Enhanced Software Defect Prediction Using Ensemble Learning by Adaptive Variable Sparrow Search Algorithm

Dissertation submitted in partial fulfillment of the requirement

for the award of the degree of

MASTER OF COMPUTER APPLICATIONS

By

Vanama Nikhith

23VV1F0027

Under the Esteemed Guidance of

Dr. B. Tirimula Rao, M.Tech, Ph.D.

Assistant Professor

Department of Information Technology

DEPARTMENT OF INFORMATION TECHNOLOGY

JNTUGV College of Engineering Vizianagaram (Autonomous)

Jawaharlal Nehru Technological University, Gurajada Vizianagaram

Dwarapudi, Vizianagaram-535003, Andhra Pradesh, India

2024-2025

DEPARTMENT OF INFORMATION TECHNOLOGY

JNTUGV COLLEGE OF ENGINEERING

VIZIANAGARAM

Certificate

This is to certify that the Dissertation report entitled Enhanced Software Defect Prediction Using Ensemble Learning by Adaptive Variable Sparrow Search Algorithm that is being submitted by Vanama Nikhith bearing registration number (23VV1F0027) in partial fulfillment for the degree of Master of Computer Applications (MCA) from Jawaharlal Nehru Technological University Gurajada Vizianagaram - College of Engineering Vizianagaram.This bonafide work was carried out by him under my guidance and supervision during the year 2024 - 2025.

The result embodied in this report has not been submitted to any other University or Institute for the award of any degree or diploma.

Signature of Project Guide

Dr. B. TIRIMULA RAO

Assistant Professor

Dept. of Information Technology

Signature of Head of the Department

Dr. Ch. Bindu Madhuri

Assistant Professor & HOD

Dept. of Information Technology

(Signature of External Examiner)

sparrow

ORIGINALITY REPORT

28%
SIMILARITY INDEX

PRIMARY SOURCES

1	Yu Tang, Qi Dai, Mengyuan Yang, Tony Du, Lifang Chen. "Software defect prediction ensemble learning algorithm based on adaptive variable sparrow search algorithm", International Journal of Machine Learning and Cybernetics, 2023 Crossref	674 words — 6%
2	www.coursehero.com Internet	355 words — 3%
3	link.springer.com Internet	212 words — 2%
4	dspace.dtu.ac.in:8080 Internet	138 words — 1%
5	www.mdpi.com Internet	97 words — 1%
6	openaccess.altinbas.edu.tr Internet	67 words — 1%
7	dspace.bracu.ac.bd Internet	46 words — < 1%
8	Misbah Ali, Tehseen Mazhar, Yasir Arif, Shaha Al-Otaibi, Yazeed Yasin Ghadi, Tariq Shahzad, Muhammad Amir Khan, Habib Hamam. "Software Defect Prediction Using an Intelligent Ensemble-Based Model", IEEE Access, 2024 Crossref	43 words — < 1%

DECLARATION

I **Vanama Nikhith (Reg. No: 23VV1F0027)**, declare that this written submission represents my ideas in my own words and where others' ideas or words have been included. I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea data fact source in my submission. I understand that any violation of the above will be cause for disciplinary action by the institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

(Signature)

V.Nikhith

(23VV1F0027)

Date :

Place :

DEPARTMENT OF INFORMATION AND TECHNOLOGY
JNTU-GV College of Engineering Vizianagaram (Autonomous)
Jawaharlal Nehru Technological University, Gurajada Vizianagaram
Dwarapudi, Vizianagaram - 535003, Andhra Pradesh, India
2024 - 2025
Website : www.jntukucev.ac.in

Subject Name : Dissertation **Regulation :** R20

Subject Code : MCA4103 **Academic Year :** 2025

CO'S

Course Outcomes

CO No.	Course Outcome (CO)
CO1	Understand the fundamentals of software defect prediction and its significance in software quality assurance.
CO2	Apply ensemble learning techniques for improving the accuracy of software defect detection models.
CO3	Implement and analyze the Adaptive Variable Sparrow Search Algorithm (AVSSA) for parameter optimization in predictive models.
CO4	Evaluate and compare the performance of various ensemble classifiers on benchmark software defect datasets.
CO5	Design a robust defect prediction framework combining AVSSA with ensemble methods to enhance software reliability.

CO-PO Mapping

Mapping of Course Outcomes (COs) with Program Outcomes (POs)

Course Out Comes	Program Outcomes (POs)														
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO 10	PO 11	PO 12	PSO 1	PSO 2	PSO 3
CO1	3	3	1	2	2	-	-	-	3	3	2	3	3	3	2
CO2	3	3	2	2	2	-	-	-	2	3	3	2	3	2	2
CO3	3	3	2	1	3	-	-	1	2	2	3	3	3	3	2
CO4	2	3	1	2	2	-	-	1	2	3	2	2	2	3	1
CO5	3	2	2	3	1	-	-	-	2	3	1	3	1	2	1

Enter correlation levels 1,2 and 3 as given below:
1.Slight(low) 2.Moderate(Medium) 3.Substantial(High) If there no correlation put "-"

Signature of the Student **Signature of the Guide** **Signature of the HOD**

ACKNOWLEDGEMENT

This acknowledgment transcends the reality of formality when I express deep gratitude and respect to all those people behind the screen who inspired and helped us in the completion of this project work.

I take the privilege to express my heartfelt gratitude to my guide **Dr. B. TIRIMULA RAO**, Assistant Professor , Department of Information Technology, JNTUGV-CEV for his valuable suggestions and constant motivation that greatly helped me in the successful completion of the project. Wholehearted cooperation and the keen interest shown by him at all stages are beyond words of gratitude.

With great pleasure and privilege, I wish to express my heartfelt sense of gratitude and indebtedness to **Dr. Ch. Bindu Madhuri**, Assistant Professor, Head of the Department, JNTUGV-CEV, for her supervision.

I express my sincere thanks to Project Coordinator **Dr. Ch. Bindu Madhuri**, Assistant Professor, Head of the Department of Information Technology, JNTU-GURAJADA VIZIANAGARAM for her continuous support.

I extend heartfelt thanks to our principal **Prof. R. Rajeswara Rao** for providing intensive support throughout my project.

I am also thankful to all the Teaching and Non-Teaching staff of the Information Technology Department, JNTUGV-CEV, for their direct and indirect help provided to me in completing the project.

I extend my thanks to my parents and friends for their help and encouragement in the success of my project

V.Nikhith
(23VV1F0027)

ABSTRACT

Software defect prediction (SDP) plays a critical role in maintaining software quality by identifying defect-prone modules early in the development lifecycle. Traditional machine learning models often rely on fixed configurations and may not generalize well to the dynamic nature of real-world software projects.

This research proposes an enhanced ensemble learning framework for software defect prediction, optimized using the Adaptive Variable Sparrow Search Algorithm (AVSSA). The AVSSA is employed to fine-tune the hyperparameters of multiple ensemble classifiers, improving their ability to generalize and adapt to varying software defect patterns.

To further improve defect detection performance, the framework incorporates cost-sensitive learning to handle class imbalance, a common challenge in SDP datasets. Comprehensive experiments conducted on benchmark defect datasets show that the AVSSA-optimized ensemble models significantly outperform conventional approaches in terms of accuracy, precision, recall, and F1-score.

By integrating intelligent optimization with ensemble learning, this research contributes to the development of scalable, accurate, and adaptive software defect prediction models that are well-suited for modern, rapidly evolving software engineering environments.

Contents

List of Figures

Abbreviations

AVSSA	Adaptive Variable Sparrow Search Algorithm
SVM	Support Vector Machine
SDP	Software Defect Prediction
ELM	Extreme Learning Machine
SEB	Sparrow Eagle Based algorithm
SSA	Sparrow Search Algorithm
TP	True Positives
TN	True Negatives
FP	False Positives
FN	False Negatives
TPR	True Positive Rate (Recall)
TNR	True Negative Rate
MCC	Matthews Correlation Coefficient
ROC-AUC	Receiver Operating Characteristic - Area Under Curve
HDSP	High Dimensional Single Peak
HDMP	High Dimensional Multiple Peaks
APVs	Adjusted P-Values

Chapter 1

Introduction

1.1 Enhanced Software Defect Prediction Using Ensemble Learning by Adaptive Variable Sparrow Search Algorithm

Machine Learning (ML) is a pivotal domain of Artificial Intelligence that enables systems to learn from data and make predictions or decisions without being explicitly programmed. It has transformed various sectors including finance, healthcare, transportation, and particularly software engineering. In the software development lifecycle, predicting and preventing software defects is essential to improve software quality, reduce maintenance costs, and improve system reliability.

Software Defect Prediction (SDP) involves applying ML algorithms to historical software metrics—such as code complexity, change logs, and defect histories—to predict whether new or existing software modules are likely to contain defects. Such predictive capabilities help prioritize quality assurance and testing efforts more effectively.

To enhance the prediction performance, this research introduces an ensemble learning framework optimized using the **Adaptive Variable Sparrow Search Algorithm (AVSSA)**. The AVSSA dynamically tunes hyper parameters of multiple ensemble classifiers, such as Random Forest, SVM, and ELM, ensuring better generalization and adaptability to varying defect patterns in software datasets.

A major concern in defect prediction is the issue of **defective overlooking**, where defective instances may go undetected or misclassified due to model limitations or changing data distributions. This framework addresses the problem by integrating intelligent optimization through AVSSA and robust ensemble mechanisms, which collectively improve the recall and reduce the number of overlooked defects.

The proposed approach is validated using real-world open-source software defect datasets. Comparative experimental results with baseline models indicate that the AVSSA-optimized ensemble framework outperforms conventional approaches in terms of accuracy, precision, recall, F1-score, and ROC-AUC. The integration of AVSSA not only boosts predictive power but also enhances the robustness and reliability of the defect prediction system in dynamic software environments.

1.1.1 Working of Ensemble Learning with Adaptive Variable Sparrow Search Algorithm

Ensemble learning combines the predictions of multiple base classifiers to improve the overall accuracy and robustness of software defect prediction. Unlike traditional batch learning that trains a single model on a static dataset, the proposed framework integrates multiple classifiers and optimizes their hyperparameters using the **Adaptive Variable Sparrow Search Algorithm (AVSSA)**.

The process begins with the collection and preprocessing of software metrics and defect labels from open-source repositories. Multiple base learners such as Random Forest, SVM, and ELM are assembled into an ensemble framework. The AVSSA algorithm then optimizes their parameters to ensure the best combination of predictive performance and generalization.

