, A. E. (2017). A large-scale study of the impact of feature selection techniques on defect classification models. *Paper presented at the 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*.
Gong, L., Zhang, H., Zhang, J., Wei, M., & Huang, Z. (2022). A comprehensive investigation of the impact of class overlap on software defect prediction. *IEEE Transactions on Software Engineering*.
Goyal, S. (2022a). Genetic evolution-based feature selection for software defect prediction using SVMs. *Journal of Circuits, Systems and Computers, 31*(11), 2250161.
Goyal, S. (2022b). Software fault prediction using evolving populations with mathematical diversification. *Soft Computing, 26*(24), 13999–14020.
Guo, W.-Y., Wang, Y., Dai, F., & Xu, P. (2020). Improved sine cosine algorithm combined with optimal neighborhood and quadratic interpolation strategy. *Engineering Applications of Artificial Intelligence, 94*, Article 103779.
Hassouneh, Y., Turabieh, H., Thaher, T., Tumar, I., Chantar, H., & Too, J. (2021). Boosted whale optimization algorithm with natural selection operators for software fault prediction. *IEEE Access, 9*, 14239–14258.
Iqbal, A., & Aftab, S. (2020). A classification framework for software defect prediction using multi-filter feature selection technique and MLP. *International Journal of Modern Education & Computer Science, 12*(1).
Iqbal, A., Aftab, S., Ullah, I., Bashir, M. S., & Saeed, M. A. (2019). A feature selection based ensemble classification framework for software defect prediction. *International Journal of Modern Education and Computer Science, 11*(9), 54.
Jia, L. (2018). A hybrid feature selection method for software defect prediction. *Paper presented at the IOP Conference Series*. Materials Science and Engineering.
Kakkar, M., & Jain, S. (2016). *Feature selection in software defect prediction: A comparative study*. Paper presented at the 2016 6th International Conference-Cloud System and Big Data Engineering (Confluence).
Khan, M. A., Elmitwally, N. S., Abbas, S., Aftab, S., Ahmad, M., Fayaz, M., & Khan, F. (2022). Software defect prediction using artificial neural networks: A systematic literature review. *Scientific Programming, 2022*.
Kuhn, D., Esfahani, P. M., Nguyen, V. A., & Shafieezadeh-Abadeh, S. (2019). Wasserstein distributionally robust optimization: Theory and applications in machine learning. In *Operations research & management science in the age of analytics* (pp. 130–166). Informs.
Lee, J., Choi, J., Ryu, D., & Kim, S. (2022). Holistic Parameter Optimization for Software Defect Prediction. *IEEE Access, 10*, 106781–106797.
Luo, T., Hou, C., Nie, F., & Yi, D. (2018). Dimension reduction for non-Gaussian data by adaptive discriminative analysis. *IEEE transactions on cybernetics, 49*(3), 933–946.
Malhotra, R., Nishant, N., Gurha, S., & Rathi, V. (2021). Application of particle swarm optimization for software defect prediction using object oriented metrics. *Paper presented at the 2021 11th International Conference on Cloud Computing*. Data Science & Engineering (Confluence).
Mastery, M. L. (2018). An Introduction to Feature Selection. *Last accessed: January, 3*.
Maxim, B. R., & Pressman, R. S. (2014). *Software engineering: A Practitioner'S approach*. Britania Raya: McGraw-Hill Education.
Menzies, T., Greenwald, J., & Frank, A. (2006). Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering, 33*(1), 2–13.
Mirjalili, S. (2016). SCA: A sine cosine algorithm for solving optimization problems. *Knowledge-Based Systems, 96*, 120–133.
Mitchell, M. (1998). *An introduction to genetic algorithms*. MIT press.
Mohammad, U. G., Imtiaz, S., Shakya, M., Almadhor, A., & Anwar, F. (2022). An optimized feature selection method using ensemble classifiers in software defect prediction for healthcare systems. *Wireless Communications and Mobile Computing, 2022*.
Mustaqeem, M., Mustajab, S., & Alam, M. (2024). A hybrid approach for optimizing software defect prediction using a gray wolf optimization and multilayer perceptron. *International Journal of Intelligent Computing and Cybernetics*.
