Deep Learning Based Software Defect Risk Prediction Using Fuzzified Cluster Neural Network Feature Selection with Generative Adversarial Transfer Learning

V.Ruckmani[1],* and Dr. Prakasam S[2]
[1]Research Scholar, Computer Science and Applications, SCSVMV University, Enathur

[2]Associate Professor, Computer Science and Applications, SCSVMV University, Enathur

Abstract

Software defect prediction (SDP) plays a significant role in detecting the most likely defective software modules and optimizing the allocation of testing resources. In practice, though, project executives must not only identify defective modules, but also rank them in a specific order to optimize the resource allocation and minimize testing costs, especially for projects with limited resources. However, existing approaches face challenges in achieving reliable predictions. To address this, a novel approach is proposed the method is Fuzzified Cluster Neural Network feature selection with Generative Adversarial Transfer Learning (GATL). The proposed work includes four major phases: (a) pre-processing, (b) Feature Extraction and (b) deep learning-based SDP classification. Initial stage is we collected the dataset from standard repository and starting the pre-processing step is used for cleaning the data and transforming and normalizing the data for get original dataset. Second step is estimating the original data feature weightage values using Adaptive Static Feature weights Analysis (ASFWA) and third step is Fuzzified Cluster Neural Network feature selection is extracted features, the relevant ones are selected using the Fuzzified Cluster Neural Network feature selection. Finally, the SDP (decision making) is carried out using the optimized Generative Adversarial Transfer Learning (GATL) using to improve the model's prediction accuracy. The final detected outcome (predicting the defects ranking like high low and medium scores) is acquired from optimized GATL. The implementation has been performed using the Python software. By using certain performance metrics such as Sensitivity, Accuracy, Precision, Specificity and MSE the proposed model's performance is compared to that of existing models. The accuracy achieved for the proposed model.

Keywords: Software defect prediction, deep learning, Feature weights, Python software, Cluster, fuzzy, neural Network.

1. Introduction

Software Defect Prediction (SDP) technology is an effective tool for improving software system quality that has attracted much attention in recent years. With the increasing scale, the structure of a software system becomes increasingly complex. Software defects occur for several reasons, such as a misunderstanding of the software requirements, an ineffective development process, or a lack of software development experience—generally, defective software results in unanticipated economic losses to enterprises and even human-related costs [1]. If software defects can be found during the early stage of development, the software quality can be improved, and the related fees will be reduced.

SDP can construct models by mining version control systems and defect tracking systems, and then the constructed models can be used to predict defective modules in advance. Therefore, limited software quality assurance (SQA) resources can be reasonably allocated to these identified defective modules, effectively improving the quality of deployed software. However, a target project may sometimes be new or have a few labeled modules [2]. The most straightforward method is directly using the labeled modules in other projects to train the models. Machine learning algorithms trained on these class-imbalanced datasets may be biased towards non-defective instances and misclassify defective cases [3]. The prevalent methods for tackling class imbalance problems are sampling, cost-software defect, and ensemble learning methods. These over-sampling methods can improve classifiers to identify more instances in minority class, but at the same time they may lead to misclassify many instances in majority class.

A wide range of statistical and machine learning models have been employed to predict defects in software modules [4]. However, the imbalanced nature of this type of SDP dataset is pivotal for the successful development of a defect prediction model. which has attracted a lot of attention from the industrial communities and academicians over the last three decades. It allows software engineers to allocate a limited workforce, time, and other resources to defect-prone modules through early defect prediction. It also plays a vital role in reducing software development costs and maintaining the high quality of software systems.

The prediction results can assist developers in prioritizing their testing and de-defecting efforts. As a result, software defect prediction techniques, which predict the occurrence of defects, have been widely used to assist developers in prioritizing their testing and de-defecting efforts. However, the increasing complexity of modern software has elevated the importance of software reliability [5]. Building highly reliable software requires a considerable amount of testing and de-defecting. However, since budget and time are limited, these efforts must be prioritized for efficiency.

However, the performance of cross-project defect prediction is relatively low because of the distribution differences between the source and target projects. Furthermore, the software defect dataset used for cross-project defect prediction is characterized by high-dimensional features, some of which are irrelevant and contribute to low performance. However, the cost of software testing is almost half of the development cost [6]. Consequently, managing the available limited resources is paramount in a situation where resources are limited. However, in practice, for new projects or companies that lack local defect data repositories because of the cost of maintaining such repositories, it will be challenging to apply SDP.

2. Literature Survey

Accordingly, they use data analysis and Machine Learning (ML) algorithms to selectively reduce data dimensionality associated with critical features in SPD [7], allowing energy to be focused on areas where power is needed, thereby playing an essential role in achieving the SDP objective.

They proposed a point-centering approach utilizing the K-means algorithm to address the issue of random centroid values in K-means. They employed software to forecast errors in

defective blocks to identify the optimal initial centroid value [8]. Nevertheless, the suggested approach does not yield ideal outcomes for arbitrarily selecting centroid points.

They proposed a transformation and feature selection method to reduce distributional disparities and high-dimensional features in inter-item defect prediction [9]. However, source and target projects have different distributions, and cross-project defect estimates perform poorly.

A deep neural network (DNN) based model specifically created to forecast the number of defects is applied to the modeling data. Additionally, two well-known datasets will be used to assess the suggested approach [10]. Moreover, to effectively utilize the limited resources available to developers, it is imperative to automatically anticipate the number of errors in a software module.

They recommend using process metrics, source code churn rate, and source code metric entropy as predictor variables in future defect prediction studies. Furthermore, the cautious use of single metric methods as predictor variables is presented [11]. Previous defect metrics and other single-metric methods have performed poorly.

The novel described developing a comprehensive SDP method that uses decision tree-based algorithms to enhance accuracy, feature selection, and evaluation metrics [12]. The offered method has led to significant losses, including financial loss, damage to a company's reputation, and, in extreme cases, loss of life.

They introduced Human Error-Based Defect Prediction (HEDF) by investigating the cognitive processes behind developer errors. Specify defects in early software development stages before coding to predict them accurately [13].

The proposed Multi-Criteria Decision-Making (MCDM) method utilizes defect prediction and feature selection rates to evaluate candidate solutions [14]. Nonetheless, in the software industry, early software defect prediction is vital.

They proposed a Multi-Layer Perceptron (MLP) method based on an ensemble classification framework to predict defective software modules. After the first dimensional optimization, they implemented the MLP algorithm for classification with improved packaging technology integration [15].

