IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)
An International Open Access, Peer-reviewed, Refereed Journal

MACHINE LEARNING APPROACHES FOR FORECASTING SOFTWARE FLAWS

Miss. Mehnaz M. Sheikh, Mr. Hirendra Hajare
M.Tech Scholar, Assistant Professor

Computer Science and Engineering,

Ballarpur Institute of Technology, Ballarpur, India

Abstract: Predicting software bugs before they happen is really significant for making sure that software works well and doesn't charge too much to fix. This paper is about using computer programs to forecast future software glitches by looking at past data. They tried out three different programs to see which one works best: Naïve Bayes, Decision Tree, and Artificial Neural Networks. They originate that these programs can predict software problems pretty precisely. They also looked at different kinds of cyber threats like backdoors, viruses, and exploits, using a dataset called UNSW-NB15. They used a method called feature selection to pick out the most important things to look at. Then they used a classification method called EC to figure out what kind of threat it is. Overall, they found that their method works really well for identifying and classifying threats accurately.

Index Terms - Software Defect, Software testing, Software bug, Prediction model, Artificial Neural Network, Software defect Prediction (SDP), Software Quality, Decision Tree (DT) , Machine Learning , Naïve Bayes(NB).

I. INTRODUCTION

The internet is growing quickly, with billions of users and vast amounts of data being generated every day. This growth brings challenges in data security and privacy, especially with increasing hacking tools and techniques. Intrusion Detection Systems (IDS) play a energetic role in identifying and answering to threats by monitoring network traffic for doubtful activity.
Traditional IDS methods face tasks in acquainting to the unique characteristics of IoT devices and in efficiently treatment the vast amounts of data. Machine learning algorithms offer a auspicious solution by automatically learning from data to detect and classify intrusions. However, existing IDSs still struggle with false alarms and identifying unknown attacks. Improved IDS systems based on machine learning can address these issues by continuously learning and adapting to new threats. This research focuses on developing an IDS system that combines feature selection and classification algorithms to improve accuracy. A novel VFS-EC algorithm is proposed to improve intrusion detection rates and to handle various challenges. The study's main contributions include:
1. Developing an effective IDS system capable of quickly and accurately detecting all types of attacks using cross-verified Artificial Neural Networks with Random Forest Classifiers (EC) and the VFS feature selection approach.
2. Using the UNSW-BoT Dataset to evaluate the proposed IDS system's performance and compare it with existing models.

In addition to intrusion detection, software bug prediction is another critical aspect of software development. Predicting faulty modules early can significantly improve software reliability, quality, and maintenance costs. Machine learning techniques, such as Naïve Bayes, Decision Trees, and Artificial Neural Networks, are commonly used in software bug prediction. This paper evaluates the capabilities of these classifiers using different datasets and comparison measures like accuracy, precision, recall, F-measures, and ROC curves. The paper also discusses related work, provides an overview of selected ML algorithms, describes datasets and evaluation methodology, presents experimental results, and concludes with future directions.

The number of active users on the internet has increased to 4.66 billion in recent days, conferring to the Global Internet Statistics Report generating more than 2 quintillion bytes of data per day. It demonstrates that the velocity of data access from various sources has accelerated dramatically, as has the development of techniques and hacking tools. As a result, data security and privacy are required to protect data from various intrusions or hostile attacks. Because of the speed of data and increased volume, traditional intrusion detection systems were unable to detect intrusions or assaults in a timely and efficient manner. Certain computational procedures, on the other hand, are difficult by nature to handle such data, necessitating advanced intelligent approaches and strong technologies. Intrusion detection systems, or IDS, play a grave role in identifying attacks. The IDS system will monitor network traffic in order to detect threats, attacks, or suspicious activity. When this type of activity is detected, it may send an alert to the appropriate administrator. A change of machine learning techniques can be used to efficiently deal with the infiltration. Different machine learning techniques can be used to efficiently manage and classify intrusions or attacks [1-3]. For more than two decades, Intrusion Detection Systems (IDS) have played a pivotal role in enhancing network and information system security, particularly in safeguarding smart IoT devices against various attacks. However, traditional IDS methods face challenges when applied to IoT owing to unique protocol stacks and architectural constraints. Consequently, new solutions are needed, such as hardware-based applications using network probes, albeit at a substantial resource cost. To address these challenges, researchers are turning to machine learning algorithms, which can effectively detect suspicious attacks by analysing network traffic. Despite the advancements, existing IDSs still struggle with high false alarm rates and the inability to identify unknown attacks. This has led to a focus on developing IDSs with lower false alarm rates and higher detection rates, with machine learning-based IDS showing promise due to its ability to extract useful information from large datasets. To contribute to this field, the study proposes a novel IDS system that combines feature selection with classification algorithms, leveraging the VFS-EC algorithm to enhance detection accuracy across various attack types. The effectiveness of the proposed system is evaluated using the UNSW-BoT Dataset, demonstrating improved categorization performance compared to existing models.

II. RELATED WORK:

Almost all computer software contains defects that introduced during the coding process. Over time, some of these flaws are being identified, often through quality assurance testing. The later those defects are being discovered and addressed in the development cycle, the higher the associated costs of fixing them [4; 50]. Consequently, early detection of flaws is paramount.

Testing, as a primary means of identifying flaws or defects, is crucial in software development! Myers et al. define software tests as executing a program to uncover errors. Testing happens across various levels like unit testing, function testing, system testing, regression testing, and integration testing, you know. Unit testing, mainly examines specific code sections and detects defects for cost savings, leading to cost savings, right? By concentrating on the most error-prone areas, we can save costs sooner than usual, which is essential.

The process of predicting parts of software susceptible to faults is known as Software Defect Prediction (SDP). SDP entails using measurements that are derived from sources like the source code and development process to determine if these metrics might offer insights into defects. Research indicates that testing-related tend to consume quite a significant portion, anywhere 50% to 80%, of total time [12]. Given this substantial, it is essential to focus the testing efforts on areas where defects are likely to arise, because this can possibly result in significant cost savings. SDP has been under study since the 1970s, originally simple equations with source code measurements as variables for prediction [17]. Subsequently, the focus of the prediction techniques shifted towards statistical analysis, expert estimations, as well as Machine Learning (ML) [8]. Among these approaches, ML has emerged as the most successful method [8, 22].

2.1 Data Mining with Machine Learning

Machine learning (ML) is a subset of artificial intelligence that focuses on enabling computer programs to learn from data. It attempts to reproduce the human learning process computationally, primarily by observing patterns, and based on those patterns to generate the whole [24 ML can be broadly divided into two main types: supervised learning and unsupervised learning. Supervised learning involves learning from labelled examples, where the outcome of each training sample is known [56, p. 40], while unsupervised learning involves learning from data without fixed outcomes. In this master thesis, especially in the classification of samples into two or more groups, the focus on supervised learning is important because the Problem Report (TR) contains information about files that have been found to be defective the existing database.