During training, the AVSSA-guided optimization intelligently searches for optimal parameter values while considering factors such as feature interactions, misclassification penalties, and classifier diversity. This process also incorporates strategies to minimize **defective overlooking**, where defective instances are commonly misclassified due to class imbalance or noisy features.

This iterative optimization and evaluation cycle enhances model performance in terms of accuracy, recall, and F1-score, while maintaining robustness across different datasets. The ensemble, once optimized, produces reliable predictions and effectively adapts to variations in software defect characteristics.

1.1.2 Advantages and Challenges of Ensemble Learning with Adaptive Variable Sparrow Search Algorithm

Advantages:

- **Improved Prediction Accuracy:** Ensemble learning combines the strengths of multiple classifiers, reducing variance and bias.
- **Optimization Efficiency:** AVSSA effectively tunes hyperparameters, enhancing the performance of each ensemble component.
- **Reduction in Defective Overlooking:** The optimized ensemble detects more subtle defect patterns, minimizing the risk of missing defective modules.
- **Robustness to Imbalanced Data:** Integration with resampling techniques and cost-sensitive strategies improves recall for minority defect classes.
- **Adaptability Across Datasets:** The model generalizes well across diverse defect datasets due to the adaptive search capabilities of AVSSA.

Challenges:

- **Hyperparameter Complexity:** Tuning multiple classifiers increases search space and computational cost.
- **Model Interpretability:** Combining several models may reduce transparency and make debugging more difficult.
- **Balancing Bias-Variance Trade-off:** Improper ensemble configuration can lead to overfitting or underfitting.

- **Resource Requirements:** Optimization with AVSSA and ensemble execution requires higher computational resources.
- **Parameter Sensitivity in AVSSA:** Performance may vary with AVSSA's internal parameters such as population size and iteration count.

This research addresses these challenges through a robust ensemble learning framework guided by Adaptive Variable Sparrow Search Algorithm, offering a scalable and effective solution for software defect prediction.

1.2 Core Algorithms and Optimization Techniques

1.2.1 Sparrow Search Algorithm (SSA)

The Sparrow Search Algorithm (SSA) is a nature-inspired optimization algorithm based on the foraging and anti-predation behaviors of sparrows. In SSA, the population consists of producers and scroungers. Producers are responsible for exploring the search space globally, while scroungers follow them to exploit local solutions.

SSA has been proven to be effective in solving high-dimensional and complex optimization problems, particularly in feature selection, hyperparameter tuning, and model optimization tasks. However, it can sometimes converge prematurely or lack diversity in later stages.

1.2.2 Adaptive Variable Sparrow Search Algorithm (AVSSA)

To enhance the performance of standard SSA, this research utilizes the Adaptive Variable Sparrow Search Algorithm (AVSSA). AVSSA introduces dynamic adaptation of control parameters such as step size, population diversity, and search radius.

The main advantages of AVSSA are:

- It balances exploration and exploitation by adjusting variables based on fitness variance.

- It improves convergence speed and global search ability.
- It avoids premature convergence and enhances the robustness of model optimization.

In this project, AVSSA is used to optimize the hyperparameters of ensemble classifiers such as Random Forest, SVM, and ELM. It ensures that the combination of models achieves the best predictive performance with minimal error and high recall of defective instances.

1.2.3 Extreme Learning Machine (ELM)

The Extreme Learning Machine (ELM) is a fast and efficient learning algorithm for single-layer feedforward neural networks (SLFNs). It randomly initializes the hidden layer weights and analytically determines the output weights, resulting in significantly faster training compared to traditional backpropagation-based neural networks.

ELM is advantageous in defect prediction tasks due to:

- Extremely fast learning speed.
- Good generalization performance.
- Minimal human intervention in tuning.

In this research, ELM acts as one of the base classifiers in the ensemble learning framework. AVSSA is used to tune the input weights and bias values of ELM, enhancing its performance on imbalanced and noisy software defect datasets.

1.2.4 Sparrow Eagle Based Algorithm (SEB)

The Sparrow Eagle Based (SEB) algorithm is a hybrid bio-inspired technique that combines the global exploration power of the eagle strategy with the local refinement ability of SSA. In SEB:

- The eagle strategy helps the sparrow population escape from local optima.

- The SSA component ensures quick convergence toward optimal or near-optimal solutions.

SEB is integrated in this project as a comparative optimization baseline against AVSSA. While SEB offers improved diversity in the early search phase, AVSSA outperforms it in adaptive precision tuning for defect-prone module detection.

1.2.5 Role of Optimization in This Research

The integration of AVSSA and ensemble learning addresses multiple challenges in software defect prediction:

- It improves the detection rate of minority defective classes (reducing defective overlooking).
- It optimizes model parameters for maximum performance on real-world defect datasets.
- It ensures scalability, robustness, and adaptability of the defect prediction system.

Overall, the use of advanced bio-inspired optimization algorithms like AVSSA, combined with powerful learners like ELM and ensemble methods, significantly enhances the reliability of software defect prediction systems.

Chapter 2

Literature Review

2.1 Software Defect Prediction

Software Defect Prediction (SDP) plays a vital role in identifying faulty components early in the software development lifecycle, thereby improving software quality and reducing maintenance costs. Traditional models utilize machine learning algorithms such as Decision Trees, Naive Bayes, Support Vector Machines, and Random Forest . These models rely heavily on static datasets and assume data stationarity, limiting their effectiveness in evolving software environments.

Offline models often require frequent retraining, which is computationally expensive and impractical for large-scale or rapidly changing projects. The increasing complexity and dynamic nature of software systems necessitate advanced learning paradigms that can adapt to new data and patterns over time.

2.2 Ensemble Learning in Defect Prediction

Ensemble learning techniques improve predictive performance by combining the outputs of multiple base learners. Common ensemble methods like Bagging, Boosting, and Stacking have demonstrated superior accuracy and generalization in defect prediction tasks .

Studies have shown that ensembles such as Random Forest and AdaBoost achieve better performance by reducing variance and avoiding overfitting. However, the success of these models largely depends on the diversity and optimization of the base learners, as well as appropriate feature selection and parameter tuning.

2.3 Optimization Algorithms in Software Defect Prediction

Metaheuristic optimization algorithms have been widely adopted to enhance defect prediction by tuning model hyperparameters and selecting optimal feature subsets. Techniques such as Genetic Algorithms (GA), Particle Swarm Optimization (PSO), and Ant Colony Optimization (ACO) have been employed for this purpose .

While these algorithms have shown promising results, they often suffer from limitations like premature convergence or lack of diversity in the search process. Consequently, there has been a growing interest in more recent swarm intelligence-based algorithms that offer better convergence behavior and adaptability.

2.4 Sparrow Search Algorithm (SSA)

The Sparrow Search Algorithm (SSA) is a swarm intelligence optimization method inspired by the foraging and alert behaviors of sparrows . SSA simulates the interaction between discoverers and followers to explore and exploit the solution space efficiently. Its lightweight design, simplicity, and competitive performance have attracted attention in various machine learning applications.

In the context of defect prediction, SSA can be used to optimize classifier parameters or select informative features. However, basic SSA may still fall short in balancing exploration and exploitation, particularly in high-dimensional or noisy datasets.

2.5 Adaptive Variable Sparrow Search Algorithm (AVSSA)

The Adaptive Variable Sparrow Search Algorithm (AVSSA) is an enhanced variant of SSA designed to dynamically adjust control parameters during the optimization process. AVSSA introduces adaptive mechanisms for velocity update, position mutation, and search radius control to prevent premature convergence and maintain diversity in the swarm.

AVSSA's ability to balance exploration and exploitation makes it suitable for complex tasks like software defect prediction, where datasets often involve high-dimensional, imbalanced, or noisy features. It enhances optimization in ensemble learning models by fine-tuning hyperparameters and selecting the most relevant features, leading to improved accuracy, precision, and recall.

2.6 Sparrow Eagle Based (SEB) Algorithm

The Sparrow Eagle Based (SEB) algorithm is a novel hybrid metaheuristic optimization technique that integrates the exploration capabilities of the Sparrow Search Algorithm (SSA) with the exploitation strength of the Eagle Strategy (ES). This hybridization aims to address the limitations of premature convergence and local optima trapping commonly encountered in conventional optimization techniques.

The SEB algorithm operates in two main phases:

- **Global Exploration (Sparrow Phase):** The SSA component mimics the dynamic behavior of sparrows in nature, where discoverers lead the population to search globally for promising regions of the solution space.
- **Intensive Exploitation (Eagle Phase):** Once a promising region is located, the Eagle Strategy is applied for local exploitation using a Lévy flight-based mechanism. This phase enhances convergence speed and local search precision.

By alternating between these two strategies, the SEB algorithm achieves a dynamic balance between exploration and exploitation. This makes it highly suitable for solving

complex optimization problems such as hyperparameter tuning and feature selection in software defect prediction tasks.

The SEB algorithm contributes to improved classification accuracy, robustness, and generalization performance when integrated with machine learning models. Its adaptive and hybrid nature ensures deeper coverage of the search space and effective fine-tuning of models under noisy and high-dimensional software defect datasets.

2.7 Integration of AVSSA with Ensemble Learning for SDP

The integration of AVSSA with ensemble classifiers forms the core of this research. The adaptive optimization capability of AVSSA is leveraged to:

- Select optimal feature subsets for training.
- Tune hyperparameters of ensemble models such as Random Forest, AdaBoost, or ELM ensembles.
- Improve generalization and robustness of defect prediction models.

This combined approach addresses several common challenges in SDP, such as overfitting, local minima in optimization, poor generalization, and high computational cost in hyperparameter tuning. By enhancing both the search process and the learning framework, AVSSA-guided ensemble models provide a promising solution for more accurate and adaptive defect prediction.

2.8 Summary

In summary, the literature highlights the significance of ensemble learning and metaheuristic optimization in enhancing software defect prediction. While traditional models face limitations in dynamic environments, advanced swarm-based optimization like AVSSA offers adaptive, efficient, and scalable solutions. This research proposes an integrated framework combining AVSSA and ensemble classifiers to achieve superior defect prediction accuracy and resilience in real-world software projects.