Nemati, S., Basiri, M. E., Ghasem-Aghaee, N., & Aghdam, M. H. (2009). A novel ACO–GA hybrid algorithm for feature selection in protein function prediction. *Expert Systems with Applications, 36*(10), 12086–12094.
Osman, H., Ghafari, M., & Nierstrasz, O. (2017). *Automatic feature selection by regularization to improve bug prediction accuracy*. Paper presented at the 2017 IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE).
Pandey, S. K., & Tripathi, A. K. (2021). An empirical study toward dealing with noise and class imbalance issues in software defect prediction. *Soft Computing, 25*(21), 13465–13492.
Pudjihartono, N., Fadason, T., Kempa-Liehr, A. W., & O'Sullivan, J. M. (2022). A review of feature selection methods for machine learning-based disease risk prediction. *Frontiers in Bioinformatics, 2*, Article 927312.
Salgotra, R., & Singh, U. (2017). Application of mutation operators to flower pollination algorithm. *Expert Systems with Applications, 79*, 112–129.
Sekaran, K., & Lawrence, S. P. A. (2024). Mutation boosted salp swarm optimizer meets rough set theory: A novel approach to software defect detection. *Transactions on Emerging Telecommunications Technologies, 35*(3), e4953.
Sharma, M., & Kaur, P. (2021). A comprehensive analysis of nature-inspired meta-heuristic techniques for feature selection problem. *Archives of Computational Methods in Engineering, 28*, 1103–1127.
Shepperd, M., Bowes, D., & Hall, T. (2014). Researcher bias: The use of machine learning in software defect prediction. *IEEE Transactions on Software Engineering, 40*(6), 603–616.
Singh, S., & Haider, T. U. (2022). Selection of best feature reduction method for module-based software defect prediction. *Paper presented at the Journal of Physics: Conference Series*.
Thirumoorthy, K. (2022). A feature selection model for software defect prediction using binary Rao optimization algorithm. *Applied Soft Computing, 131*, Article 109737.
Ullah, Z., Naqvi, S. R., Farooq, W., Yang, H., Wang, S., & Vo, D.-V.-N. (2021). A comparative study of machine learning methods for bio-oil yield prediction–A genetic algorithm-based features selection. *Bioresource Technology, 335*, Article 125292.
Wahono, R. S. (2015). A systematic literature review of software defect prediction. *Journal of software engineering, 1*(1), 1–16.
Wahono, R. S., & Herman, N. S. (2014). Genetic feature selection for software defect prediction. *Advanced Science Letters, 20*(1), 239–244.
Wang, K., Liu, L., Yuan, C., & Wang, Z. (2021). Software defect prediction model based on LASSO–SVM. *Neural Computing and Applications, 33*, 8249–8259.
Wang, P. (2012). Mutual information-based feature selection approach for software defect prediction. *Journal of Computer Applications, 32*(06), 1738.
Wen, J., Fang, X., Cui, J., Fei, L., Yan, K., Chen, Y., & Xu, Y. (2018). Robust sparse linear discriminant analysis. *IEEE Transactions on Circuits and Systems for Video Technology, 29*(2), 390–403.
Yang, J., & Honavar, V. (1998). Feature subset selection using a genetic algorithm. *IEEE Intelligent Systems and their Applications, 13*(2), 44–49.
Yang, X., Zheng, M., & Liu, L. (2023). Robust Two-dimensional T $\ell_1 $-norm linear discriminant analysis for image recognition. *IEEE Signal Processing Letters*.
Yu, L., & Liu, H. (2004). Efficient feature selection via analysis of relevance and redundancy. *The Journal of Machine Learning Research, 5*, 1205–1224.
Zhang, C., Soda, P., Bi, J., Fan, G., Almpanidis, G., García, S., & Ding, W. (2023). An empirical study on the joint impact of feature selection and data resampling on imbalance classification. *Applied Intelligence, 53*(5), 5449–5461.
Zhu, K., Ying, S., Zhang, N., & Zhu, D. (2021). Software defect prediction based on enhanced metaheuristic feature selection optimization and a hybrid deep neural network. *Journal of Systems and Software, 180*, Article 111026.