Different algorithm families developed a wide variety of ML algorithms for supervised learning, also known as classifiers, as evidenced by previous software defect prediction (SDP) research These algorithms include decision trees, classification rules, neural networks and probabilistic classifiers. Research has shown that in many cases the J48 decision tree outperforms the OneR algorithm. Again, Shafital. [48] investigated the use of the ZeroR classification rule algorithm alongside OneR, and found that OneR generally performs better. When predicting the value of the majority group, OneR outperformed ZeroR [56, p. 459]. Arisholm and so on. conducted two separate experiments [1, 2] using meta-learners such as Decorate and AdaBoost with a J48 decision tree. Decorate is shown to exhibit good performance on small data sets and slightly better on large data sets. However, no specific definitions were given for "small" and "large" datasets.

2.2 Association Rule Mining

Software metrics refer to quantitative measurements that provide numerical values or markers for the attributes of the measured object [18]. An attribute in this context represents an object or characteristic of an object such as length, life span, or cost. The entity itself can be unique and can include the source code of an application or function in the software development process. Software metrics can be organized into different families, each of which focuses on different aspects of software development.

- **Static code metrics**

Static code parameters are measurements extracted directly from the source code itself, including common metrics such as signal lines (LOC) and cyclonic complexity Source lines (SLOC) include various metrics, such as physical LOC (SLOCP). that counts total lines, blank LOC in the case of blank lines (BLOC), annotated lines f comment-LOC (CLOC) refer to lines with LOC (SLOC-L) logical comments so [41]. The cyclonic complexity number (CCN), also known as the McCabe metric, linearly measures the complexity of a module's decision process by counting the number of independent paths each time a path's control flow splits, such as if, for, while, case, means, &&, ||, or ? in source code [28]. Other static code parameters include compiler instruction count and data declaration count.

Feature selection methods in machine learning generally fall into two categories: packaging and filtering. Wrappers use the same ML algorithm to select features, finding the value of different features. However, filters use heuristics based on data characteristics to consider different features [20]. Filters are useful for their fast performance and are compatible with any ML algorithm. They can handle large feature sets well. Generally, however, they must stay clear of the problem of classification. Wrappers, although versatile and applicable to any classification problem, require the feature selection process to be repeated for each algorithm used for prediction, since the same algorithm is used for feature selection and distribution.

III. PROPOSED METHODOLOGY

In this phase, we look into the trials of the proposed Intrusion Detection System (IDS) version VFS-EC. The usual workflow is carefully illustrated in Fig. 1 , which offers a detailed review of the methodology. Initially, the us-NB15 statistics set is loaded into the system, forming the idea for next evaluation.

Once the records are received, numerous preprocessing steps are done to make sure facts brilliant and readiness for analysis. This step is essential for cleaning and organizing the statistics set, putting off any inconsistent or inappropriate statistics that might affect the accuracy of the version

In order to in addition improve the accuracy of the IDS version, a function selection process is implemented the usage of the proven typical choice (VFS) set of rules. This set of rules intelligently identifies and selects the most relevant functions from the facts set, streamlining the quest manner and improving universal overall performance

Once the function is chosen, the Novel EC algorithm is used to classify the irregularities in the statistics set. This set of rules is in particular designed to correctly discover and classify diverse kinds of inputs, imparting treasured insights into potential protection dangers

On the prediction aspect, the IDS machine now not humblest detects abnormalities but additionally identifies the form of attack being accomplished. This capability is vital to rapidly deal with security breaches and increase suitable countermeasures.

Finally, the performance of the newly developed VFS-EC machine is thoroughly evaluated using numerous overall performance metrics. These measures include accuracy, precision, bear in mind, and F-measures, among others. In addition, the overall performance of the VFS-EC model is compared with the present IDS model to show its superiority.

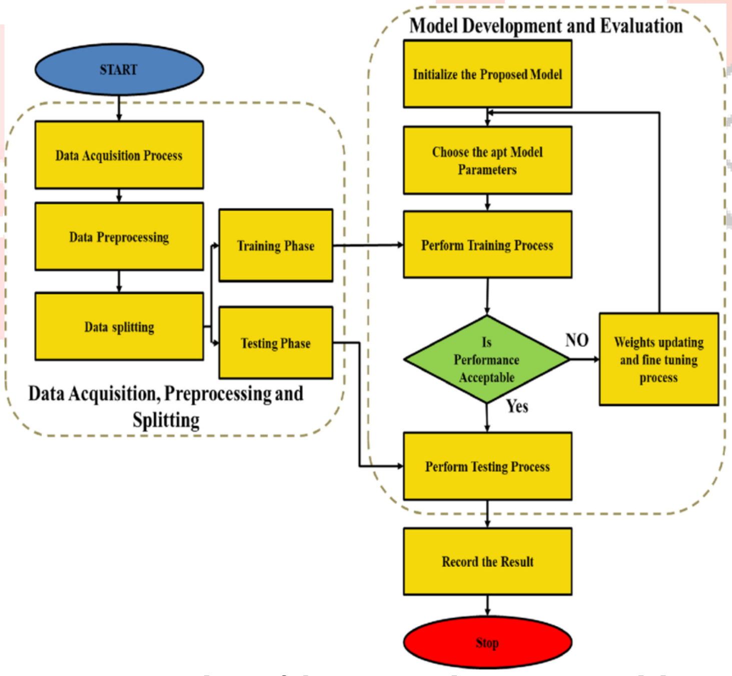

Model Development and Evaluation
START
Data Acquisition Process
Data Preprocessing
Data splitting
Training Phase
Testing Phase
Data Acquisition, Preprocessing and Splitting
Initialize the Proposed Model
Choose the apt Model Parameters
Perform Training Process
Is Performance Acceptable
NO
Yes
Weights updating and fine tuning process
Perform Testing Process
Record the Result
Stop

Figure 1. Flow of the proposed VFS-EC Model

Here, “1” is denote feature is nominated and second “0” is represents feature is rejected.

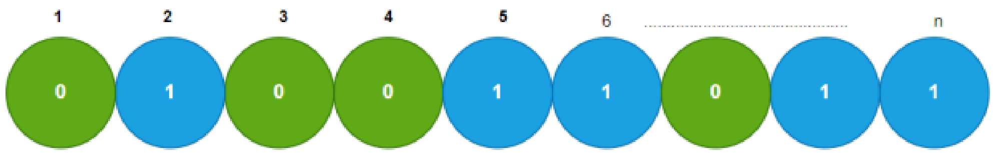


Figure 2. Feature Selection

3.1 VFS Algorithm for Feature Selection

Here, we provide a complete explanation of the proposed method. Feature selection is important for improving learning performance, reducing computational complexity, and building efficient classification models. A validated feature selection (VFS) algorithm is used to identify suitable features in the data set with the aim of optimizing the feature subset for analysis

Typically, each feature in the VFS algorithm is represented as a binary vector of N entries, where N corresponds to the total number of features in the data set. A value of 0 indicates that the feature is not selected, while a value of 1 indicates that the feature is selected for inclusion in the subgroup.