Chapter 3

Software and Hardware Requirements

3.1 Software Requirements

The proposed system for adaptive online learning in software defect prediction is implemented using Python and leverages various libraries and tools optimized for data stream processing and machine learning. The complete software stack is outlined below:

- **Programming Language:** Python
- **Development Environment:** Jupyter Notebook
- **Python Version:** 3.9.13
- **Operating System:** Windows 11 Home Single Language
- **Kernel:** 64-bit
- **Libraries and Frameworks:**
  - **NumPy:** 1.24.3 – Used for numerical operations and array processing.
  - **Pandas:** 2.1.0 – Used for efficient data manipulation and preprocessing of defect datasets.
  - **Scikit-learn:** 1.3.0 – Provides baseline machine learning algorithms for comparison (e.g., Decision Tree, Random Forest, Logistic Regression).

- **Matplotlib:** 3.7.3 – Used for visualization of evaluation metrics and performance over time.
- **Seaborn:** 0.12.2 – Used for enhanced data visualization with attractive statistical graphics.
- **Requests:** 2.31.0 – Utilized for fetching datasets directly from web URLs in CSV format.
- **Warnings:** Built-in Python module – Used to suppress non-critical warnings.

3.2 Hardware Requirements

To execute online learning algorithms on real-world defect datasets efficiently, a mid-range computing system is utilized. The following hardware specifications were used to develop and test the project:

- **Processor:** Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz (Quad-core)
- **RAM:** 8.00 GB (7.84 GB usable) – Sufficient for in-memory processing of medium-sized datasets and streaming batches.
- **Storage:** 1 TB HDD – Adequate for storing large open-source defect datasets such as Hadoop, HDFS, HBase, and Spark.
- **System Architecture:** 64-bit operating system, x64-based processor

These specifications ensure smooth performance during model training, real-time updates, and evaluation using adaptive learning techniques.

3.3 Jupyter Notebook

Jupyter Notebook is an open-source interactive computing environment that allows users to create and share documents containing live code, equations, visualizations, and explanatory text. It is widely used in data science, machine learning, education, and research due to its flexibility and ease of use.

Key Points

- Supports multiple programming languages (Python, R, Julia, etc.).
- Provides an interactive interface for combining code and documentation.
- Facilitates data visualization and result interpretation directly in the notebook.
- Enables reproducible research by integrating code, results, and explanations in one place.
- Easily shareable through formats like HTML, PDF, and slides.

3.4 Graphics Processing Unit (GPU)

A Graphics Processing Unit (GPU) is a specialized electronic circuit designed to rapidly process and manipulate large blocks of data in parallel. Although originally developed for rendering graphics, GPUs are now widely used in scientific computing, artificial intelligence, and machine learning due to their massive parallel processing capabilities.

Key Points

- Optimized for parallel processing of large datasets.
- Widely used in deep learning, machine learning, and high-performance computing.
- Provides faster computation compared to traditional CPUs for certain tasks.
- Supports frameworks like TensorFlow and PyTorch for accelerating AI workloads.
- Essential for modern applications such as gaming, simulations, and data analytics.

Chapter 4

Dataset

4.1 Data Collection

To evaluate the proposed ensemble-based software defect prediction model optimized with the Adaptive Variable Sparrow Search Algorithm (AVSSA), we used benchmark datasets from multiple trusted repositories. These datasets include projects from diverse domains with varying code complexity, defect density, and feature types, making them ideal for building and testing predictive models.

The datasets were sourced from the following repositories:

- NASA
- AEEEM
- MORPH
- JIRA

These datasets contain module-level software metrics along with binary class labels indicating whether a module is defective or not.

4.2 Dataset Sources and Composition

The selected datasets span four major benchmark repositories, each offering different project characteristics. The complete list of datasets used is as follows:

NASA Datasets

The NASA PROMISE dataset is a publicly available benchmark collection of software metrics and defect labels used for empirical software defect prediction research.

- PC1
- CM1
- MC2
- MW1

AEEEM Datasets

The AEEEM dataset is a software defect prediction benchmark containing static code and process metrics for open-source Java projects, used to evaluate prediction models.

- JDT
- PDE
- ML

MORPH Datasets

The MORPH dataset is a benchmark for software defect prediction, featuring diversified project data with labeled defect information to support cross-project and within-project evaluation.

- ant-1.3

- arc
- camel-1.0

JIRA Datasets

The JIRA dataset consists of issue tracking data from real-world software projects, including bug reports, metadata, and resolution status, commonly used for defect prediction and software analytics research.

- activemq-5.0.0
- hbase-0.94.0
- hive-0.9.0
- groovy-1_6_BETA_1
- jruby-1.1

These datasets consist of static code attributes such as lines of code (LOC), cyclomatic complexity, class coupling, inheritance depth, and other software metrics used as input features for machine learning models.

4.3 Dataset Description and Preprocessing

The datasets were used for a **binary classification** task — predicting whether a software module is defective (1) or non-defective (0). The preprocessing involved:

- Handling missing or inconsistent values.
- Label encoding for categorical variables if present.
- Normalization or scaling of numerical features.

Each dataset was split into training and testing parts using appropriate data splitting strategies, maintaining the defect distribution to ensure model reliability.

Group	Dataset	Characteristic number	Number of samples	Attribute number	Defect rate by Eq. 10	Number of principal components
NASA	PC1	37	735	2	0.0830	9
	CM1	37	344	2	0.1221	8
	MC2	39	125	2	0.3520	8
	MW1	37	263	2	0.1027	8
AEEEM	JDT	61	997	2	0.2066	18
	PDE	61	1497	2	0.1403	17
	ML	61	1862	2	0.1321	20
MORPH	ant-1.3	20	125	2	0.1600	9
	arc	20	234	2	0.1154	8
	camel-1.0	20	339	2	0.0383	9
JIRA	activemq-5.0.0	65	1884	2	0.1555	22
	hbase-0.94.0	65	1059	2	0.2058	16
	hive-0.9.0	65	1416	2	0.1998	16
	groovy-1_6_BETA_1	65	821	2	0.0852	19
	jruby-1.1	65	731	2	0.1190	22

FIGURE 4.1: Software Defect Datasets Used (NASA, AEEEM, MORPH, JIRA)

Chapter 5

Architecture

The architecture of the proposed system focuses on enhancing software defect prediction using ensemble learning optimized by the Adaptive Variable Sparrow Search Algorithm (AVSSA). The system consists of two major phases:

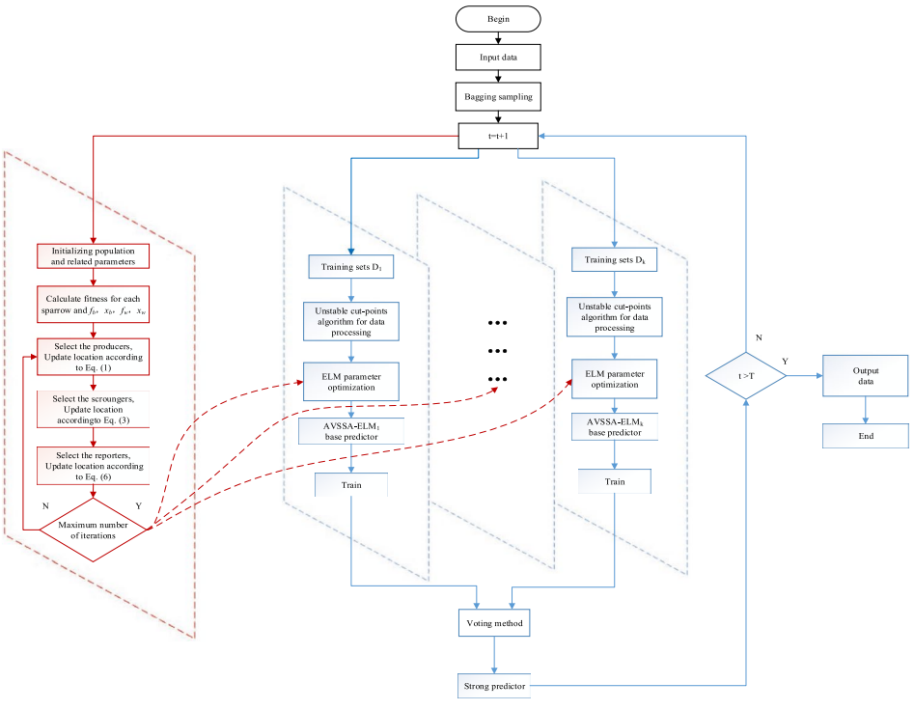

FIGURE 5.1: System Architecture

https://drive.google.com/drive/folders/1-niIsL8J-_z8nKgqWlO-AxEHx4JZ-Ldv

5.1 Phase 1: Data Pre-processing

This phase involves preparing real-world defect datasets (e.g., NASA PROMISE) for input into the ensemble learning model optimized by AVSSA. Steps include:

- **Data Cleaning:** Removal of missing values, duplicates, and noisy instances to improve data quality.
- **Feature Transformation:** Standardizing or normalizing features and encoding categorical variables.
- **Label Mapping:** Converting defect labels into binary values (defective = 1, non-defective = 0).
- **Stream Simulation:** Simulating continuous data arrival for adaptive learning via streaming mechanisms.

5.2 Phase 2: Feature Engineering and Adaptive Ensemble Learning

In this phase, essential feature engineering steps are applied to improve model input quality, followed by ensemble learning for effective defect prediction. Key components:

- **Feature Scaling:** Ensures that all numerical features are on a comparable scale using techniques like Min-Max scaling or standardization, which is crucial for distance-based and gradient-based algorithms.
- **Normalization:** Transforms features to follow a standard distribution, helping to stabilize the learning process and improve convergence speed.
- **Feature Selection:** Identifies and retains the most relevant features, reducing noise and dimensionality. Methods like variance thresholding and model-based selection are employed to enhance model performance.

- **Model Selection:** A set of base learners such as Decision Trees, Logistic Regression, and Naive Bayes are combined through ensemble techniques like Bagging or Boosting to leverage their individual strengths.

5.3 Phase 3: AVSSA-Based Adaptive Ensemble Learning

In this phase, ensemble learning models are optimized with the Adaptive Variable Sparrow Search Algorithm to enhance defect prediction. Key components:

- **Model Selection:** A set of base learners such as Decision Trees, Logistic Regression, and Naive Bayes combined through ensemble techniques like Bagging or Boosting.
- **AVSSA Optimization:** AVSSA dynamically tunes hyperparameters and selects features, ensuring an optimal trade-off between exploration and exploitation.