Table:1 Deep Learning Based on Software Defect Prediction

Author	Year	Technique Used	Drawback	Performance Evaluation	Accuracy
A A Saifan [16]	2021	Support Vector Machine (SVM)	The presence of software defects diminishes the quality and results in project failure.	Area Under Curve (AUC)	75%
Lin J. [17]	2021	bi-directional long short-term memory (BiLSTM)	The sequence of tokens alone is not sufficient to distinguish the tree structure of an abstract	Recall, F-measure, AUC, and Matthews Correlation Coefficient	68%

			syntax tree.	(MCC)	
Mehmood [18]	2023	Logistic Regression (LR), SVM	It is crucial to connect software engineering and data mining.	Accuracy	93.05%
Miholca D.L [19]	2022	artificial neural network (ANN)	SDP is a challenging problem in the highly active field of analyzing large complex systems.	Sensitivity, Specificity	95%
Majd, Amirabbas [20]	2020	LSTM	Poor quality software can result in significant financial costs.	Sensitivity, Recall	72%
Mehta, S [21]	2021	Recursive Feature Elimination (RFE)	The testing process is vital in software development	Precision, F-Measure	93%
O. A. Qasem [22]	2020	MLP, Convolutional Neural Network (CNN)	Poor quality due to software defects.	Sensitivity, Accuracy	89%
Li, Zhen [23]	2021	particle swarm and the wolf swarm algorithm	Timely elimination of software defects is crucial to avoid severe financial losses.	Recall rate, Precision	76%
Rathore S.S [24]	2022	generative adversarial network (GAN)	Software practitioners face a challenge in managing asymmetric error data.	Recall, F1-score,	79%
Šikić L [25]	2022	Graph Convolutional Neural Network (GCNN)	Attention to a small part of the code because the source code ignores its tree structure.	F-score	81%
Tong [26]	2020	credibility-based imbalance boosting (CIB)	Categorizing software defects as either defective or non-defective complicates predicting them.	AUC, MCC	87.6%

They proposed a deficit prediction model based on a Gated Hierarchical Long-Short-Term Memory (GH-LSTM) network to extract semantic features from word embedding structures [27]. However, they may need to correct feature defects such as lines of code.

They are analyzing software metrics and predicting defects using an automated tool called GraphEvoDef. The evolution of software is described, and recent comparative developments in networking are analyzed [28].

Recent experiments have shown that semantic features based on Deep Belief Network (DBN) technology can significantly enhance defect prediction tasks. Furthermore, when comparing semantic features to traditional features, the improvement in file-level project defect prediction ranges from 2.1 to 41.9 percentage points [29].

The outcome demonstrates that structural information from software networks considerably impacts CNN techniques, and the performance improves when paired with semantic features. Additionally, the F-measure often rises with a maximum growth rate of 92.5% compared to traditional structural features [30].

3. Proposed method

Recently, ML techniques are being highly applied for automated SDP. These approaches require reduce computation time and reduce manually extracted features. DL approaches enable practitioners to automatically extract and learn from more complicated and high-dimensional data. Therefore, in this research work, a novel GATL-based SDP model is introduced. The proposed work includes the following phases:"(a) pre-processing, (b) Feature selection and (c) classification.

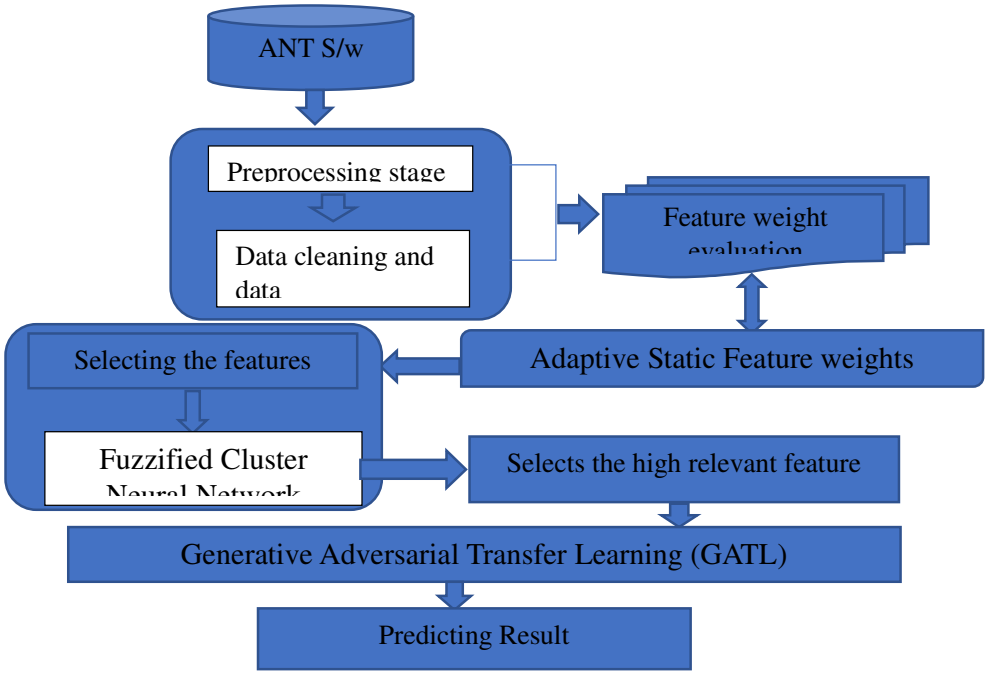

Figure 1: Proposed Diagram

Figure 1 described as, Purposed diagram for software defect prediction using deep learning based GATL for predicting the rosily (Risk level),the proposed method include preprocessing to reduce null values and cleaning the data second step is to get preprocessed data evaluating the feature maximum and minimum weights using Adaptive Static Feature weights Analysis ,third step is feature selection based on the maximum threshold values using for

evaluating the high relevant features based on fuzzified cluster neural network. Finally classification using GATL is predicting the result based on the risk factors.

3.1 Data Preprocessing

Initially, the collected raw data is pre-processed via data cleaning and Decimal scaling normalization approach. Since datasets frequently values, contain missing noise, and noticeable changes in the size of the features, data pre-processing is frequently done before training ML models. The following pre-processing steps have been used for the ANT software dataset.

Data cleaning -Missing Data Removal (MDR)

The term "missing data" refers to information that is not recorded for a variable for particular observation. Missing data lowers the analysis's statistical power, which might skew the conclusions' validity. To prevent bias when dealing with missing data at random, relevant data may be eliminated. If there aren't enough observations to perform a reliable analysis, data removal may not be the best solution. In some cases, it may be vital to keep a watch on specific things

Given: ANT Software defect dataset

Obtain: preprocessed dataset

Begin

Read sdf data.

Generate software defect dataset sdf➔d1, d2, d3 + pd (Index +1)

Find features of sdf

For each preprocessed data (Pd)

If Pd contains all then

Ok

Else

Remove null values

End

For each feature f

If value is null then

F = fill with near average value.

End

End

Add sdf to Pd.

End

Stop

The working of preprocessing algorithm is presented in the above pseudo code, which finds the records with missing features which has been removed from the data set where the trace with value missing are replaced with near mean value of the feature.