Our proposed method simplifies the process to a single equation by simplifying sine and cosine with additional parameters. Rather than switch between two brands, we recommend always choosing one brand. Since sine and cosine functions yield values from 1 to +1 in the interval [0, 2π], the choice between them is particularly arbitrary. Therefore, adopting a single channel does not degrade the performance of the algorithm.

To strike a balance between exploration and application, we introduce a new equation that includes the random variables C and r1 . This facilitates effective transitions between detection and control methods. Equation (1) below shows the updated position, which ensures that the algorithm is efficient enough to lead to a search location.

$$X(i,j)_{t+1} = \quad 1- [s-t^* (\frac{s}{\max\ iter})]\ ^*X(i,j)_t + [s-t^* \frac{s}{\max\ iter}]\ ^*\gamma \quad \text{if } C< s1$$

$$\text{where } s1=s=t^*s/\max \text{ iter}$$

$$P_1(j)_t + [s-t^* \frac{s}{\max\ iter}]\ ^*\sin(dist)^*|weight * P_1(j)_t - X(i,j)_t| \quad \text{if } C>= s1 \qquad \textbf{Eq .1}$$

In this equation C is defined as the product of the constants b, d, where b usually takes integer values in the range [1, 5] S1 represents the field of advance locations or direction of motion, where dist means random variable from 0 to 2π, also the weighting variable representing the variance between the current position and its previous position is a random value. The d parameter is used to select different search approaches, whether involving sine functions or not, based on different random values. In addition, the task of d weights fixes a value greater than 1, thus influencing the selection process.

Algorithm 1: Verified Selected Features (VFS) .

```
EXPLANATIONS:
- Number of searches
- Dataset validation
- The snowflake
- Maximum number of iterations
- The upper limit
- Lower limit

SOURCES:
- Vector of 20 optimal solutions

1. Start a general search (X) .
2. Calculate the cost function of each agent (fit), and select the agent with the best position (Best_pos).
3. Set t = 2
4. WHEN a ( t ≤ max_iter ) IS
   a. set s = t * (s / maximum) .
   b. For each factor (X), from dimension (j) DO
      - Obtained random variable dist ∈ [0, 2π].
      - Random variable loading
```

```
        - Create a random variable $C = b * rand()$ .
        - update the rank of the result:
            - If $(C<s)$ THEN
                - Set m = Get_rand_position() % Get random position [lb, ub] from search position.
                - Transform $X(j) = (1 - r1) * X(j) + (r1) * m$
            - however
                - update X(j) = Best_pos(j) + s*(sin(dist)) *abs(weight* Best_pos(j) - X(j));
            - ENDIF
    c. Calculate a new cost function (fit) for each agent (X), and determine the agent with the best position (Best_pos).
    d. Increase t
5. END TIME A

END ALGORITHM
```

Algorithm 1 shows the VFS algorithm, which is described in detail here. Initially, the algorithm randomly determines the locations of solutions under the search area. It then loops for the maximum number of iterations to find the best feature subset.

At each iteration, the algorithm calculates the fitness score (Fit) for each solution using the specified cost function, which is usually performed to evaluate the effectiveness of the feature subset and then determines the best solution (Best_pos) at of the population based on the highest fitness score.

The solution is updated using Equation (1). This equation simplifies both diversification and intensification processes by controlling for two parameters, (s_1) and (d). The value of (s_1) decreases with each iteration, while (d) is a random number ranging from 0 to 1. If (d < s_1), then diversification is pursued; otherwise, they want to reinforce it. These update parameters ensure a balanced transition between the two methods.

The variables used in the algorithm are proportional to the number of objects in the data set. This variable is constrained in [0, 1], where a value close to 1 indicates that the corresponding feature is likely to be selected for classification

In the fitness calculation for individual solutions, each variable is a determining boundary for inclusion. In particular, Equation (4) defines the fitness score (f_{ij}) for a feature based on its value (X_{ij}), with a threshold of 0.5:1.
[f_{ij} = \begin {entry} 1, & \text{if } X_{ij} > 0.5 0, & text{else} end{entry}].

Where the update position of the solution violates the constraints ([0, 1]), a simple truncation rule is used as v.

3.2 CLASSIFIER:

The K-Nearest Neighbours (KNN) algorithm is applied in software bug prediction using machine learning by first selecting suitable features from software metrics datasets and then using these features such as lines of code, complexity measures, and developer experience, traditional KNN classifiers. During training, the KNN algorithm memorizes the entire dataset and calculates the distance between data points based on the chosen distance metric, usually Euclidean distance If a new software module needs to be classified as faulty or not error a, KNN identifies K nearest neighbours from the training set. The implementation is evaluated using metrics such as accuracy, precision, recall, and F1-score to verify its effectiveness in predicting software errors which great accuracy.

K-Nearest Neighbor (KNN) classification works by calculating transition weights based on the proximity of data points through a trial-and-error process. It's like trying different weights until you find the best match. In this study, KNN is used as part of the fitness task because it is actually good at sorting topics into classes.

Now, let's talk fitness business. Imagine trying to make the best choices from several options. It's like picking players for a sports team – you want them to be right but also not too much. Thus, the fitness function examines each feature set and scores it. These scores are calculated using a formula (Equation 3) that considers the number of classifier errors (classification error), the number of selected features (D), and the total number of features (N) There were also two parameters α and β, which tells us that the accuracy, . And how important are the number of parts?

$$\text{Fitness} = S * 0.01 * E_R(D) + C * \frac{|M|}{|N|} \qquad \text{Eq .3}$$

The mutation agent introduces new traits not present in the ancestor, helping to produce new individuals. It takes data representations such as binary, real, or integer formats and includes many mutation methods. Mutations generally involve selecting one or more bits at random and changing their values based on a predetermined probability. Equation 4 is a representation of the mutation process, which is similar to that shown

$$P_c^{v+1} = \text{Mutation}(P_c^{v}) \qquad \text{Eq. 4}$$

The analysis used the VFS algorithm for feature selection (as shown in Figure 2). Initially, the algorithm initializes the location, evaluates the search agent with the objective function, and updates the optimal solution and location again. This approach helps explore possible solutions, and can identify the most important features.

3.3 EC ALGORITHM USED TO CLASSIFY IDENTIFIED INTRUSION

This integrated model uses artificial neural network (ANN) modeling along with random forest segmentation and k-fold cross-validation. Unlike many other models, this method does not require hyperparameter adjustments, so it is particularly suitable for large data sets with low memory requirements The quality of random forest classification is evaluated by cross-validation, a remodeling is considered. The EC method improves the accuracy of the intrusion classification, contributing to the development of more robust algorithms. A random forest segmentation with multiple decision trees predicts outcomes by collectively voting for the best option. Overall, it outperforms individual decision trees.