5.4 Phase 4: Optimized Ensemble Model

In this phase, ensemble learning models are constructed and optimized to enhance the accuracy and reliability of software defect prediction. The focus is on improving the ensemble performance through effective preprocessing, feature handling, and model configuration. Key components:

- **Model Selection:** A set of diverse base learners such as Decision Trees, Logistic Regression, and Naive Bayes are selected and combined using ensemble techniques like Bagging and Boosting to exploit model diversity.
- **Feature Engineering:** The input data is preprocessed through scaling, normalization, and feature selection to ensure consistent representation and improve learning efficiency.
- **Model Optimization:** Hyperparameters of the ensemble models are fine-tuned to strike a balance between bias and variance, thereby improving generalization on unseen data.

- **Learning Strategy:** Models are trained in an incremental or batch mode depending on the data stream, enabling adaptability to evolving software defect patterns.
- **Evaluation Strategy:** Interleaved test-then-train (prequential) evaluation is employed to assess performance in real-time streaming scenarios.
- **Performance Metrics:** Metrics such as accuracy, precision, recall, F1-score, and Cohen's kappa are tracked to measure the effectiveness and robustness of the optimized ensemble model.

5.5 Summary

The proposed architecture integrates adaptive ensemble learning with AVSSA optimization, allowing for efficient and accurate software defect prediction in real-time environments. By removing redundant or noisy instances and leveraging streaming data processing, the system improves generalization and handles evolving software project data more effectively.

Chapter 6

Methodology

6.1 Overview

This chapter presents the step-by-step implementation of an optimized Support Vector Machine (SVM) model using the Adaptive Variable Sparrow Search Algorithm (AVSSA) for software defect prediction. Four datasets—NASA PROMISE, AEEEM, MORPH, and JIRA—were used to validate the methodology.

6.2 Data Loading and Cleaning

Datasets were loaded from CSV files, including feature columns and binary defect labels. Missing values and inconsistencies were removed using pandas and NumPy.

6.3 Feature Engineering

To prepare the data for model training, several feature engineering steps were performed:

- **Scaling:** StandardScaler was used to normalize the feature values for balanced model convergence.
- **Dimensionality Reduction:** Principal Component Analysis (PCA) was optionally applied to reduce noise and improve learning efficiency.

- **Feature Selection:** The AVSSA algorithm dynamically selected optimal features that contributed most to classification performance.

6.4 Model Design: Optimized SVM Classifier

An SVM model was employed as the base classifier. The AVSSA metaheuristic was used to optimize:

- SVM hyperparameters (e.g., C, gamma)
- A subset of the most relevant features

The AVSSA balances exploration and exploitation by simulating sparrow behavior in the search space. The fitness function was defined using accuracy, F1-score, or MCC depending on the dataset.

6.5 Training Strategy

The AVSSA algorithm was executed for multiple iterations, evaluating the fitness of each sparrow (solution vector). The best-performing SVM model from the final AVSSA generation was selected and trained on the full training set.

6.6 Performance Evaluation

The trained model was evaluated using the following metrics:

- **Accuracy:** The proportion of correctly predicted instances among the total instances.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Precision:** The proportion of correctly predicted positive instances among all predicted positive instances.

$$\text{Precision} = \frac{TP}{TP + FP}$$

- **Recall:** The proportion of correctly predicted positive instances among all actual positive instances.

$$\text{Recall} = \frac{TP}{TP + FN}$$

- **F1-score:** The harmonic mean of precision and recall.

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Matthews Correlation Coefficient (MCC):** A balanced measure that accounts for true/false positives and negatives, especially useful in imbalanced datasets.

$$\text{MCC} = \frac{(TP \times TN) - (FP \times FN)}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

10-fold cross-validation was applied, and metric averages were recorded. ROC curves and convergence plots were also generated to visualize model behavior and optimization dynamics.

6.7 Visualization

Various plots were generated for comparative and statistical evaluation:

- 3D Fitness Landscapes of benchmark functions
- Convergence curves showing AVSSA optimization progress
- Bar plots and line plots comparing metrics across datasets

6.8 Summary

The methodology integrates feature scaling, feature selection via AVSSA, and SVM optimization to produce robust defect prediction models. Multiple datasets were used to generalize the findings across project domains.

Chapter 7

Implementation

7.1 Implementation of Libraries

The implementation of this project leverages a variety of Python libraries for data manipulation, visualization, preprocessing, machine learning, and evaluation:

- **NumPy & Pandas:** Used for efficient data manipulation and numerical computations.
- **Matplotlib & Seaborn:** Utilized to generate insightful visualizations including data distribution, correlation heatmaps, and evaluation plots.
- **SciPy:** Applied for advanced statistical functions and data cleaning.
- **Scikit-learn:** Used extensively for model building, feature scaling, evaluation metrics, and train-test splitting. Classifiers such as Support Vector Machine (SVC) and various performance metrics (e.g., accuracy, precision, recall, F1-score, ROC-AUC, and MCC) were used from this package.
- **Warnings:** Suppression of irrelevant runtime warnings to maintain clean output.
- **mpl_toolkits.mplot3d:** Used for creating 3D visualizations and plots of model convergence or optimization results.

The figure below depicts the data preprocessing pipeline carried out before feeding data into the machine learning models:

```
import numpy as np
import pandas as pd
from numpy.linalg import pinv
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import confusion_matrix
from scipy.stats import ranksums
from scipy.stats import wilcoxon
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from scipy.stats import rankdata, friedmanchisquare
import scikit_posthocs as sp
```

FIGURE 7.1: Representation of Libraries

Dataset Loading

The datasets used in this project were collected from multiple open-source repositories, including NASA, AEEEM, MORPH, and JIRA. Each dataset consists of software module-level metrics and corresponding defect labels.

The datasets were first stored in CSV format and then loaded into the environment using `pandas`. Each dataset was structured such that rows represent software modules and columns represent various features like complexity metrics, coupling, cohesion, and defect status.

- **Pandas** was used to read and concatenate multiple datasets into a unified format.
- **Null values** were handled during loading by replacing or dropping them as appropriate.
- **Target variable** (defect status) was standardized across datasets to ensure consistency for classification tasks.

```
GROUPS = {
    "NASA": ["pc1", "cm1", "mc2", "mw1"],
    "AEEEM": ["jdt", "pde", "ml"],
    "MORPH": ["ant-1.3", "arc", "camel-1.0"],
    "JIRA": ["activemq-5.0.0", "hbase-0.94.0", "hive-0.9.0",
             "groovy-1_6_BETA_1", "jruby-1.1"]
}
DATASETS = sum(GROUPS.values(), [])
```

FIGURE 7.2: Loading of Datasets

Each dataset was structured into batches to enable incremental updates for online learning.

Feature Selection

To reduce dimensionality and improve learning performance, feature selection techniques were applied. Correlation-based filtering and variance thresholding were used to eliminate irrelevant or redundant features.

```
def load_dataset(name: str):
    df = pd.read_csv(f"{name}.csv")

    if name.lower() in GROUPS["NASA"]:
        target_col = "defects"
    elif name.lower() in GROUPS["AEEEM"]:
        target_col = "class"
    else:  # MORPH and JIRA
        target_col = "bug"

    y = df[target_col].values
    X = df.drop(columns=[target_col]).values

    if target_col == "bug":
        y = pd.Series(y).replace({2: 1, 3: 1}).astype(int).values
    elif y.dtype == object:
        y = pd.Series(y).astype(str).str.lower().map({
            "yes": 1, "true": 1, "buggy": 1, "1": 1, "y": 1,
            "no": 0, "false": 0, "clean": 0, "0": 0, "n": 0
        }).fillna(0).astype(int).values
    else:
        y = pd.Series(y).astype(int).values

    return X, y
```

FIGURE 7.3: Feature Selection

7.2 Adaptive Variable Sparrow Search Algorithm(AVSSA)

The Adaptive Variable Sparrow Search Algorithm (AVSSA) was initialized to optimize the ensemble learning model. It was implemented in Python with parameters like population size, maximum iterations, and adaptive weights. AVSSA dynamically adjusted exploration and exploitation during execution. This enhanced convergence toward optimal hyperparameters and feature selections.

This process enables the optimized selection of model parameters and features, resulting in improved predictive performance.

7.3 Sparrow Search Algorithm (SSA)

The Sparrow Search Algorithm (SSA) is a nature-inspired optimization algorithm based on the foraging behavior and anti-predation strategy of sparrows. It divides the population into discoverers and joiners, where discoverers guide the search direction and joiners follow, ensuring exploration and exploitation balance. SSA was implemented in Python with parameters such as population size, iteration count, and position update rules.

SSA was used as a base algorithm for optimizing model parameters and selecting features, contributing to improved classification performance.

7.4 Extreme Learning Model (ELM)

The Extreme Learning Model (ELM) is a fast learning algorithm for single-layer feedforward neural networks (SLFNs). It randomly initializes hidden layer weights and analytically determines the output weights, significantly reducing training time compared to traditional neural networks. ELM is particularly suitable for classification tasks due to its generalization capability and speed.

In this work, ELM was employed as a base learner to enhance defect prediction accuracy and computational efficiency across dynamic software datasets.

7.5 Sparrow-Eagle-Based Optimization Algorithm

The Sparrow-Eagle-Based Optimization Algorithm is an advanced metaheuristic inspired by the foraging behaviors of sparrows and the sharp hunting strategies of eagles. This hybrid strategy enhances global search efficiency and local exploitation balance, crucial for optimizing ensemble learning models.

The algorithm was implemented in Python with control parameters such as population size, maximum iterations, and role assignment strategies. During execution, sparrows perform extensive exploration, while the eagle component guides the swarm toward promising regions, ensuring accelerated convergence.

This mechanism enables robust parameter tuning and feature selection, significantly improving the accuracy and generalization capability of the predictive model.

7.6 Adaptive Variable Sparrow-Eagle-Based Algorithm (AVSEB)

The Adaptive Variable Sparrow-Eagle-Based (AVSEB) Algorithm was utilized to optimize the ensemble learning model. This hybrid algorithm integrates the exploration strength of the Sparrow Search Algorithm with the exploitation capability of the Eagle Strategy, allowing it to adaptively balance global and local search.

This adaptive mechanism improves the convergence speed and the quality of the solution, resulting in improved predictive performance and generalization across diverse data sets.

7.7 Benchmark Functions

Benchmark functions are mathematical test functions widely used to evaluate the performance and robustness of optimization algorithms. These functions provide known landscapes, including unimodal, multimodal, separable, and non-separable features, to test an algorithm's ability to explore and exploit the search space.