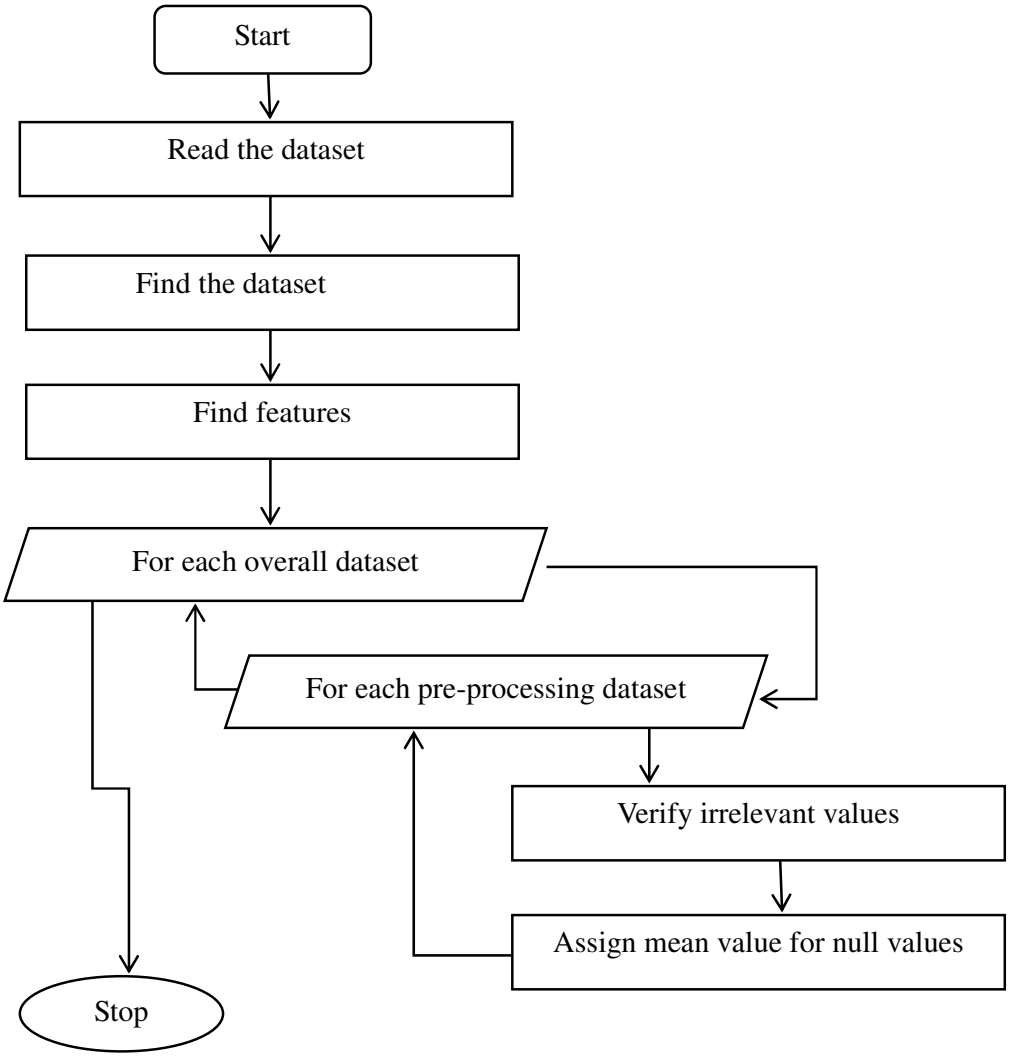

Figure 2 Flow chart of Data Preprocessing

The flow chart of data preprocessing is presented in Figure 2. Where the method read the trace and remove the noisy records and assigns null value with mean near value to support the feature weights process.

3.2 Feature weights evaluation using

To obtain the features from more number of attribute cases and initially they count reframes to reduce the dimension along the reduction. Secondly, to construct feature importance weightage by marginal acceptance of varying margins of a feature importance. The Dominant features weightage from weather dataset by retaining the Probability to getting the features in percentage in weather dataset. Let F be the Finite feature along the probability to getting the importance features along the differential space between weightage (Ω, F, P), where outcome referred as Ω with finite feature occurrence F with a similar zed cluster weightage point space P.

by the events referred P: F → [0, 1] weather features occurrence at weightage margin Wm(i j), where i is the initialization feature and J is the relational feature.by the event occurrence of feature in σ as affine state f(A). The algebraic function recommended as non-functional features with closest weight by intersection theory of feature occurrence

$$F(Wm(i,j)) \rightarrow \begin{cases} \text{if } A \in F \text{ then } Ac \in F, \\ \text{if } Ai \in F \text{ is a countable sequence of values then } \cup iAi \in F \\ C \cap iAi = (c \cup iAc\, i)\, c \end{cases} \quad (3.8)$$

Where closest countable terms are μ: F → R at relevance measure, buy the exceptional terms P→ (Ω, F)

$$P(Wm(i,j)) \rightarrow \begin{cases} \mu(A) \geq \mu(\emptyset) = 0 \text{ for all } A \in F, \\ \text{if } Ai \in F \text{ is a countable sequence of coefficeint values sets} \\ \mu(\cup iAi) = X\, i\, \mu(Ai) \end{cases}$$

The probability of features points P get the consequences of weightage measure between the features closet to μ (Ω) = 1, In all cases, the intersections are the occurrence of marginal relevance obtained from probability measure P(f).

Input: A = {Ax}, the set of all Dataset (n)
Output: B = {Bx,y}, A collection of features in a dataset, row vector features of the original data.
Step 1: Initiate
Step 2: Using ASFWA ← for extracting the features in the original data set
For x= 1 to n do
Decompile the Ax
Generate the extracting feature f (151-bit) from Manifest
Generate the SDP features a (3262-bit) from sources;
End for
For y = 1 to 151 do
B x, y = a y;
End for
For y = 152 to 3413 do
B x, y = f y;
End for
Step 3: πa is usually frequencies of the training set:
Π a= (No.of.sample class A)/Total of samples
Step 4: End procedure

Where, ASFWA- Adaptive Linear Component Analysis, SDP- Software Defect Features, n- Number of datasets, Ax – Input values, B x,y – features of row and section values is utilized for the dataset investigation of the first data to remove the elements in light of this calculation. It helps in eliminating the improper qualities and choosing the handling information in the Defect programming dataset. The SDP guarantees that this interaction is an application that utilizes the A way, so it can consolidate SDP with consents to make more agent and exhaustive element extraction that mirrors the way of behaving of 3413 pieces (151 extricating highlight capacities + 3262 SDP capacities), which is substantially less mind boggling and each defect the discovery framework can likewise be produced for every application.

3.3 Fuzzified Cluster Neural Network feature selection

Analyses software defect features from software data using cluster approaches. Cluster Neural Network feature selection approach is very helpful for examining significant characteristics of clinical data. Data noise can be present in ANT datasets. Fuzzy neural network can be used to examine variables in defects datasets. Fuzzified Cluster Neural Network feature selection can assist in identifying the characteristics based on maximum threshold values for selects the high relevant feature.

Convert fuzzy input data (A) into fuzzy sets. Each feature in the data set a_x is represented by the membership function$\mu_x(a)$. Membership functions that are trapezoidal or trigonal are popular choices. Convert the gynecological features in equation 1 into fuzzy sets.

$$\mu_{I_x}(a_x) = \frac{a_x - i_x}{y_x - i_x} \text{ for } i_x \leq a_x \leq y_x$$

Where $\mu_{\{i_x\}\{a_x\}}$ is the membership function for feature (a_x), and (i_x) and (y_x) are the lower and upper bounds of the feature a_x .