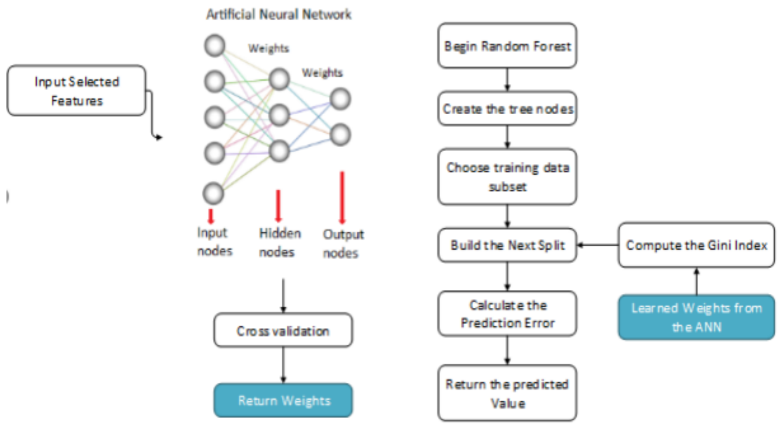


Fig 3: EC Algorithm

The feature selection process continues its execution through the search space without any particular stop. Different generation strategies and analytical methods influence the selection of reference decisions. Stop criteria based on subgroup generation require reaching a maximum number of objects or generations. Based on the search methods, the feature selection stops if the new function does not yield a good subset or when an optimal subset is found according to the search criteria, the feature selection process stops by selecting the features subset correctly are returned to the machine learning process. A generalized algorithm for feature selection is described below, representing the data set (D), the search technique (S), the evaluation measure (M), the stop criterion (Sc), and the optimal subset of features (F optimal). is, updating the best subset (F optimal) until the stopping criterion is met and finally the best subsets are returned.

The aim of the study is to investigate and evaluate three supervised machine learning (ML) algorithms: Naive Bayes (NB), Artificial Neural Network (ANN), and Decision Tree (DT), especially prediction focusing on their performance accuracy and power over software errors. The study provides a comparative analysis of these selected ML algorithms to demonstrate their effectiveness in this context. Supervised machine learning algorithms work through implication functions that distinguish correlations and dependencies between known inputs and outputs of labelled training data This function enables the prediction of effects values for additional inputs based on patterns obtained from training data.

3.4. Naive Bayes (NB)

Naive Bayes is a forthright and well-organized probabilistic classifier based on Bayes theorem with the assumption of independence between features. NB is not an algorithm but a family of algorithms with the common principle of assuming the class is self-governing of the presence or absence of a particular object or of other objects. Naive Bayes (NB) is a popular machine learning algorithm used in software bug prediction. In software bug prediction, Naive Bayes is often used as a classification algorithm. Its implementation determines whether a software module or component is prone to faults based on different features and attributes.

Naive Bayes algorithms are based on the Bayes theorem and make a strong assumption of feature independence, which means that each feature contributes independently to the probability of a particular outcome (in this case the presence or absence of errors) Despite this simplifying assumption, none of Bayes generally works well in practice is known for its simplicity, efficiency and efficiency in large data applications in

Naive Bayes in software bug prediction is trained on historical data including elements extracted from software artifacts (such as source code criteria, complexity measures, and code churn), labels indicating whether an artifact is bug-free or not Once trained once, then new software or unknown artifact values using the Naive Bayes model Can be predicted based on potential errors Overall, Naive Bayes is a valuable tool in software bug forecasting, providing insight into the likelihood of software artifacts and bugs and helping software developers prioritize testing and maintenance efforts

3.5. Artificial Neural Networks (ANNs)

Artificial neural networks are computational models inspired by organic neurons. ANNs are nonlinear classifiers that can model complex relationships between inputs and outputs. They have an interconnected set of controls called neurons that organize to produce outputs. Each connection between neurons carries a signal, and each neuron calculates its output using a nonlinear function of the sum of all inputs. Artificial neural networks (ANNs) have indeed been used in software bug prediction tasks. In this context, ANNs are used to analyse various software objects, such as source code files or modules, and determine whether they contain bugs or How ANNs are commonly used in software bug prediction is:

1. Data Collection and Maintenance: Collects historical data on software artifacts and labels indicating their error status (either error or no error) Attributes are extracted from these artifacts, which software metrics, code complexity measures, and others are included Relevant features can also be included. The data are then pre-processed and divided into training and test sets.

2. Network Architecture Design: Designed neural network. It specifies the number of layers, the number of neurons in each layer, and the activation functions used. The input layer of the network retrieves the omitted features, and the output layer provides predictions of possible errors in software artifacts

3. Training of the network: The neural network is trained using the training dataset. During training, the network learns how to map the inputs to the corresponding error codes. This is usually done using backpropagation and other optimization algorithms, where the weights of the network are adjusted iteratively to reduce the prediction error.

4. Validation and testing: After training, the performance of the neural network is tested using a separate validation data set. This helps to examine the generalizability of the model to unseen data and its functionality. Various analytical parameters such as precision, accuracy, recall, and F1-score are used to measure the performance of neural networks.

5. Prediction: Once a neural network is trained and validated, it can be used to predict the probability of bugs in new software products. Developers can insert additional features of a new product into the trained model, and the web page comes up with a prediction of whether the product is likely to be faulty

Overall, ANN offers a powerful and flexible approach to software defect prediction, enabling the automatic identification of potential problem areas in software products by using ANN, organizations can improve their software quality control and reduce the potential for defects in their software products.

3.6. Decision Tree (DT)

Decision tree is a extensively used learning technique in data mining. It uses a planning and prediction system in which the observation of the item acts as a branch to reach the target value of the item in the document. The DT consists of decision nodes, with many branches, and leaf nodes representing the final decision. The decision process involves a tree structure based on the features of the input data. Decision trees are often used in software bug forecasting because they provide an interpretable model and can handle statistical and classification data efficiently as here are some common uses of Decision Trees in software bug forecasting:

1. Data Collection and Preparation: As with other machine learning techniques, the first steps include historical data collection of software artifacts and labels indicating their bug status (bug or bug-free) Features Excluded from these artifacts, such as software metrics, code complexity measures, other related features and features. The data are then preprocesses and divided into training and test sets.

2. Model Training: Decision trees are trained using a set of training data. During training, the algorithm divides the data set iteratively based on the factor that provides the best separation between samples with and without errors This process continues until certain stability criteria are met, hear as a maximum tree depth or a minimum number of leaf nodes to obtain.