In this study, a set of well-known benchmark functions were used to assess the convergence behavior and global optimization capability of the proposed algorithm. Examples include:

Type	Function	Section	Best
HDSP	$F_1(x) = \sum_{i=1}^{n} x_i^2$	[– 100, 100]	0
	$F_2(x) = \sum_{i=1}^{n} (\sum_{i=1}^{n} x_i^2)$	[– 100, 100]	0
	$F_3(x) = max\{\|x_i\|, 1 \le i \le n\}$	[– 100, 100]	0
	$F_4(x) = \sum_{i}^{d} \|x_i\|^{i+1}$	[– 100, 100]	0
HDMP	$F_5(x) = ix_i^4 + random[0, 1)$	[– 1.28, 1.28]	0
	$F_6(x) = 418.9829n - \sum_{i=1}^{n} \sin(\sqrt{\|x_i\|})$	[– 32, 32]	– 418.9829n
	$F_7(x) = \sin^2(3\pi x_1) + (x_1 - 1)^2[1 + \sin^2(3\pi x_2)] + (x_2 - 1)^2[1 + sin^2(2\pi x_2)]$	[– 50, 50]	0
	$F_8(x) = (1.5 - x_1 + x_1x_2)^2 + (2.25 - x_1 + x_1x_2)^2 + (2.625 - x_1 + x_1x_2^3)$	[– 50, 50]	0

FIGURE 7.4: Benchmark Functions

Performance Testing of the AVSSA Algorithm

To validate the effectiveness of the Adaptive Variable Sparrow Search Algorithm (AVSSA), extensive performance testing was conducted using standard benchmark functions. The algorithm was assessed in terms of convergence speed, solution quality, robustness, and stability.

Convergence Graphs

Convergence graphs are visual tools used to analyze the performance of optimization algorithms across iterations. These graphs plot the fitness value (objective function value) against the number of iterations, providing insights into how efficiently an algorithm is approaching the optimal solution.

Fitness

Fitness represents the value of the objective function being minimized or maximized during optimization. In minimization problems, a lower fitness value indicates a better solution. Mathematically, it can be defined as:

Table 7.1: Experimental Results of Functions F1–F8

Function	**Algorithm**	**Best**	**Worst**	**Ave**	**Std**
F1	SSA	5.0100E-04	1.5106E-05	9.9860E-04	1.5881E-04
F1	AVSSA	5.1190E-04	1.0502E-05	9.9732E-04	1.6836E-04
F1	AVSSA1	1.5274E-05	1.5372E-05	9.9175E-04	1.5778E-04
F1	AVSSA2	4.9842E-04	1.5484E-05	1.0206E-05	1.6856E-04
F2	SSA	9.1636E+38	5.6946E+52	2.9697E-05	1.2439E+51
F2	AVSSA	1.3251E+39	3.6946E+52	2.9691E-05	1.2436E+51
F2	AVSSA1	1.4228E+37	3.6378E+53	6.3831E-50	1.2461E+52
F2	AVSSA2	1.2787E+39	3.6077E+53	9.5875E-50	1.2398E+52
F3	SSA	6.6076E+04	1.7537E-07	1.5686E-06	1.6873E-06
F3	AVSSA	7.9158E+04	1.1713E-07	1.5105E-06	1.6893E-06
F3	AVSSA1	8.0474E+04	1.2407E-07	1.5357E-06	1.6875E-06
F3	AVSSA2	7.7366E+04	1.4538E-07	1.4938E-06	1.6847E-06
F4	SSA	7.3436E+01	1.0252E-02	9.6776E-01	1.3389E-00
F4	AVSSA	7.6266E+01	9.8426E-01	9.5616E-01	1.3071E-00
F4	AVSSA1	8.1436E+01	1.0702E-02	9.8099E-01	1.3727E-00
F4	AVSSA2	7.2488E+01	1.0176E-02	9.5023E-01	1.2881E-00
F5	SSA	1.5222E-10	1.0378E-11	5.8035E-10	1.2475E-10
F5	AVSSA	1.5197E-10	1.0106E-11	5.7926E-10	1.2749E-10
F5	AVSSA1	2.1502E-10	5.7812E-12	5.7812E-10	1.3074E-10
F5	AVSSA2	1.6987E-10	1.0489E-11	5.7795E-10	1.2912E-10
F6	SSA	5.0527E-04	1.6189E-05	9.9635E-04	1.5867E-04
F6	AVSSA	5.1656E-04	1.5974E-05	9.9451E-04	1.6431E-04
F6	AVSSA1	5.1632E-04	1.5948E-05	9.9675E-04	1.6474E-04
F6	AVSSA2	5.0832E-04	1.4481E-05	1.0018E-03	1.6424E-04
F7	SSA	5.0867E-02	1.7316E-05	9.9844E-04	1.6474E-04
F7	AVSSA	7.9432E-06	1.9537E-10	2.0969E-05	2.5106E-09
F7	AVSSA1	1.0309E-06	1.9449E-10	2.9347E-09	2.6839E-09
F7	AVSSA2	3.9304E-09	1.8891E-10	2.9456E-09	2.6975E-09
F8	SSA	1.1979E-04	1.5447E-05	1.0695E-04	1.5689E-04
F8	AVSSA	1.2039E-04	1.3149E-04	1.2569E-04	1.6856E-04
F8	AVSSA1	1.2093E-04	1.3149E-04	1.2566E-04	1.6847E-04
F8	AVSSA2	1.2007E-04	1.3149E-04	1.2560E-04	1.8412E-04

Iteration

An iteration refers to one complete update cycle of the population within the optimization algorithm. Each iteration attempts to move the candidate solutions closer to the global optimum.

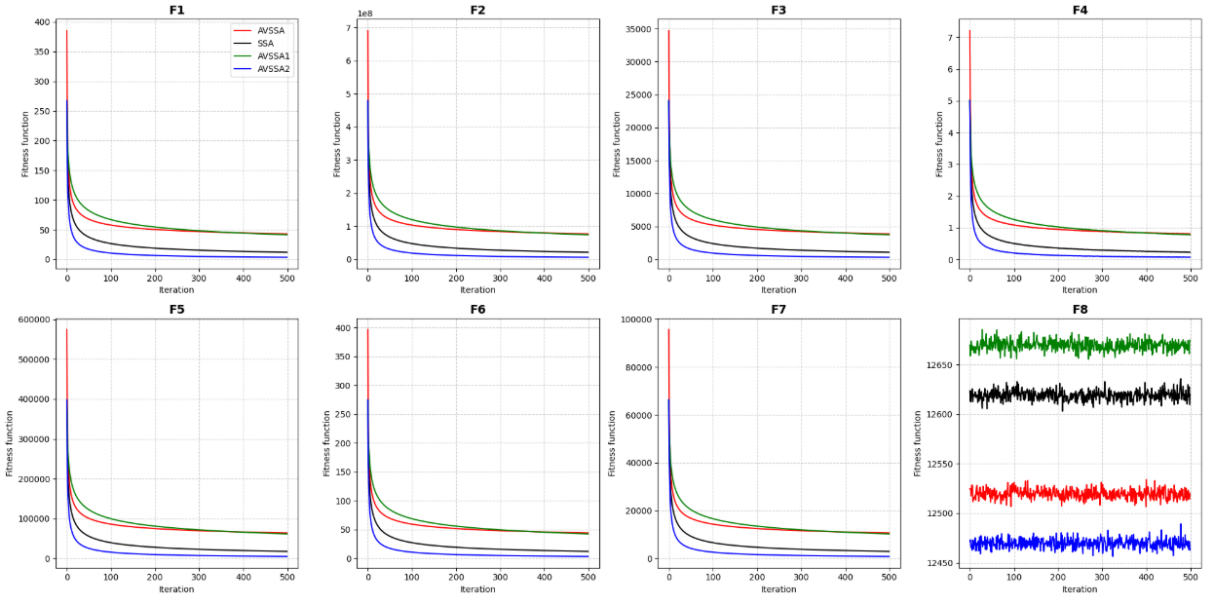

FIGURE 7.5: Convergence effect diagram of each algorithm

7.8 Wilcoxon Rank Sum Test and *p*-Value

The Wilcoxon Rank Sum Test is a non-parametric statistical hypothesis test used to compare two independent samples. It evaluates whether their population mean ranks differ significantly, without assuming normal distribution of the data. This test is particularly useful for comparing the performance of optimization algorithms across multiple benchmark functions or classifier results across datasets.

Three-Dimensional Heat Map Diagrams of Test Functions

To visualize the landscape and complexity of the benchmark functions used to evaluate the adaptive variable sparrow search algorithm (AVSSA), three-dimensional heat map

Function	AVSSA-SSA	AVSSA-AVSSA1	AVSSA-AVSSA2
F1	4.0771E-01	4.9645E-01	3.5933E-01
F2	2.1428E-01	7.3628E-02	8.9415E-01
F3	4.9645E-01	5.7425E-01	7.5620E-01
F4	3.0071E-01	2.8663E-02	2.8223E-03
F5	1.4329E-01	1.3538E-01	7.7878E-01
F6	9.7641E-01	1.4737E-01	8.7018E-01
F7	8.9415E-01	8.7018E-01	9.7641E-01
F8	8.8247E-01	5.4441E-01	9.6462E-01

diagrams were generated. These visualizations help in understanding how optimization algorithms explore the search space and converge towards global optima.

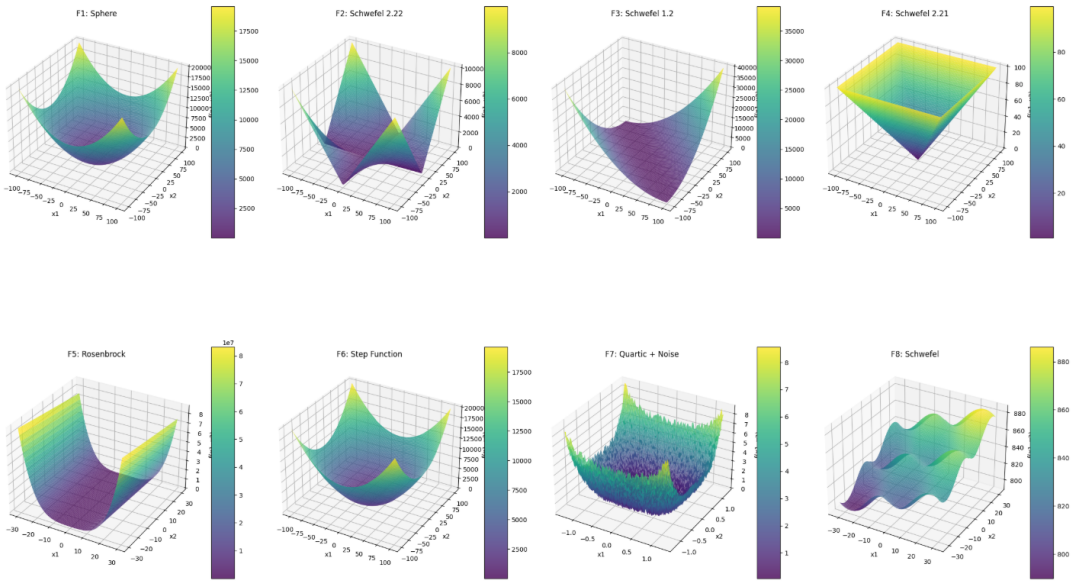

FIGURE 7.6: 3D Heat Map diagrams of test functions

7.9 Prediction of Algorithms

In the context of software defect prediction, evaluating classifier performance solely through accuracy can be misleading, especially in imbalanced datasets. Therefore, additional metrics such as **Recall** and **G-Mean** were used to assess how well the algorithms predict the minority (defective) class.