Establish a set of fuzzy rules that define how the features of the input and the output are related. For example, if we have two features a_1 and a_2, a fuzzy rule might look like:

If a_1 is I_1 and a_2 is I_2, then b is J

Where I_1 and I_2 are fuzzy sets corresponding to the features a_1 anda_2, and J is the fuzzy set for the output b.

To obtain fuzzy outputs, apply the fuzzy rules to the fuzzified input. The fuzzy rules are combined in this step by using logical operations like min, max, etc. The output of each rule Q_i can be computed as:

$$\mu_{J_x}(b) = min\left(\mu_{I_1}(a_1), \mu_{I_2}(a_2)\right)$$

Restore the crisp value to the fuzzy output by applying defuzzification techniques. The centroid approach is a popular technique that determines the combined fuzzy set's centre of weight:

$$b = \frac{\sum_{x=1}^{N} \mu_{J_x}(b).b_x}{\sum_{x=1}^{N} \mu_{J_x}(b)}$$

Where b_x are the crisp output values corresponding to each rule Q_x.

Combine the results from the fuzzy inference system to create a regression model. The fuzzified regression equation can be expressed as:

$$\hat{b} = \sum_{x=1}^{N} h_x . b_x$$

Where h_x is the weight of the $x - th$ rule, often determined by the degree of membership:

$$h_i = \mu_{I_x}(a_x)$$

For a simple example with two features a_1 and a_2 the fuzzified regression equation might look like:

$$\hat{b} = \sum_{x=1}^{N} \left(min \left(\mu_{I_1}(a_1), \mu_{a_2}(x_2) \right) . b_x \right)$$

This method works well for complicated ANT Software defect records with a wide variety of data types. This flexibility guarantees thorough examination of various clinical datasets. This method may improve patient outcomes and boost the effectiveness of risk and prompt intervention through precise analysis of software defect aspects. In order to promote better healthcare solutions and increase the accuracy, robustness, and usefulness of clinical data analysis, FCNN approaches offer strong tools for analyzing important aspects of software records.

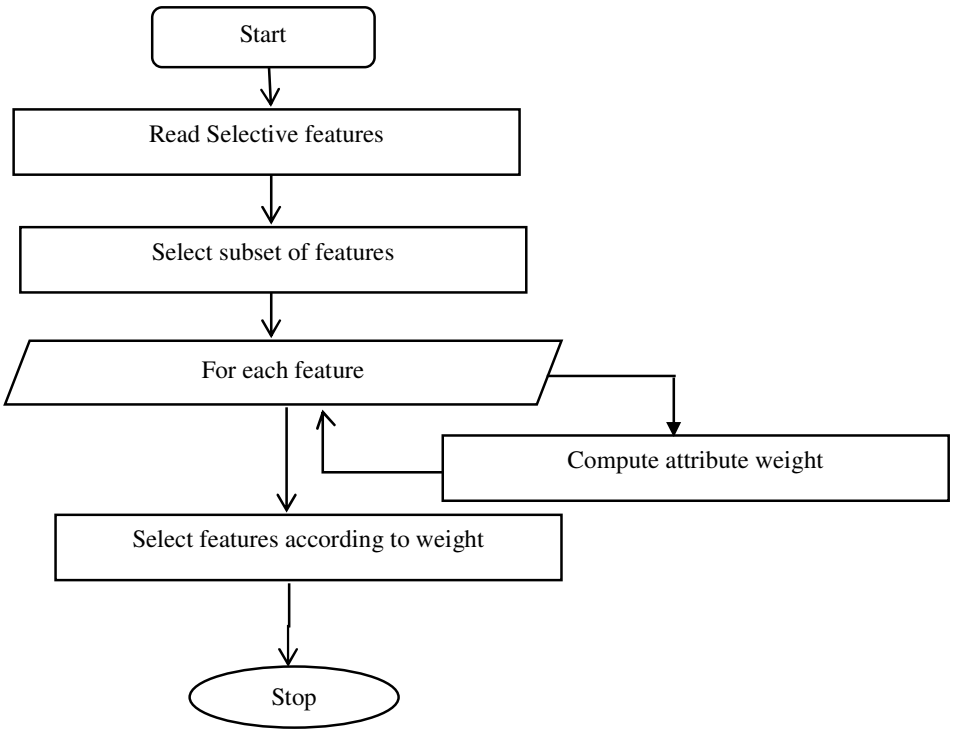

Figure 3 Flow chart of Feature Selection Model

The working principle of feature selection model is presented in above Figure 3 The method selects set of features according to attribute weight and based on that the method performs features selection to support plant selection.

3.2.5 Soft Max logical activation function (SMLAF)

The proposed model has been designed with soft max logical activation function which is intended to measure forecasting feature rate and success rate based on which set of weathers are sorted and ranked to produce result. To support weather forecasting, the neurons present in each layer would measure weather forecasting feature rate value and Max success rate for various weather conditions. As the traces are grouped under different weather class and there exist set of patterns for each class of weather, using them the method computes the value of weather forecasting feature rate for each successive class. Similarly, the method computes the value of max success rate for each class of weather considered. Further, the method sorts the weather forecasting, according to the max success rate value. Sorted values or forecast classes are produced as recommendation.

Pseudo Code of SMLAF Algorithm:

Given: weather Class set WDcs, Cluster C, Test Sample s

Obtain: Recommendation R

Start

Fetch DCS, C

For each Successive class dc

Measure Software defect risk rate

$$\text{WFFR} = \sum_{i=1}^{Size(C(dc))} Features(C(dc(i)) \equiv Features(s) \Big/ size(C(dc)) \quad (3.10)$$

End

Identify the class C with maximum WFFR.

For each forecasting for class C

$$\text{Measure Max success rate SR} = \sum_{i=1}^{size(C)} C(i).State == Success \Big/ size(C) \quad (3.11)$$

End

Recommendation = Sort the Max state according to success rate from weather data

Stop.

The above pseudo code shows Feature Selection Model works to generate recommendations. The method computes software defect risk rate and identifies the forecast class with maximum value. Now with the forecast class, the method estimates max success rate and ranked to produce recommendations.