3. Model Analysis: After a decision tree is trained, its performance is tested using a separate validation data set. This helps in assessing the generalizability of the model and its performance on unobserved data. Various analytical metrics such as accuracy, precision, recall, and F1-score are used to measure the performance of decision tree models.

4. Interpretation: One of the main advantages of decision trees is that they can be interpreted. The resulting tree structure can be easily visualized and understood by domain experts, providing insight into factors affecting software error prediction.
Prediction: In addition to training and evaluation, decision tree modeling can be used to predict the probability of bugs in new software products. Developers can input features of a new product into the trained model, and the decision tree produces predictions of the likelihood of artifacts being faulty

Overall, decision trees offer a flexible and interpretable method for software defect prediction, making them especially useful for applications where transparency and common sense are important by using decision trees implementation, organizations can improve their software quality control and reduce potential defects in their software products.

Specifically, naïve Bayes relies on probabilistic cognitive, artificial neural networks mimic biological neural networks to capture composite associations, and decision trees formulate hierarchical decision rules to allocate data Learning objectives to evaluate and compare the performance of such these algorithms are performed in software fault prediction tasks are to provide insights into their effectiveness and suitability for this particular application

IV. EXISTING METHODOLOGY

4.1 Existing System

Several existing systems use machine learning (ML) techniques to predict software bugs. Here are a few examples:

1. **NASA Metrics Data Program (MDP):** This software is one of the first and best-known data sets for error prediction analysis. It contains software metrics data collected from various NASA projects, as well as a list of whether each module contains errors. Researchers have used this dataset to develop and evaluate ML models for error prediction.

2. **PROMISE Repository:** The PROMISE repository provides a collection of data specially collected for software engineering research. These data sets include various aspects of software development including bug prediction. Researchers often use data sets from the PROMISE repository to benchmark and compare different ML algorithms to determine errors.

3. **Grecko and Madey ski Dataset:** This dataset compiled by Jureczko and Madeyski contains software metrics data from open source projects with labels indicating bug proneness It has been widely used in bug prediction research and support in ML-based bug development prediction models Given.

4. **Bug Prediction Challenge Datasets:** Many bug prediction challenge datasets have been released as part of research competitions over the years. These datasets typically contain software metrics data from real-world projects, with labels indicating a bug. Researchers use this list to evaluate the performance of their ML models in a competitive environment.

5. **Commercial tools:** Some companies offer commercial error prediction software using ML techniques. These tools are often included in software development environments to give developers insight into potential bugs in their code. Examples include Microsoft's Code Defect Prediction Tool and IBM's Rational Software Analyzer.

These existing frameworks and datasets allow researchers and practitioners interested in applying ML methods to software defect prediction By using these resources, organizations can improve the quality of their software improve processes and reduce the possibility of defects in their software products.

4.2 Limitations of Existing System

Since I have no specific information about the existing framework you describe, I will provide some general boundaries that can be attached to software bug prediction systems These limitations may vary depending on method and techniques the method of production. Here are a few general restrictions:

1. Limited Feature Set: Existing systems are using limited features or metrics to predict faults. This may overlook important sources of errors. The inclusion of additional relevant parameters can improve the accuracy and efficiency of forecasts.

2. Lack of data: The quality and fullness of the training system data can significantly affect its performance. Irregularities in the historical data, lack of standards, or incorrect error records can lead to biased or unreliable forecasts.

3. Challenges in Feature Selection: It is important to select more useful and relevant features for accurate error prediction. If the existing system lacks effective selection instruments, it may add unnecessary or unnecessary features, resulting in reduced forecast accuracy or overfitting

4. Insufficient training data: The amount and type of training data can affect the system’s ability to generalize and make accurate predictions. If the existing system has limited or imbalanced training data, it may struggle to capture patterns and underlying processes related to error occurrence

4.3 Proposed System

Based on the limitations of the existing system, here's a proposed system for software defect prediction:

1. **Enhanced feature set:** Expand the feature set used for defect prediction by incorporating additional relevant metrics and attributes. This can include code complexity, code churn, code coverage, developer experience, historical defect data, and any other domain-specific factors that can influence defect occurrence.
2. **Robust data preprocessing:** Implement thorough data preprocessing techniques to handle missing values, outliers, and data inconsistencies. This includes imputation methods for missing data, outlier detection and treatment, and normalization or scaling of the data to ensure consistency and improve model performance.
3. **Advanced feature selection:** Employ advanced feature selection techniques to identify the most informative and relevant features for defect prediction. This can include correlation analysis, information gain, feature importance analysis using machine learning algorithms, or domain expertise-driven feature selection methods.
4. **Ensemble modeling:** Utilize ensemble learning techniques to combine multiple machine learning models for defect prediction. Ensemble methods such as random forests, gradient boosting, or stacking can help improve prediction accuracy and reduce the impact of individual model biases.
5. Incorporate cross-validation: Apply cross-validation techniques during model training and evaluation to obtain more reliable and robust performance estimates. This involves splitting the data into multiple subsets, training and testing the model on different subsets, and aggregating the results to assess the model's generalization ability.

6. **Adaptability and incremental learning:** Design the system to handle evolving software by implementing mechanisms for incremental learning. This allows the system to incorporate new data and adapt the model over time without retraining the entire system from scratch.
7. **Explainable models:** Select machine learning algorithms that offer interpretability, such as decision trees or logistic regression. This enables better understanding and explanation of the factors contributing to the predictions, increasing the system's transparency and trustworthiness.
8. **Resource allocation optimization:** Develop algorithms or rules to optimize the allocation of testing and debugging resources based on the predictions. Consider incorporating cost-sensitive learning techniques that factor in the potential impact and severity of defects to prioritize efforts effectively.
9. **Continuous improvement and monitoring:** Implement a feedback loop to continuously monitor the performance of the system, collect new data, and periodically retrain the models. This ensures that the system remains up to date and adapts to changes in the software and defect patterns.

4.4 Advantages of Proposed System

The proposed system for software defect prediction offers several advantages over the existing system. Here are some key advantages:

1. Improved prediction accuracy: By expanding the feature set, implementing advanced feature selection techniques, and using ensemble modeling, the proposed system can enhance the accuracy of defect predictions. The inclusion of applicable metrics and the combination of multiple models help capture more complex patterns and dependencies, leading to more reliable predictions.
2. Better adaptability to evolving software: The proposed system incorporates mechanisms for incremental learning, allowing it to handle evolving software systems. By incorporating new data and adapting the models over time, the system can maintain its predictive performance as the software evolves, ensuring its relevance and effectiveness.
3. Enhanced resource allocation: The system optimizes resource allocation by considering the predictions and severity of flaws. By efficiently allocating testing and debugging resources, the proposed system helps prioritize efforts, leading to more effective defect detection and resolution. This can result in better overall software quality and reduced development and maintenance costs.
4. Interpretable predictions: By selecting machine learning algorithms that offer interpretability, the proposed system provides explanations for the factors contributing to defect predictions. This enhances the system's transparency and trustworthiness, allowing developers and stakeholders to understand and validate the predictions, leading to better decision-making.
5. Nonstop improvement and monitoring: The proposed system incorporates a feedback loop that allows for continuous monitoring of the system's performance. By collecting new data and periodically retraining the models, the system can adapt and improve over time. This ensures that the defect prediction capabilities remain up to date and aligned with the changing software and defect patterns.
6. Reduced false positives and false negatives: Through the combination of progressive feature selection techniques, ensemble modeling, and improved accuracy, the proposed system aims to minimize false

positives (predicting defects that don't occur) and false negatives (failing to predict actual defects). This helps avoid unnecessary rework and ensures that potential issues are identified and addressed proactively.