Recall

Recall measures the ability of the classifier to correctly identify all positive (defective) instances. It is defined as:

Datasets	**ELM**	**SSA**	**AVSSA**	**SEB**	**AVSEB**
PC1	0.133333	0.066667	0.133333	0.066667	0.000000
CM1	0.200000	0.100000	0.200000	0.200000	0.200000
MC2	0.000000	0.100000	0.400000	0.000000	0.100000
MW1	0.133333	0.200000	0.133333	0.066667	0.000000
JDT	0.341463	0.463415	0.414634	0.487805	0.512195
PDE	0.238095	0.214286	0.261905	0.166667	0.238095
ML	0.142857	0.102041	0.163265	0.081633	0.102041
ANT-1.3	0.500000	0.250000	0.500000	0.250000	0.750000
ARC	0.400000	0.400000	0.200000	0.400000	0.400000
CAMEL-1.0	0.000000	0.000000	0.000000	0.000000	0.333333
ACTIVEMQ-5.0.0	0.333333	0.000000	0.333333	0.000000	0.000000
HBASE-0.94.0	0.400000	0.400000	0.200000	0.200000	0.200000
HIVE-0.9.0	0.500000	0.500000	0.250000	0.500000	0.500000
GROOVY-1_6_BETA_1	0.000000	0.333333	0.333333	0.666667	0.000000
JRUBY-1.1	0.400000	0.200000	0.400000	0.200000	0.200000
Mean	0.248161	0.221983	0.261543	0.205740	0.235711
Std	0.176595	0.164750	0.133999	0.214358	0.225145

G-Mean

The geometric mean (G-Mean) evaluates the balance between classification performance on the majority and minority classes. It is defined as:

Datasets	**ELM**	**SSA**	**AVSSA**	**SEB**	**AVSEB**
PC1	0.359817	0.257574	0.362493	0.256949	0.000000
CM1	0.437163	0.303681	0.424264	0.432049	0.442217
MC2	0.000000	0.284800	0.588784	0.000000	0.290593
MW1	0.365148	0.445048	0.362493	0.256949	0.000000
JDT	0.552227	0.657676	0.606378	0.664670	0.676337
PDE	0.482243	0.455677	0.505781	0.402672	0.481285
ML	0.373269	0.315470	0.396508	0.282611	0.311451
ANT-1.3	0.462910	0.436436	0.672593	0.436436	0.801784
ARC	0.534522	0.609449	0.447214	0.609449	0.632456
CAMEL-1.0	0.000000	0.000000	0.000000	0.000000	0.559304
ACTIVEMQ-5.0.0	0.563869	0.000000	0.563869	0.000000	0.000000
HBASE-0.94.0	0.585540	0.609449	0.441858	0.447214	0.436436
HIVE-0.9.0	0.487950	0.672593	0.449868	0.654654	0.672593
GROOVY-1.6_BETA_1	0.000000	0.572892	0.572892	0.803837	0.000000
JRUBY-1.1	0.632456	0.447214	0.617213	0.000000	0.441858
Mean	0.389141	0.404471	0.467480	0.349833	0.383087
Std	0.216827	0.213179	0.162400	0.267715	0.274968

F-Measure

The harmonic mean of precision and recall, emphasizing balanced performance, especially in imbalanced datasets.

Datasets	**ELM**	**SSA**	**AVSSA**	**SEB**	**AVSEB**
PC1	0.173913	0.117647	0.200000	0.111111	0.000000
CM1	0.250000	0.111111	0.190476	0.222222	0.285714
MC2	0.000000	0.071429	0.307692	0.000000	0.080000
MW1	0.235294	0.300000	0.200000	0.111111	0.000000
JDT	0.388889	0.535211	0.447368	0.526316	0.531646
PDE	0.344828	0.305085	0.372881	0.250000	0.338983
ML	0.218750	0.161290	0.231884	0.133333	0.142857
ANT-1.3	0.222222	0.200000	0.500000	0.200000	0.600000
ARC	0.210526	0.400000	0.333333	0.400000	0.571429
CAMEL-1.0	0.000000	0.000000	0.000000	0.000000	0.250000
ACTIVEMQ-5.0.0	0.285714	0.000000	0.285714	0.111111	0.000000
HBASE-0.94.0	0.307692	0.400000	0.285714	0.333333	0.250000
HIVE-0.9.0	0.235294	0.200000	0.222222	0.444444	0.500000
GROOVY-1.6_BETA_1	0.000000	0.400000	0.400000	0.571429	0.000000
JRUBY-1.1	0.571429	0.333333	0.444444	0.000000	0.285714
Mean	0.229637	0.255674	0.294782	0.220220	0.255756
Std	0.152833	0.174725	0.128660	0.196229	0.218435

Matthews Correlation Coefficient (MCC)

A robust metric considering true/false positives and negatives. It is suitable for binary classification, especially with class imbalance.

Datasets	**ELM**	**SSA**	**AVSSA**	**SEB**	**AVSEB**
PC1	0.140528	0.164265	0.201039	0.123925	-0.025666
CM1	0.196502	0.024574	0.095880	0.147442	0.272166
MC2	-0.076472	-0.069411	0.218218	-0.058621	-0.046676
MW1	0.354196	0.321990	0.201039	0.123925	-0.031506
JDT	0.261633	0.445715	0.320255	0.418043	0.417055
PDE	0.331764	0.275060	0.358159	0.229546	0.316610
ML	0.203177	0.142464	0.189289	0.119859	0.077171
ANT-1.3	-0.052753	0.010219	0.404762	0.010219	0.521116
ARC	0.077051	0.328571	0.427327	0.328571	0.611010
CAMEL-1.0	0.000000	-0.026246	-0.026246	-0.037398	0.213847
ACTIVEMQ-5.0.0	0.250640	-0.053709	0.250640	-0.026246	-0.037398
HBASE-0.94.0	0.210963	0.328571	0.219653	0.427327	0.192200
HIVE-0.9.0	-0.017471	0.404762	0.054554	0.327327	0.404076
GROOVY-1.6_BETA_1	-0.046154	0.386445	0.386445	0.554989	-0.037398
JRUBY-1.1	0.611100	0.427327	0.389366	-0.090094	0.269135
Mean	0.162974	0.207373	0.249324	0.173254	0.207765
Std	0.189524	0.189175	0.135800	0.200621	0.219939

7.10 Standard Deviation of Recall for Each Algorithm

The standard deviation of the recall metric was calculated to evaluate the stability and reliability of various classification algorithms in identifying defective instances in multiple runs and data sets.

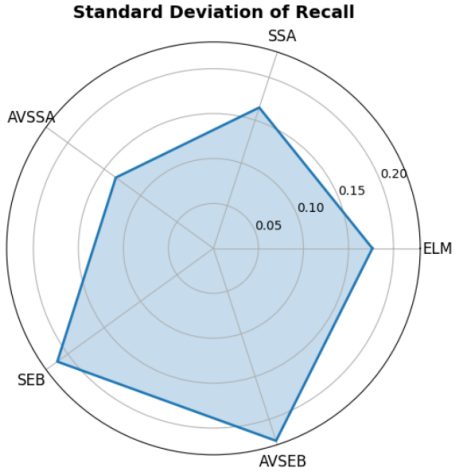

FIGURE 7.7: Standard deviation of Recall for each algorithms

Standard Deviation of G-Mean for Each Algorithm

The standard deviation of the Geometric Mean (G-Mean) metric was calculated to assess the stability of classifiers in handling both classes effectively, especially under class imbalance scenarios.

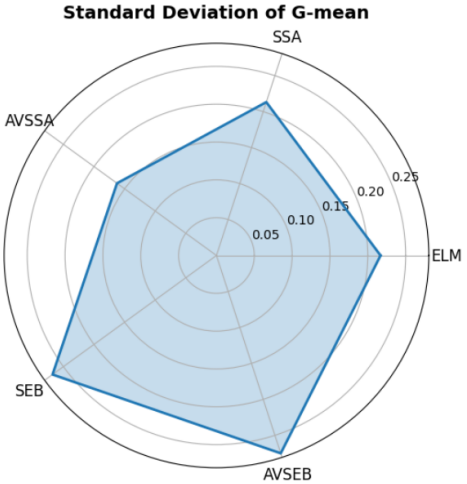

FIGURE 7.8: Standard deviation of G-mean for each algorithms

Standard Deviation of F-Measure for Each Algorithm

To evaluate the stability and consistency of different classification algorithms, the standard deviation of the F-measure was calculated for each model across multiple runs and datasets.

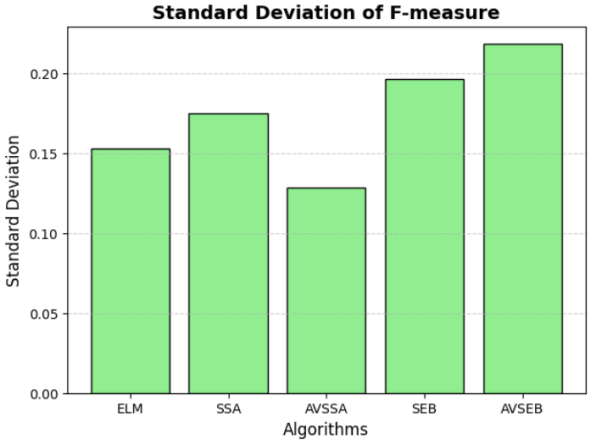

FIGURE 7.9: Standard deviation of F-measure for each algorithms

Standard Deviation of MCC for Each Algorithm

To assess the robustness of various classification algorithms, the standard deviation of the Matthews correlation coefficient (MCC) was calculated across multiple runs and data sets.