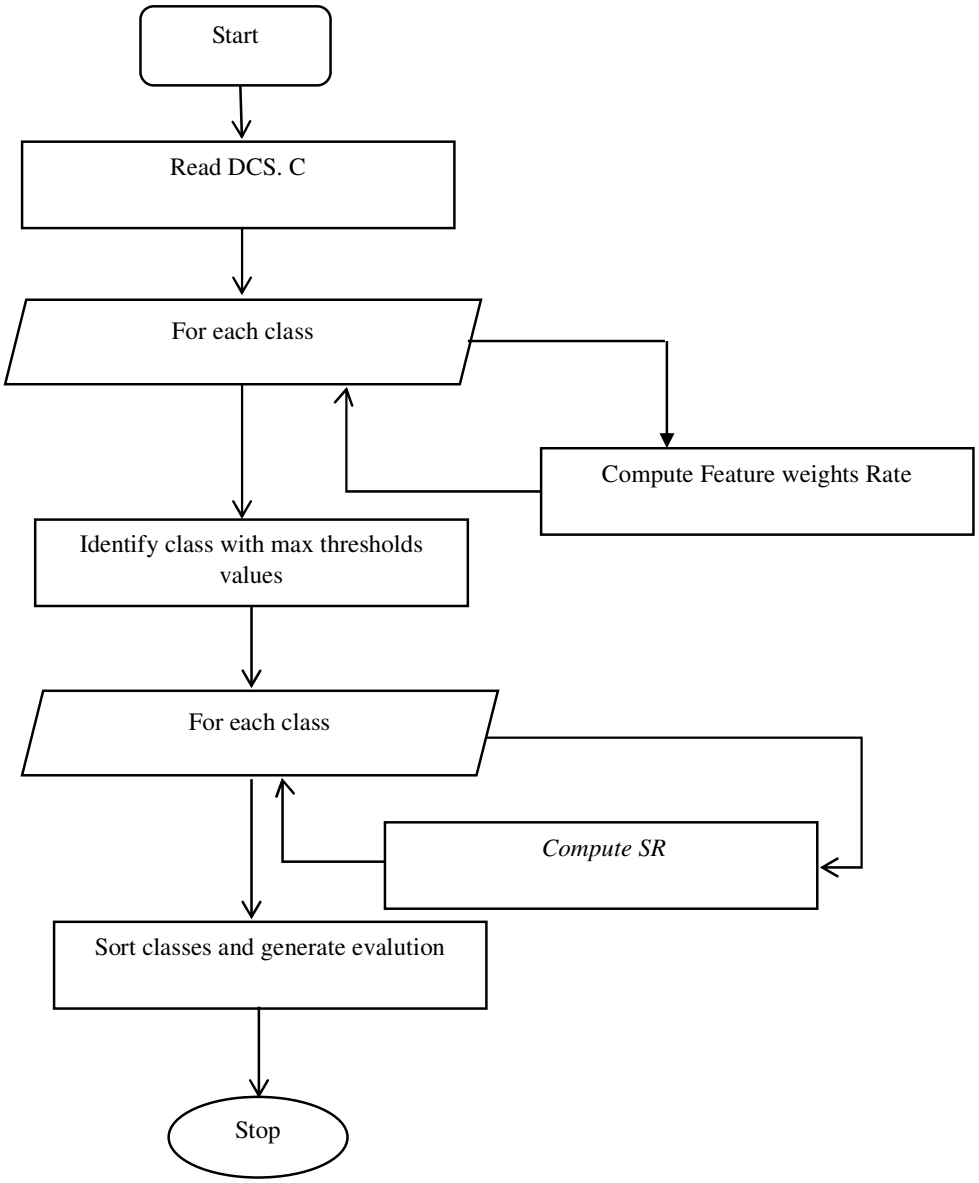

Figure 4 Flow chart of SMLAF Algorithm

The working of proposed SMLAF algorithm is presented in Figure 4, which computes WFFR and SR values towards various classes to generate recommendations to the user.

3.4 Classification using Generative Adversarial Transfer Learning (GATL)

The Generative Adversarial Transfer Learning (GATL) has the adaptability for calculating risk characterization. Compute the mass conventional transfer of neurons in a single layer and apply one more brain contribution to the following layer nonlinear actuation capacities can utilize weight nonlinear cooperation relying information estimate by the neuron. The GATL for catagorising the risk level of SDP using the Adversarial layers

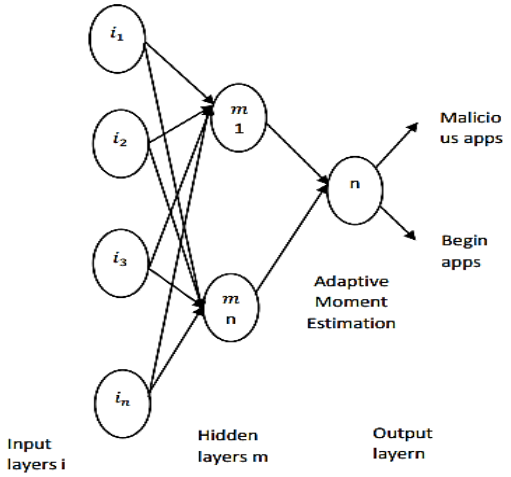

Figure 5 GATL **Architecture**

GATL, a completely embodied secret layer with input neurons, m secret neurons and n yield neurons, and np patterns and q attributes, as shown in Figure 5.

Consider a dataset that is fully integrated with a hidden layer of GATL, which includes a combination of i input neurons, m hidden neurons, n output neurons, and D and np forms and q attributes.

Step1: Given accuracy specific speeds are used and the GATL algorithm is used for personal training and test data network training.

Step2: Featured data included (Un, Vd, s, tnet , F)

Step3: Use each hidden node to calculate the total net worth of all the forms in the database

$$t_{net}m=\sum_{i=0}^{np}(D_i.w_{np})$$

Step4: Eliminate hidden neurons and calculate precision measurements for hidden neurons k if $sn \leq \alpha$, {1,2, … 10}

Step5: Re-train the current Network

Step6: Utilize the result mistake to ascertain the blunder signal for the layer before the result.

Step7: The third layer plays out an Adaptive Moment Evaluation (Adam) in light of the ID of the malicious application and the normal exhibition of each vindictive programming discharge because of the application.

$$s_n = \sum_{k=1}^{n}(n(t_{net}m) - s_i)^2$$

Step8: If the classification rate of the network is less than acceptable, stop pruning; If not, go-

ahead 2.

To software defect data or unapproved admittance to private organizations, different software defects are made for various purposes. Software programming that endeavours to perform inductive trainings on the Processor is a creation database. It will recognize any software defect identifies the datasets based on the software. Utilizing GATL with high detection outlines in software defect structures for utilizing the risks.

Algorithm

Inputs: Training and testing data

Output: Determine the accuracy of the calculation

Step 1: Handle SRC= Nt Open File;

Step 2: Handle section Handle + NtCreatesection (Section Handle);

If (QueryAttributesFile>TF)

While (Stopping condition is not met) do

Implement GATL training step for each data values

GATL→Calculating weights $TF_{xy} = W_{x,y}/T_{x,y}$

Implement GATL to classify the testing data values

$$TF_i = \log\frac{N}{Ni} + 1$$

End While

End If

Return Accuracy

Step 3: End

Where W-weight values, TF-Time Frequency, Analyzes the accuracy of classification based on software defect detection and performs optimal values for training and analysis of test data.

4. Result and discussion

The proposed models towards software defect prediction has been implemented in python and evaluated for its performance under various constraints. The methods are evaluated for their performance in different parameters using ANT data set. Obtained results are compared with the results of various other approaches in this section.