Overall, the proposed system offers improved prediction accuracy, flexibility, resource allocation, interpretability, and continuous upgrading compared to the existing system. These advantages contribute to enhanced software quality, reduced development costs, and more efficient allocation of testing and debugging resources.

V. UML Description

5.1 Use Case Diagram

Use case diagrams act as visual representations of system functionality from the users' perspective. They play an important role in identifying the types of roles involved, which may be users, systems, or external agencies, and how they interact with the system Provide scenarios illustrating those interactions this provides a clear understanding of the system actions and roles played by various tasks Emphasis is placed on required activities or tasks. Overall, functional diagrams provide a comprehensive overview of system operations and help stakeholders understand the scope and objectives of the system.

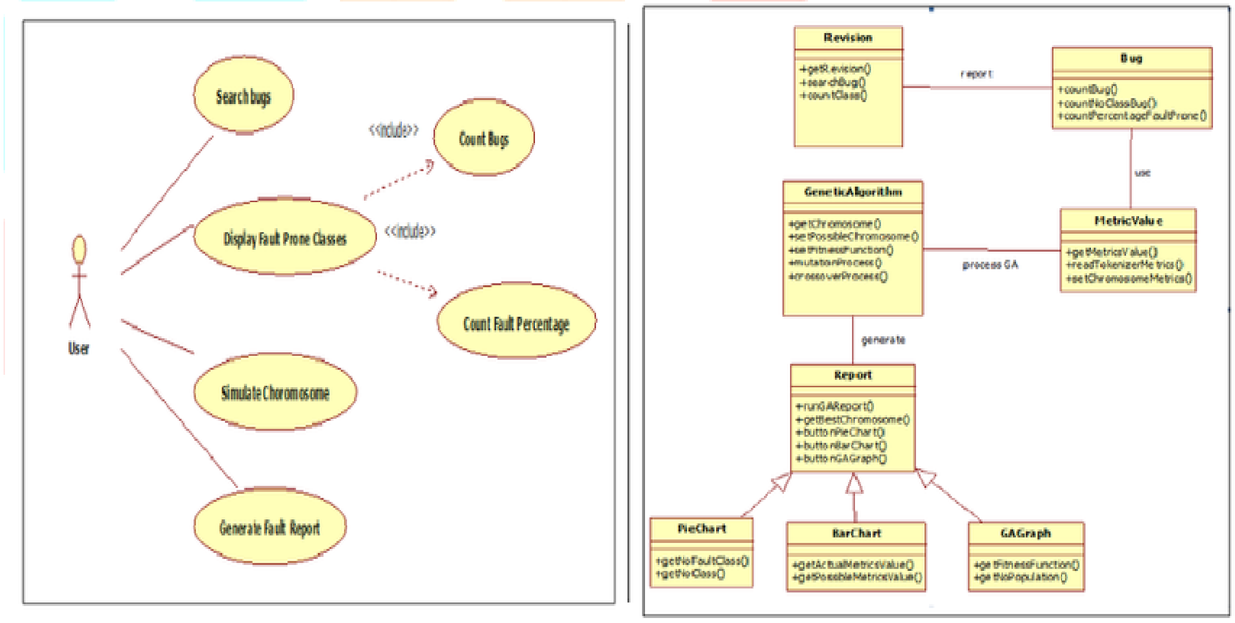


Figure 4. Use Case Diagram a software defect prediction system

5.2 Class Diagram

A study diagram is a visual representation that provides insight into the static structure of a system. They help visualize the structure of the system by identifying the component classes and their properties and methods, and class diagrams show the relationships between these classes, such as associations, dependencies, synthesis, and integration. By presenting this information in graphical form, class diagrams provide a clearer understanding of the structure of system classes, how they interact with each other and are a valuable tool for them developers and designers for visualizing the system, planning its execution, ensuring accuracy and scalability. Overall, class diagrams play a important role in interactive the static structure of the system and facilitating effective teamwork among sponsors involved in its development.