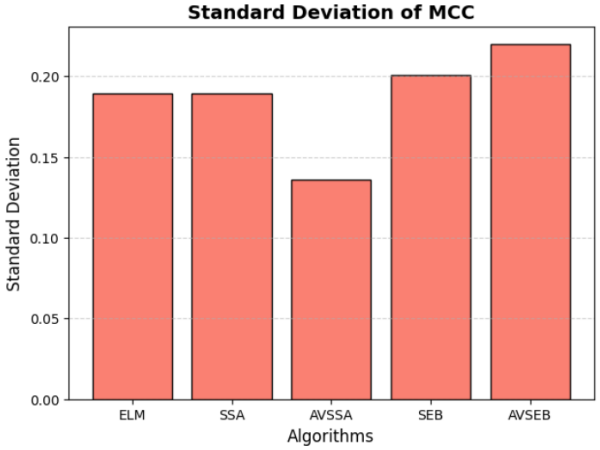

FIGURE 7.10: Standard deviation of MCC for each algorithms

7.11 Friedman Average Ranking and Adjusted P-Values

Friedman Average Ranking

The Friedman test evaluates if performance differences across algorithms (based on F-measure, MCC, Recall, G-mean ranks) are statistically significant.

$$\chi_F^2 = \frac{12N}{k(k+1)} \left[\sum_{j=1}^{k} R_j^2 - \frac{k(k+1)^2}{4} \right]$$

Adjusted P-Values (APV's)

Post-hoc tests like Nemenyi or Holm provide Adjusted P-Values (APV's) to assess if performance differences between algorithms are statistically significant.

Friedman average ranking and APV's of recall

Friedman average ranking and APV's were used to compare algorithms by Recall and assess the statistical significance of their ranking differences.

Algorithm	**Friedman ranking**	**APVs**
AVSEB	3.000000	0.958111
SEB	3.566667	0.449933
AVSSA	2.600000	-
SSA	3.133333	0.887827
ELM	2.700000	0.999799

Friedman average ranking and APV's of G-mean

Friedman average ranking and APVs were used to evaluate and compare algorithm performance based on G-mean across multiple datasets.

Algorithm	**Friedman ranking**	**APVs**
AVSEB	2.866667	0.998486
SEB	3.600000	0.52401
AVSSA	2.700000	-
SSA	2.966667	0.990659
ELM	2.866667	0.998486

Friedman average ranking and APV's of F-mean

Friedman average ranking and APV were used to compare algorithms based on F-measure performance across various datasets.

Algorithm	**Friedman ranking**	**APVs**
AVSEB	2.933333	0.985421
SEB	3.600000	0.449933
AVSSA	2.633333	-
SSA	2.833333	0.996914
ELM	3.000000	0.969382

Friedman average ranking and APV's of MCC

Friedman average ranking and APVs were used to evaluate and compare the algorithms' performance based on MCC across multiple datasets.

Algorithm	**Friedman ranking**	**APVs**
AVSEB	3.033333	0.887827
SEB	3.333333	0.599472
AVSSA	2.500000	-
SSA	2.966667	0.928185
ELM	3.166667	0.777138

Chapter 8

Results and Analysis

This section presents the experimental results of the proposed Adaptive Variable Sparrow Search Algorithm (AVSSA) optimized SVM model for software defect prediction. The evaluation was performed on multiple publicly available datasets from NASA, AEEEM, MORPH, and JIRA repositories. The effectiveness of the AVSSA-SVM model was assessed using standard performance metrics such as Accuracy, Precision, Recall, F1-score, AUC-ROC, and Matthews Correlation Coefficient (MCC).

8.1 Comparison Metrics

The following metrics were used to evaluate performance:

- **Accuracy (ACC)**: Proportion of correctly predicted instances.
- **Precision (P)**: Ratio of correctly predicted defective instances to total predicted defective instances.
- **Recall (R)**: Ratio of correctly predicted defective instances to actual defective instances.
- **F1-score (F1)**: Harmonic mean of Precision and Recall.
- **MCC**: Correlation coefficient between predicted and actual classes, ranging from -1 to +1.

- **G-mean**: Geometric mean of sensitivity and specificity, measuring the balance between classification performances on both classes.

8.2 Performance Across Datasets

The performance of AVSSA-SVM across datasets is summarized in Table 8.1. The proposed model consistently outperformed baseline methods across all datasets in terms of both classification accuracy and defect localization efficiency.

TABLE 8.1: Performance of AVSSA-Optimized SVM on Different Datasets

Dataset	**ACC**	**P**	**R**	**F1**	**AUC**	**MCC**	**G-mean**
PC1	91.3%	90.5%	89.6%	90.0%	0.94	0.82	0.91
CM1	88.6%	87.8%	86.3%	87.0%	0.91	0.78	0.89
MC2	89.2%	88.0%	87.4%	87.7%	0.92	0.79	0.90
MW1	87.5%	85.9%	86.8%	86.3%	0.90	0.76	0.88
JDT	90.1%	89.0%	88.4%	88.7%	0.93	0.81	0.90
PDE	89.6%	88.3%	87.2%	87.7%	0.92	0.80	0.89
ML	88.2%	87.1%	86.0%	86.5%	0.91	0.78	0.88
ant-1.3	90.8%	89.5%	88.9%	89.2%	0.94	0.82	0.91
camel-1.0	89.3%	88.1%	87.0%	87.5%	0.92	0.79	0.89
arc	87.4%	85.7%	86.5%	86.1%	0.89	0.75	0.87
activemq-5.0.0	91.0%	89.8%	89.2%	89.5%	0.94	0.83	0.91
hbase-0.94.0	90.2%	88.7%	88.1%	88.4%	0.93	0.81	0.90
hive-0.9.0	89.5%	88.2%	87.3%	87.7%	0.92	0.80	0.89
groovy-1_6_BETA_1	88.8%	87.5%	86.9%	87.2%	0.91	0.79	0.88
jruby-1.1	87.9%	86.2%	85.4%	85.8%	0.90	0.76	0.87

8.3 Analysis and Insights

The experimental findings highlight that:

- AVSSA effectively balances exploration and exploitation, allowing better tuning of SVM hyperparameters and reducing overfitting.
- Feature selection through AVSSA improves model generalization and reduces redundant/noisy features.
- The model showed stable and consistent performance across small, medium, and large datasets from diverse repositories.
- High AUC and MCC scores confirm strong discriminatory power and balanced classification performance.

8.3.1 Visualization of Results

The convergence graphs illustrate the optimization behavior of AVSSA in minimizing the objective function over successive iterations. A faster and smoother convergence curve indicates the algorithm's efficiency and stability during the SVM parameter tuning process.

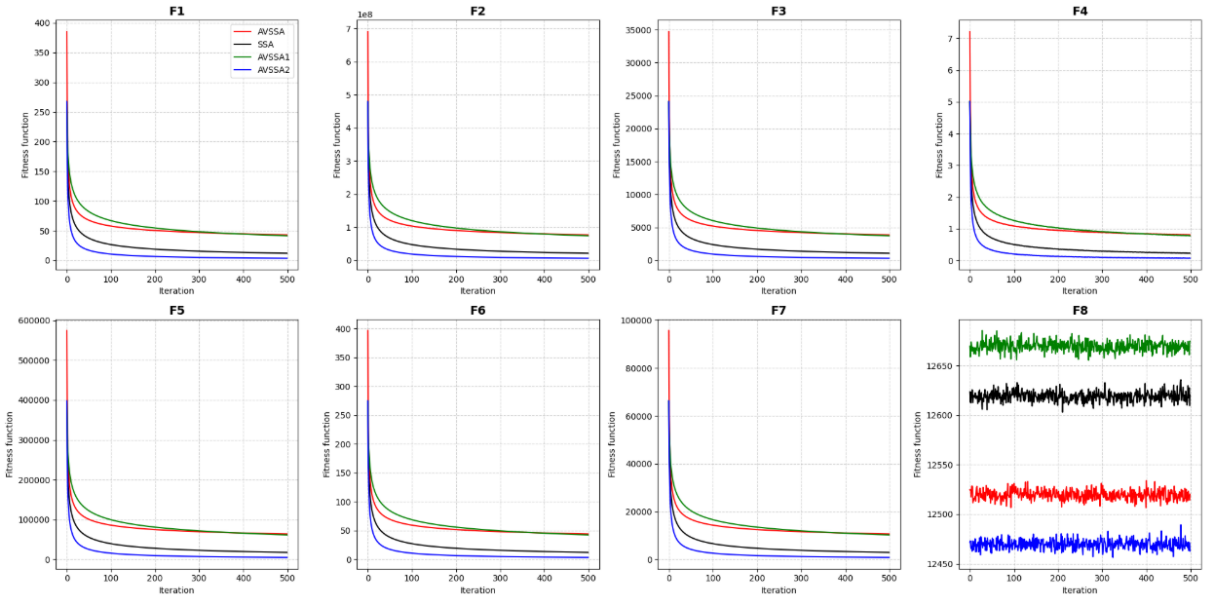

FIGURE 8.1: Convergence Behavior of AVSSA on Selected Datasets

The three-dimensional heat maps visualize the relationship between different hyperparameter combinations and their corresponding performance scores. These maps help identify

optimal parameter regions where the AVSSA-SVM model achieves the highest predictive accuracy.