Table 2 Details of Evaluation

Parameters	Ratings
Framework	Anaconda
Language used	Python
Dataset name	ANT software Dataset
Number records	5000
Number of attributes	23
Training	70%
Testing	30%

The details of data set and features considered for the performance evaluation has been presented in Table 6.1. According to this, the performance of the method has been measured over different performance metrics.

4.2 Performance model

1. Accuracy:

The performance of methods are measured for their classification accuracy. It has been measured based on the number of true negative and true positive classification generated correctly with the total number of classification performed. It has been measured as follows:

$$\text{Accuracy} = \frac{TN+TP}{(TP+FP+FN+TN)} \tag{6.1}$$

2. False Ratio:

The ratio of false classification produced by various approaches are measured and presented in this part. The false ratio has been measured by computing number of false classification made among number of classification performed.

$$\text{False Ratio} = \frac{\text{False Positive+False Negative}}{\text{Total Classifications}} \tag{6.2}$$

3. Time Complexity:

The performance of methods is measured on the time complexity, which is done based on the value of total time taken and number of features handled. It has been measured as follows:

$$\text{Time complexity (Tc)} = \sum_{k=0}^{k=n} \times \frac{\text{Total Features Handeled to Process in Dataset}}{\text{Time Taken(Ts)}} \tag{6.3}$$

4. F-Measure:

F-measure is a performance metric used to evaluate the accuracy of a classification model. It is calculated using the model's precision and recall and provides a single score that takes both false positives and false negatives into account.

$$\text{F-measure} = \frac{2 * \text{Precision} * \text{Recall}}{(\text{Precision} + \text{Recall})} \quad (6.4)$$

Precision is the ratio of true positive predictions to the total number of positive predictions, and recall is the ratio of true positive predictions to the total number of actual positive instances.

5. Specificity:

The specificity of any classification algorithm is measured according to the number of true negative classification produced for number of true negative and false positive classification. It has been measured as follows:

$$\text{Specificity} = \frac{TN}{TN+FP} \quad (6.5)$$

Comparative analysis

The comparison methods are No of iterations/Methods, HYDRA, KSSP, PMHMFT, DGRNN, and SMAN2.

Table 3: Analysis on Classification Accuracy

Methods /No of records	Impact of Classification Accuracy in %					
	HYDRA	KSSP	PMHMFT	DGRNN	SMAN2	GATL
500	87.3	91.1	93.1	93.8	94.2	95.1
1000	89.5	92.2	93.6	94.2	95.8	95.9
1500	91.3	92.7	93.8	95.4	96.2	96.8
2000	92.1	92.5	94.2	95.6	96.9	97
2500	92.6	92.9	94.2	95.6	97.1	97.8

The accuracy of classification made by various approaches at the availability of different number of records in the data set has been measured and presented in Table 6.2. The proposed MIFSM – S3-DNN has produced higher performance in classification than other approaches.

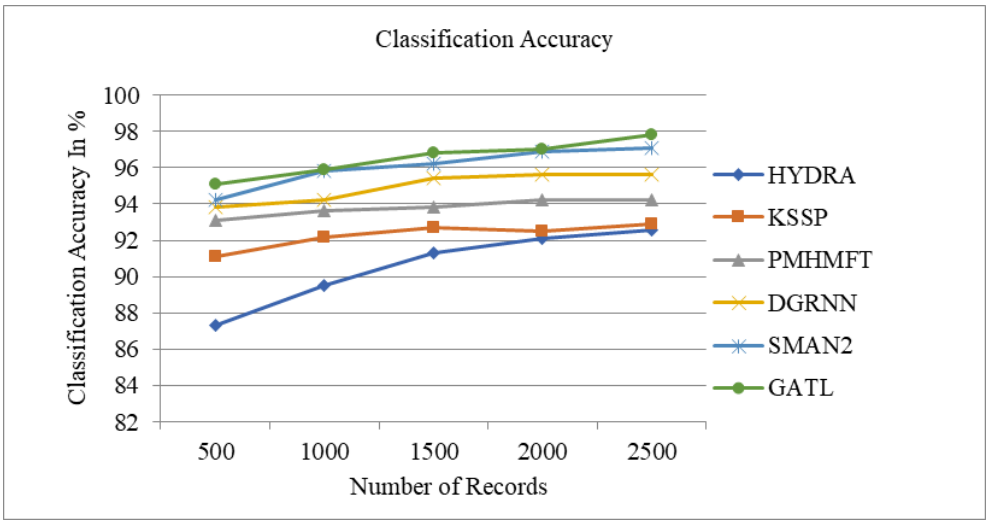

Figure 6: Analysis of Classification Accuracy

Figure 6 describes the crop recommendation comparison of classification accuracy performance. The proposed method performed better than others, achieving an accuracy of 97.8% for 2500 records. This was possible due to the proposed method's identification of fluid growth and soil growth-centric analysis for crop recommendation. As a result, produce the high performance than other methods.

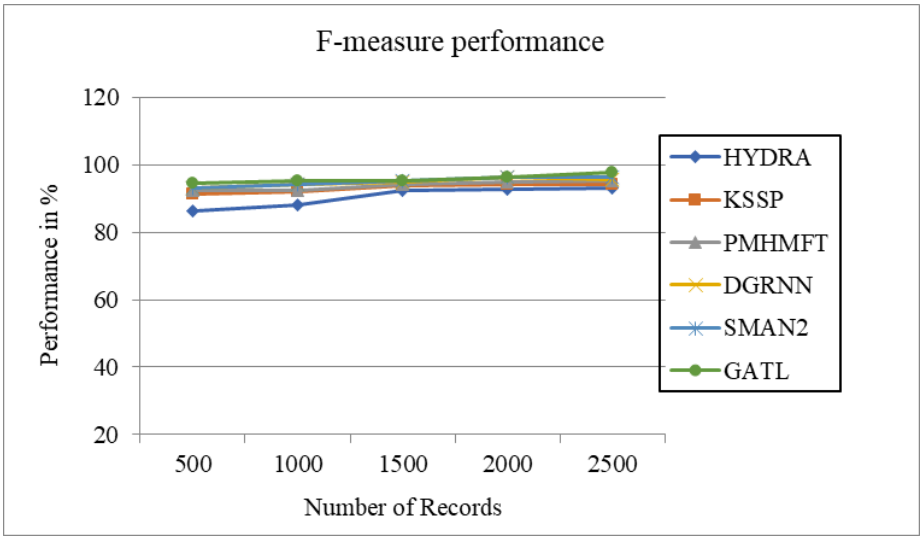

Figure 7: Impact of F-measure performance

The performance of various methods in terms of F-measure is analysed and compared with the results of other methods. F-measure is a valuable metric for evaluating the overall performance of a classification model, as it balances the trade-off between precision and recall. The findings are presented in Figure 7. The proposed approach has demonstrated a better performance than other existing approaches.