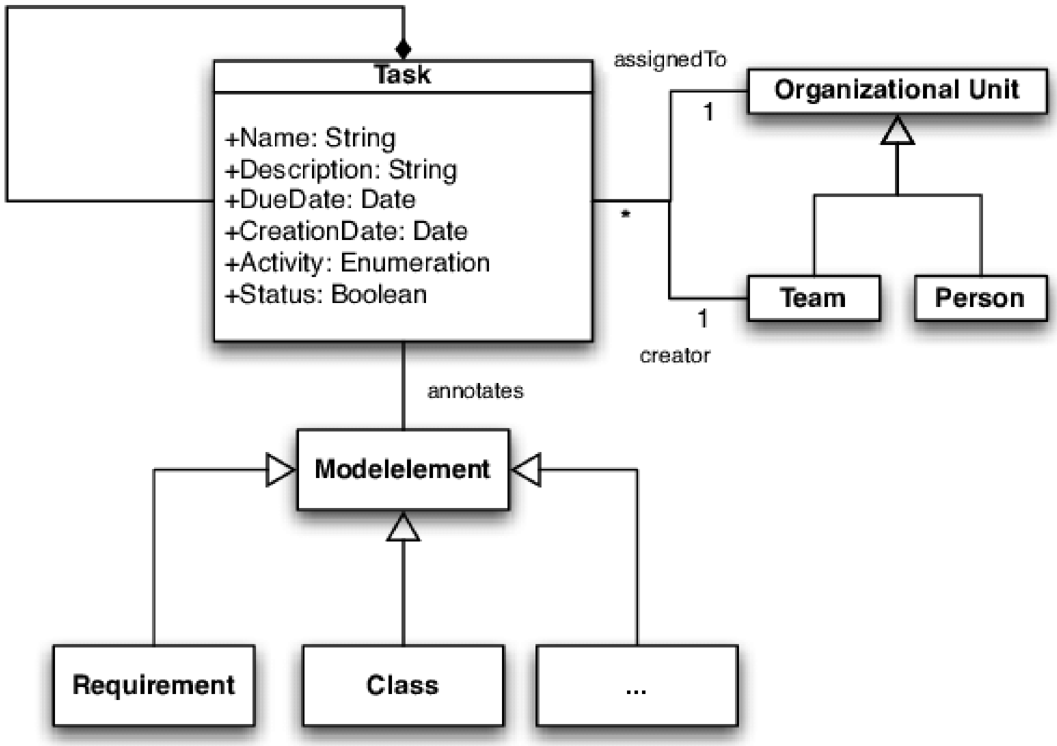
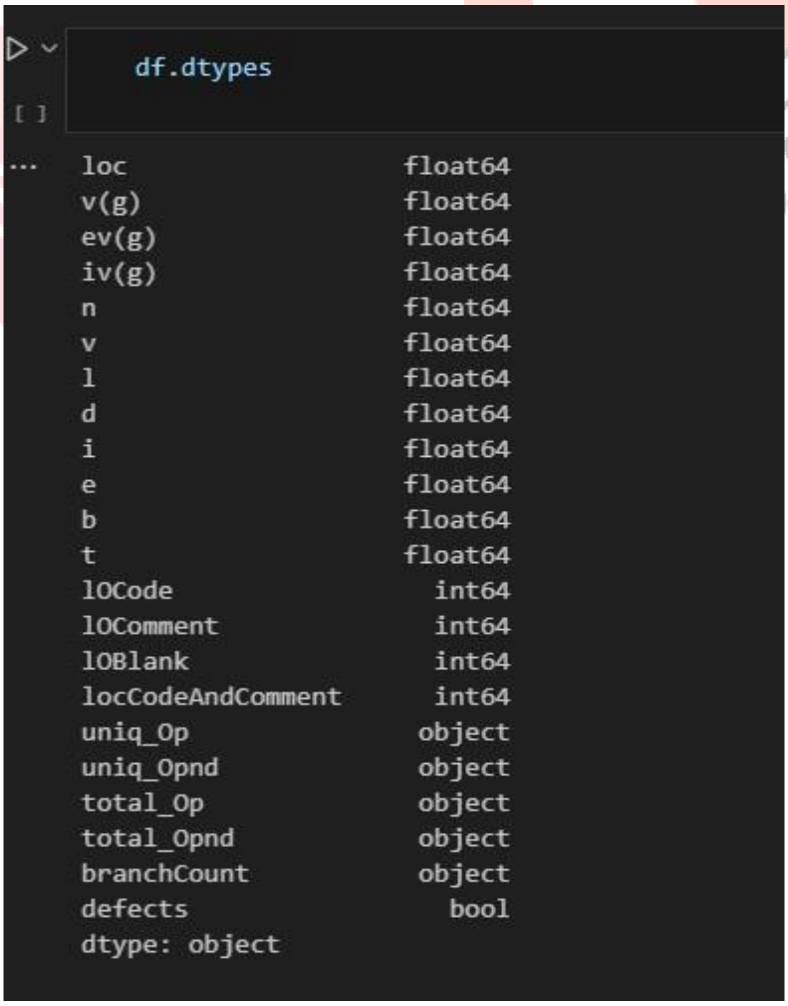


Figure 5. Class Diagram a software defect prediction system

VI. RESULT

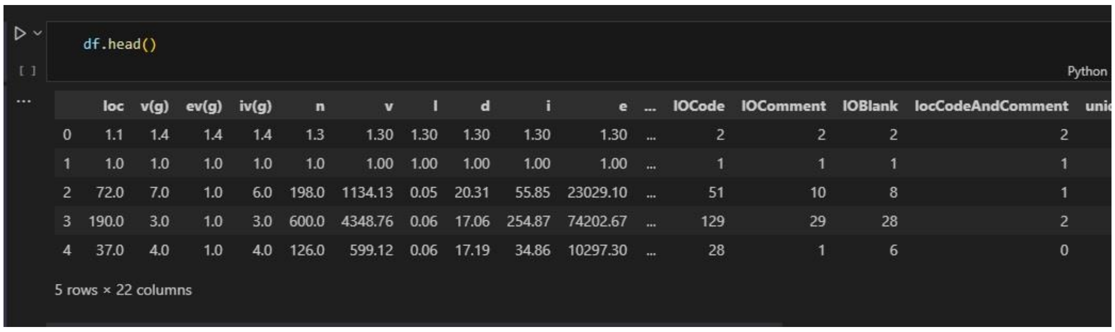
6.1 Screen shots



```
df.dtypes

loc                  float64
v(g)                 float64
ev(g)                float64
iv(g)                float64
n                    float64
v                    float64
l                    float64
d                    float64
i                    float64
e                    float64
b                    float64
t                    float64
lOCode                 int64
lOComment              int64
lOBlank                int64
locCodeAndComment      int64
uniq_Op               object
uniq_Opnd             object
total_Op              object
total_Opnd            object
branchCount           object
defects                 bool
dtype: object
```

Figure 6. Dataset for Feature of a software defect prediction system.

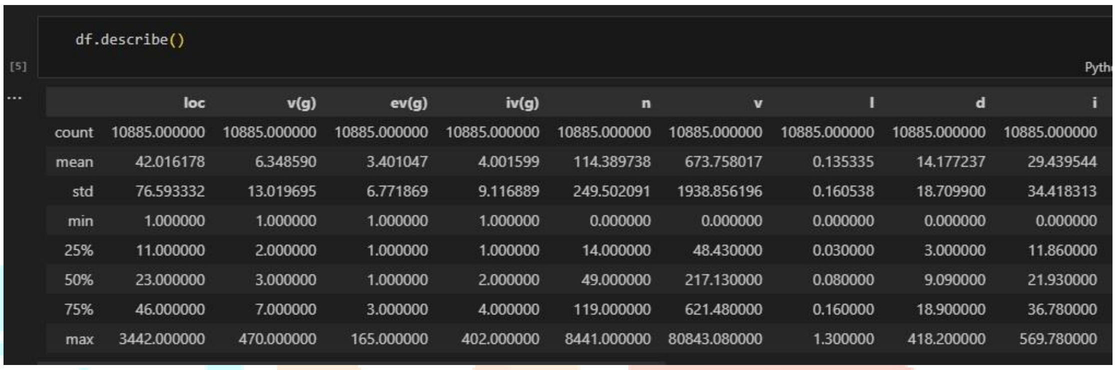


```
df.head()
```

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	...	lOCode	lOComment	lOBlank	locCodeAndComment	uni
0	1.1	1.4	1.4	1.4	1.3	1.30	1.30	1.30	1.30	1.30	...	2	2	2	2	
1	1.0	1.0	1.0	1.0	1.0	1.00	1.00	1.00	1.00	1.00	...	1	1	1	1	
2	72.0	7.0	1.0	6.0	198.0	1134.13	0.05	20.31	55.85	23029.10	...	51	10	8	1	
3	190.0	3.0	1.0	3.0	600.0	4348.76	0.06	17.06	254.87	74202.67	...	129	29	28	2	
4	37.0	4.0	1.0	4.0	126.0	599.12	0.06	17.19	34.86	10297.30	...	28	1	6	0	