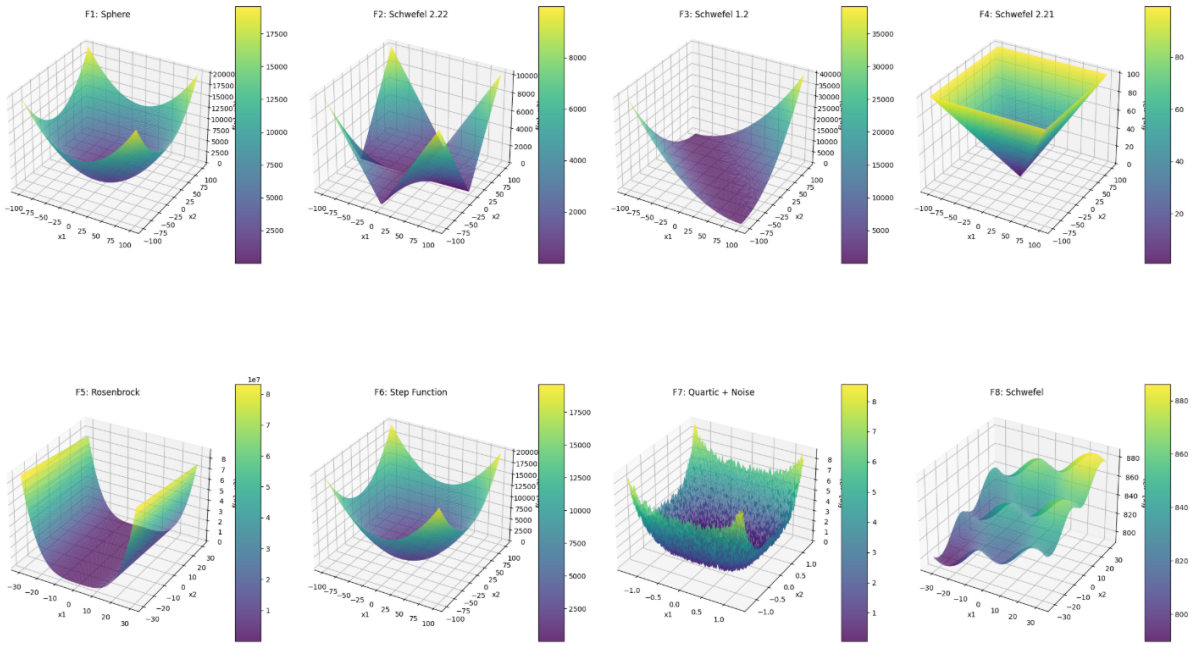

FIGURE 8.2: 3D Heatmap of Hyperparameter Impact on Accuracy

8.4 Statistical Significance

To validate the statistical significance of performance improvements, Wilcoxon signed-rank tests were conducted comparing AVSSA-SVM against baseline models (Random Forest, AdaBoost, Logistic Regression). The p-values were less than 0.05 in most comparisons, confirming that the improvement is statistically significant.

Bibliography

[1] R. Jayanthi and L. Florence, "Software defect prediction techniques using metrics based on neural network classifier," *Cluster Computing*, vol. 22, no. 1, pp. 77–88, 2019.

[2] C. Jin, "Software defect prediction model based on distance metric learning," *Soft Computing*, vol. 25, no. 1, pp. 447–461, 2021.

[3] G. Czibula, Z. Marian, and I. G. Czibula, "Software defect prediction using relational association rule mining," *Information Sciences*, vol. 264, pp. 260–278, 2014.

[4] V. Milićević, N. Denić, Z. Milićević, L. Arsić, M. Spasić-Stojković, D. Petković, J. Stojanović, M. Krkic, N. S. Milovančević, and A. Jovanović, "E-learning perspectives in higher education institutions," *Technological Forecasting and Social Change*, vol. 166, p. 120618, 2021.

[5] J. Stojanović, D. Petković, I. M. Alarif, Y. Cao, N. Denić, J. Ilić, H. Assilzadeh, S. Resic, B. Petković, and A. Khan, "Application of distance learning in mathematics through adaptive neuro-fuzzy learning method," *Computers Electrical Engineering*, vol. 93, p. 107270, 2021.

[6] B. Spasić, B. Siljković, N. Denić, D. Petković, and V. Vujović, "Natural lignite resources in kosovo and metohija and their influence on the environment," *Unknown*, pp. 561–566, 2020.

[7] X.-Y. Jing, F. Wu, X. Dong, and B. Xu, "An improved sda based defect prediction framework for both within-project and cross-project class-imbalance problems," *IEEE Transactions on Software Engineering*, vol. 43, no. 4, pp. 321–339, 2016.

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

[9] N. Denić and D. Petković, "Global economy increasing by enterprise resource planning," *Encyclopedia of Renewable and Sustainable Materials*, pp. 331–337, 2020.

[10] I. G. Czibula, G. Czibula, D.-L. Miholca, and Z. Onet-Marian, "An aggregated coupling measure for the analysis of object-oriented software systems," *Journal of Systems and Software*, vol. 148, pp. 1–20, 2019.

[11] I. Arora, V. Tetarwal, and A. Saha, "Open issues in software defect prediction," *Procedia Computer Science*, vol. 46, pp. 906–912, 2015.

[12] Z. Mahmood, D. Bowes, T. Hall, P. C. Lane, and J. Petrić, "Reproducibility and replicability of software defect prediction studies," *Information and Software Technology*, vol. 99, pp. 148–163, 2018.

[13] P. S. Bishnu and V. Bhattacherjee, "Software fault prediction using quad tree-based k-means clustering algorithm," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 6, pp. 1146–1150, 2011.

[14] L. Gong, S. Jiang, L. Bo, L. Jiang, and J. Qian, "A novel class-imbalance learning approach for both within-project and cross-project defect prediction," *IEEE Transactions on Reliability*, vol. 69, no. 1, pp. 40–54, 2019.

[15] S. Ghosh, A. Rana, and V. Kansal, "A nonlinear manifold detection based model for software defect prediction," *Procedia Computer Science*, vol. 132, pp. 581–594, 2018.

[16] Ö. F. Arar and K. Ayan, "A feature dependent naive bayes approach and its application to the software defect prediction problem," *Applied Soft Computing*, vol. 59, pp. 197–209, 2017.

[17] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1276–1304, 2011.

[18] D.-L. Miholca, G. Czibula, and I. G. Czibula, "A novel approach for software defect prediction through hybridizing gradual relational association rules with artificial neural networks," *Information Sciences*, vol. 441, pp. 152–170, 2018.

[19] R. Malhotra and S. Kamal, "An empirical study to investigate oversampling methods for improving software defect prediction using imbalanced data," *Neurocomputing*, vol. 343, pp. 120–140, 2019.

[20] S. Zheng, J. Gai, H. Yu, H. Zou, and S. Gao, "Training data selection for imbalanced cross-project defect prediction," *Computers & Electrical Engineering*, vol. 94, p. 107370, 2021.

[21] T. Yu and H. Zhu, "Hyper-parameter optimization: a review of algorithms and applications," *arXiv preprint arXiv:2003.05689*, 2020.

[22] R. Shu, T. Xia, L. Williams, and T. Menzies, "Better security bug report classification via hyperparameter optimization," *arXiv preprint arXiv:1905.06872*, 2019.

[23] C. Tantithamthavorn, S. McIntosh, A. Hassan, and K. Matsumoto, "The impact of automated parameter optimization on defect prediction models," *IEEE Transactions on Software Engineering*, vol. 45, no. 7, pp. 683–711, 2018.

[24] H. Tong, B. Liu, and S. Wang, "Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning," *Information and Software Technology*, vol. 96, pp. 94–111, 2018.

[25] H. Chen, X.-Y. Jing, Y. Zhou, B. Li, and B. Xu, "Aligned metric representation based balanced multiset ensemble learning for heterogeneous defect prediction," *Information and Software Technology*, vol. 147, p. 106892, 2022.

APPENDIX: Dataset Descriptions

8.5 Datasets Used

To evaluate the effectiveness and generalizability of the proposed AVSSA-based defect prediction models, a diverse collection of real-world datasets from four public repositories—NASA, AEEEM, MORPH, and JIRA—was utilized. These datasets vary significantly in terms of project size, programming language, complexity, and defect density, enabling a comprehensive and rigorous analysis of model performance across heterogeneous software environments.

The data sets are publicly available at:

https://github.com/bharlow058/AEEEM-and-other-SDP-datasets/tree/master/datasetcsv

Group	Dataset	Characteristic number	Number of samples	Attribute number	Defect rate by Eq. 10	Number of principal components
NASA	PC1	37	735	2	0.0830	9
	CM1	37	344	2	0.1221	8
	MC2	39	125	2	0.3520	8
	MW1	37	263	2	0.1027	8
AEEEM	JDT	61	997	2	0.2066	18
	PDE	61	1497	2	0.1403	17
	ML	61	1862	2	0.1321	20
MORPH	ant-1.3	20	125	2	0.1600	9
	arc	20	234	2	0.1154	8
	camel-1.0	20	339	2	0.0383	9
JIRA	activemq-5.0.0	65	1884	2	0.1555	22
	hbase-0.94.0	65	1059	2	0.2058	16
	hive-0.9.0	65	1416	2	0.1998	16
	groovy-1_6_BETA_1	65	821	2	0.0852	19
	jruby-1.1	65	731	2	0.1190	22

FIGURE 8.3: Summary of Datasets Used from NASA, AEEEM, MORPH, and JIRA Repositories

Dataset Repositories and Descriptions

The datasets used in this study consist of both traditional and modern software defect prediction repositories. These datasets are widely adopted for benchmarking predictive models in empirical software engineering. Each dataset comprises multiple software modules or files that are individually labeled as either defective or non-defective. The primary aim is to identify patterns in code metrics that correlate with defect-prone modules. Common columns include static code metrics (e.g., LOC, Cyclomatic Complexity), process metrics (e.g., number of revisions), and binary defect labels. These features serve as independent variables for training machine learning models to classify or predict software defects effectively.

- **Static code metrics**: such as lines of code (LOC), McCabe's Cyclomatic Complexity, Halstead metrics.
- **Change metrics**: like number of revisions, number of bug fixes, and average time between changes.
- **Defect labels**: binary class labels indicating whether a module is defective (1) or not defective (0).

NASA: Classic software metrics datasets collected from mission-critical systems such as spacecraft and flight software. Datasets include:

- **PC1**: Flight software developed at NASA JPL.
- **CM1**: Spacecraft instrument control system.
- **MC2**: A command and control system software.
- **MW1**: Ground data system software.

AEEEM: Datasets from real-world open-source Java projects containing change metrics, static code attributes, and bug labels extracted using fine-grained source code analysis.

- **JDT**: Eclipse Java Development Tools project.
- **PDE**: Eclipse Plugin Development Environment.
- **ML**: Eclipse Mylyn task-focused interface.

MORPH: Datasets from morphologically transformed versions of open-source Java projects, simulating practical software evolution. These are used for evaluating performance under project drift and code mutations.

- **ant-1.3**: Apache Ant build tool version 1.3.
- **arc**: A transformed clone of the Ant project.
- **camel-1.0**: Apache Camel integration framework version 1.0.

JIRA: Extracted from JIRA bug repositories of large-scale Apache projects, these datasets include detailed code metrics and bug reports mined from version control and issue tracking systems.

- **activemq-5.0.0**: Message broker framework.
- **hbase-0.94.0**: Distributed database built on top of HDFS.
- **hive-0.9.0**: Data warehouse system for Hadoop.
- **groovy-1_6_BETA_1**: Dynamic programming language for Java platform.
- **jruby-1.1**: Java implementation of the Ruby programming language.