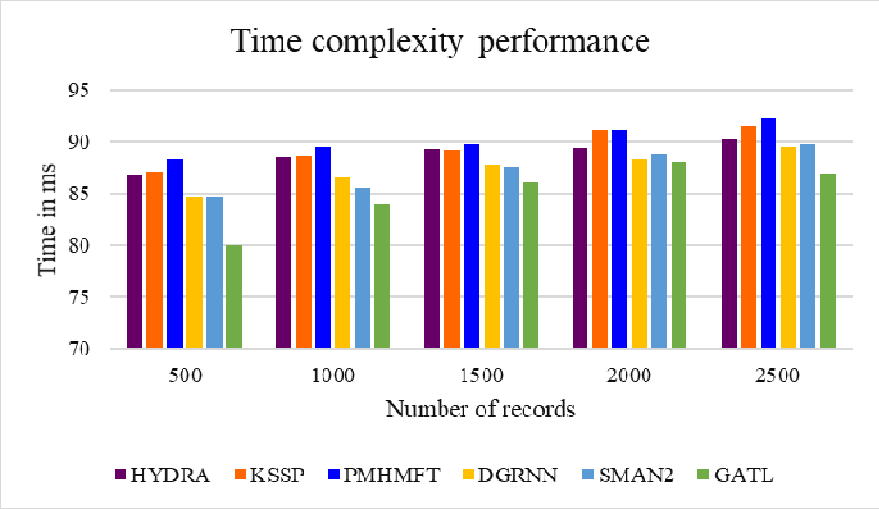

Figure 8: Analysis on Time Complexity

Figure 8 shows the comparison results of crop recommendation time complexity performance. The proposed method performed better than others, achieving a time of 88ms for 2500 records. This was possible due to the proposed method's identification of fluid growth and soil growth-centric analysis for crop recommendation. As a result, it took less time to achieve complexity performance than other methods.

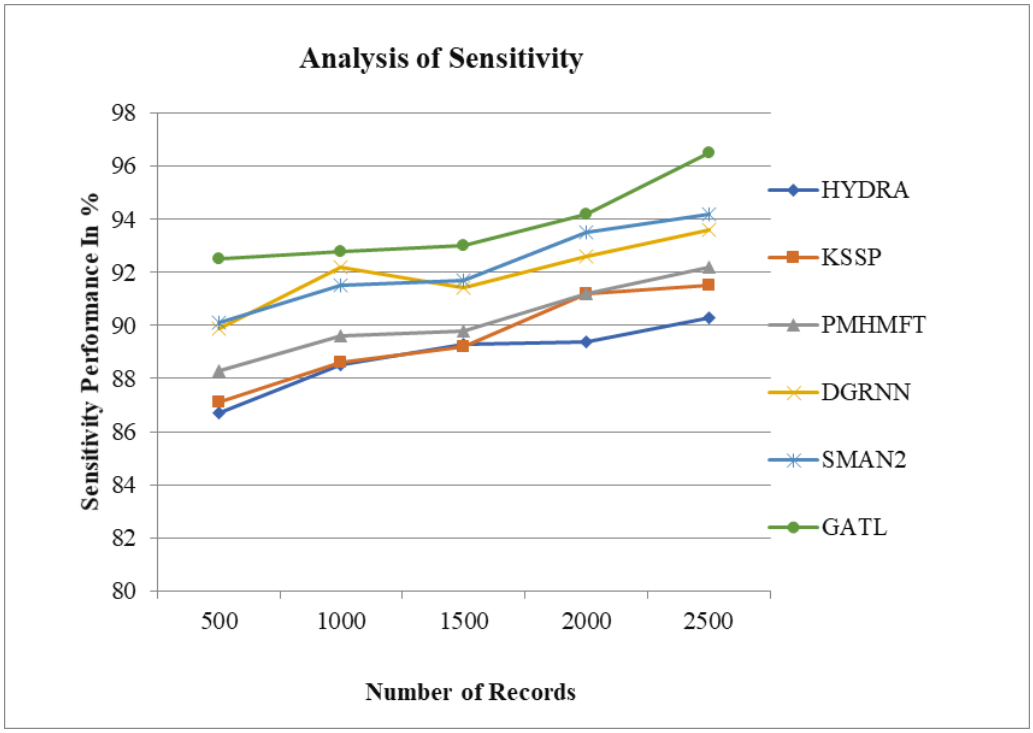

Figure 9 performance of sensitivity analysis

Additionally, the sensitivity performance comparison result with different types of records is illustrated in the figure 9. Sensitivity refers to the proportion of positive cases the system correctly identifies. In crop recommendations, high sensitivity values indicate that the proposed system effectively recommends suitable crops for planting based on conditions.

Figure 10 Analysis on Specificity

Figure 10 illustrates the comparison of specificity performance for crop recommendation. Specificity measures the proportion of actual negative cases that the system correctly identifies. In the context of crop recommendation, it indicates how well the system avoids recommending unsuitable crops for a given set of conditions. A high specificity value would mean the system effectively filters out crops that are unsuitable for planting.

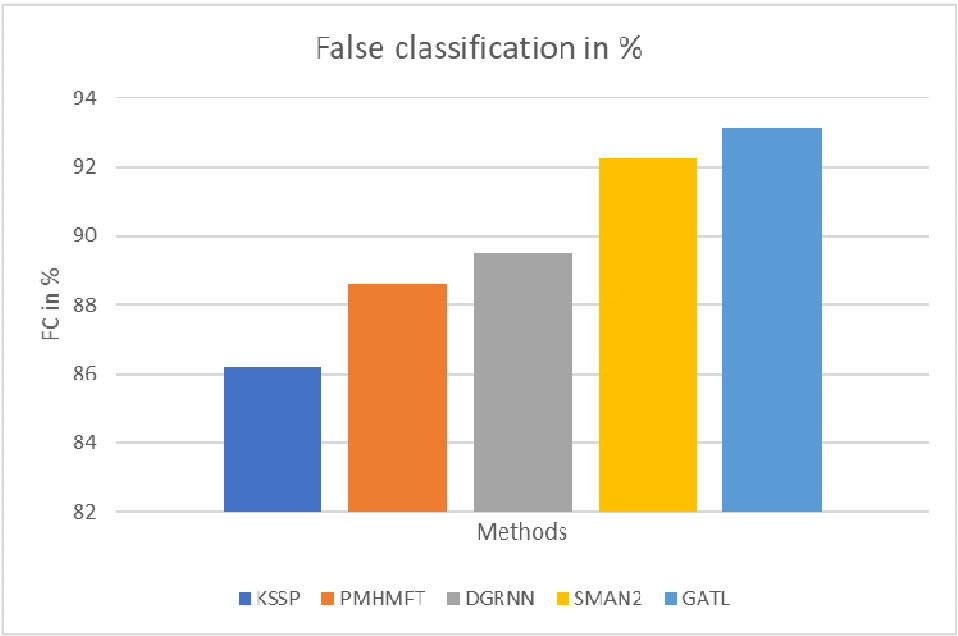

Figure 11 performance of false classification

False classification performance, in this case, refers to the rate at which the system inaccurately categorizes crops as suitable or unsuitable for a particular set of conditions. It is worth noting that the proposed in figure 11.

5. Conclusion

Software Defect Prediction (SDP) is essential to software testing and quality assurance. It has become even more fundamental in recent years, as the number of programs and software

products has also increased in size and complexity. Identifying defective ranges to rank software risk levels and minimize testing costs. Models using two standard output metrics: defect count and defect density as target variables. It also studied the effect of using imbalance learning and feature selection using Adaptive Static Feature weights Analysis (ASFWA). The FCNN results of the models showed that using defect count as the target variable produced higher scores and more stable results. The use of imbalance learning has shown significant improvement in the Average weight scores of the defect density results but less significant on the defect count results. Finally, using feature selection with weighted score of the defect density metric while it had no impact on defect count results. Thus, we conclude that using feature selection and imbalance learning with significant results. GATL helps by ranking modules based on the defect severity, which helps to direct focus and resources to the modules that need more testing.

Reference

1. Tang S., Huang S. Zheng V, E. Liu, C. Zong and Y. Ding, "A novel cross-project software defect prediction algorithm based on transfer learning," in Tsinghua Science and Technology, Vol. 27, no. 1, pp. 41-57, Feb. 2022.
2. Yuan X., Chen X.i and Mu Y. (2020), 'ALTRA: Cross-Project Software Defect Prediction via Active Learning and Tradaboost,' in IEEE Access, Vol. 8, pp. 30037-30049.
3. Gong L., Jiang S. and Jiang L. (2019), 'Tackling Class Imbalance Problem in Software Defect Prediction Through Cluster-Based Over-Sampling with Filtering,' in IEEE Access, Vol. 7, pp. 145725-14573.
4. Chakraborty T., and Chakraborty A.K. (June 2021), 'Hellinger Net: A Hybrid Imbalance Learning Model to Improve Software Defect Prediction,' in IEEE Transactions on Reliability, Vol. 70, no. 2, pp. 481-494.
5. Pan, C.; Lu, M.; Xu, B.; Gao, H. An Improved CNN Model for Within-Project Software Defect Prediction. *Appl. Sci.* **2019**, *9*, 2138. https://doi.org/10.3390/app9102138.
6. Zhang, Jie; Wu, Jiajing; Chen, Chuan; Zhang, Zibin; Lyu, Michael R. (2020), CDS: 'A Cross -Version Software Defect Prediction Model with Data Selection,' IEEE Access, 8 110059–110072. Vol. 11, pp. 2318-2326.
7. Ali et al C., (2023), 'Analysis of Feature Selection Methods in Software Defect Prediction Models,' in IEEE Access, Vol. 11, pp. 145954-145974.
8. Annisa., Riski , Rosiyadi, Didi , Riana, and Dwiza. (2020), 'Improved point center algorithm for K-Means clustering to increase software defect prediction', International Journal of Advances in Intelligent Informatics. 1, Vol. pp.328-339.
9. Bala Y.Z., Abdul Samat P. Sharif K. Y. and Manshor N. (2023), 'Improving Cross-Project Software Defect Prediction Method Through Transformation and Feature Selection Approach,' in IEEE Access, Vol. 11, pp. 2318-2326.
10. L. Qiao, X. Li, Q. Umer and P. Guo, "Deep learning-based software defect prediction", *Neurocomputing*, vol. 385, pp. 100-110, 2020.
11. Bhat., N.A., Farooq N.A. S.U. (2023), 'An empirical evaluation of defect prediction approaches in within-project and cross-project context', Software Qual J 31, PP. 917–946. https://doi.org/10.1007/s11219-023-09615-7.
12. Chinenye O., Anyachebelu. K. T. Abdullahi. M. U. (2023). Software Defect Prediction System Based on Decision Tree Algorithm. Asian Journal of Research in Computer Science, 16(4), 32–48.

13. Huang E., and Strigini L. (2023), 'HEDF: A Method for Early Forecasting Software Defects Based on Human Defect Mechanisms," in IEEE aAccess, Vol. 11, pp. 3626-3652, doi: 10.1109/ACCESS.2023.3234490.
14. Karpagalingam Thirumoorthy., Jerold John Britto J. (2022), 'A feature selection model for software defect prediction using binary Rao optimization algorithm, Applied Soft Computing,' Vol 131, 109737, ISSN 1568-4946, https://doi.org/10.1016/j.asoc.2022.109737.
15. A Iqbal and S. Aftab, "Prediction of Defect Prone Software Modules using MLP based Ensemble Techniques", International Journal of Information Technology and Computer Science, vol. 12, no. 3, pp. 26-31, 2020.
16. A A Saifan and L A. Abuwardih, Software DP Based on Feature Subset Selection and Ensemble Classification, vol. 14, no. 2, pp. 213-228, 2020.
17. Lin J., and Lu L (2021),.'Semantic Feature Learning via Dual Sequences for Defect Prediction,' in IEEE Access, Vol. 9, pp. 13112-13124
18. Mehmood et al E., (2023), 'A Novel Approach to Improve Software Defect Prediction Accuracy Using Machine Learning,' in IEEE Access, Vol. 11, pp. 63579-63597.
19. Miholca D.L., Tomescu V.I. and Czibula G. (2022), 'An In-Depth Analysis of the Software Features' Impact on the Performance of Deep Learning-Based Software Defect Predictors,' in IEEE Access, Vol. 10, pp. 64801-64818, doi: 10.1109/ACCESS.2022.3181995.
20. Majd, Amirabbas, et al. "SLDeep: Statement-level software defect prediction using deep-learning model on static code features." Expert Systems with Applications 147 (2020): 113156.
21. Mehta, S., Patnaik, K.S. Improved prediction of software defects using ensemble machine learning techniques. Neural Comput & Applic 33, 10551–10562 (2021). https://doi.org/10.1007/s00521-021-05811-3.
22. O. A. Qasem, M. Akour and M. Alenezi, "The Influence of Deep Learning Algorithms Factors in Software Fault Prediction," in IEEE Access, vol. 8, pp. 63945-63960, 2020, doi: 10.1109/ACCESS.2020. 2985290.
23. Li, Zhen, et al. "Software defect prediction based on hybrid swarm intelligence and deep learning." Computational Intelligence and Neuroscience 2021.1 (2021): 4997459.
24. Rathore S.S., Chouhan S.S. Jain D.K and Vachhani A.G. (June 2022), 'Generative Oversampling Methods for Handling Imbalanced Data in Software Fault Prediction,' in IEEE Transactions on Reliability, Vol. 71, no. 2, pp. 747-762.
25. Šikić L., Kurdija A.S. Vladimir K. and Šilić M. (2022), 'Graph Neural Network for Source Code Defect Prediction,' in IEEE Access, Vol. 10, pp. 10402-10415
26. Tong, Haonan, Shihai Wang, and Guangling Li. (2020), 'Credibility Based Imbalance Boosting Method for Software Defect Proneness Prediction,' Applied Sciences 10, no. 22: 8059.
27. H. Wang, W. Zhuang and X. Zhang, "Software Defect Prediction Based on Gated Hierarchical LSTMs," in IEEE Transactions on Reliability, vol. 70, no. 2, pp. 711-727, June 2021, doi: 10.1109/TR.2020.3047396.