5 rows × 22 columns

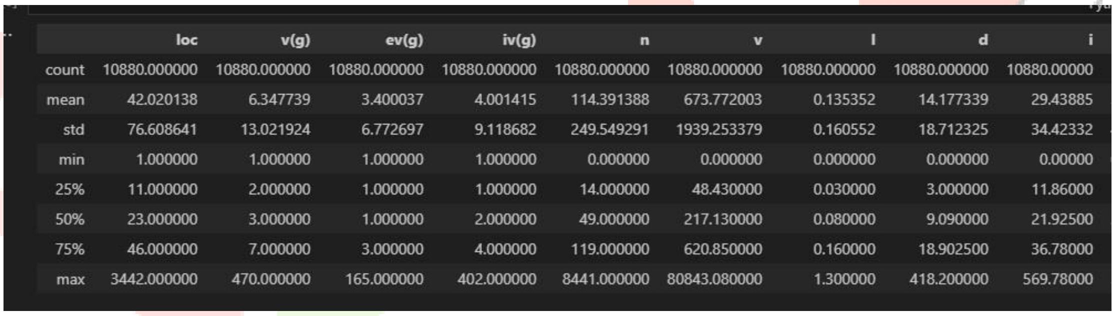
Figure 7. Dataset-1 for Feature of a software defect prediction system.



```
df.describe()
```

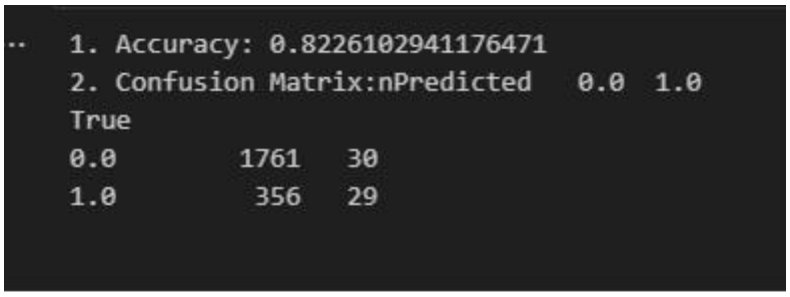
	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i
count	10885.000000	10885.000000	10885.000000	10885.000000	10885.000000	10885.000000	10885.000000	10885.000000	10885.000000
mean	42.016178	6.348590	3.401047	4.001599	114.389738	673.758017	0.135335	14.177237	29.439544
std	76.593332	13.019695	6.771869	9.116889	249.502091	1938.856196	0.160538	18.709900	34.418313
min	1.000000	1.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	11.000000	2.000000	1.000000	1.000000	14.000000	48.430000	0.030000	3.000000	11.860000
50%	23.000000	3.000000	1.000000	2.000000	49.000000	217.130000	0.080000	9.090000	21.930000
75%	46.000000	7.000000	3.000000	4.000000	119.000000	621.480000	0.160000	18.900000	36.780000
max	3442.000000	470.000000	165.000000	402.000000	8441.000000	80843.080000	1.300000	418.200000	569.780000

Figure 8. Dataset-2 for Feature of a software defect prediction system.



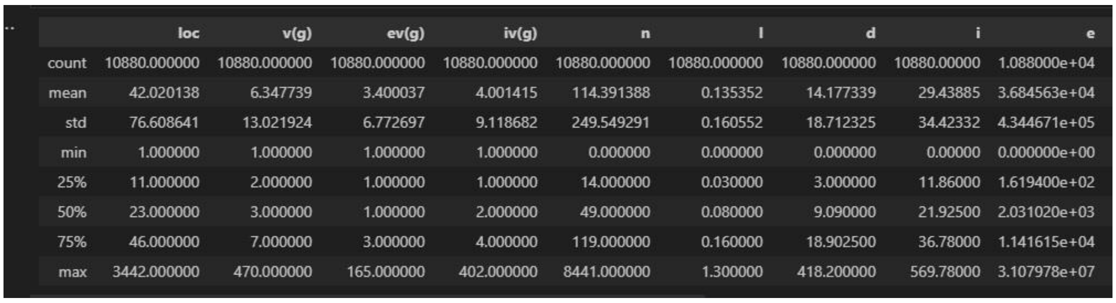
	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i
count	10880.000000	10880.000000	10880.000000	10880.000000	10880.000000	10880.000000	10880.000000	10880.000000	10880.00000
mean	42.020138	6.347739	3.400037	4.001415	114.391388	673.772003	0.135352	14.177339	29.43885
std	76.608641	13.021924	6.772697	9.118682	249.549291	1939.253379	0.160552	18.712325	34.42332
min	1.000000	1.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.00000
25%	11.000000	2.000000	1.000000	1.000000	14.000000	48.430000	0.030000	3.000000	11.86000
50%	23.000000	3.000000	1.000000	2.000000	49.000000	217.130000	0.080000	9.090000	21.92500
75%	46.000000	7.000000	3.000000	4.000000	119.000000	620.850000	0.160000	18.902500	36.78000
max	3442.000000	470.000000	165.000000	402.000000	8441.000000	80843.080000	1.300000	418.200000	569.78000

Figure 9. Dataset-3 for Feature of a software defect prediction system.



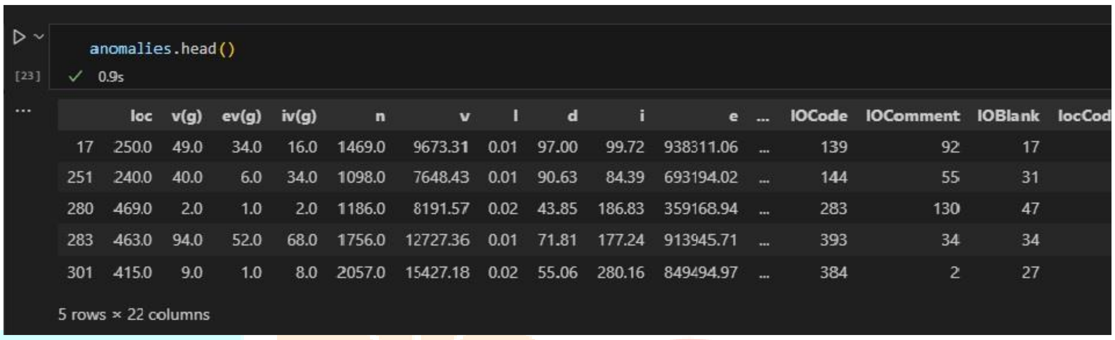
```
1. Accuracy: 0.8226102941176471
2. Confusion Matrix:nPredicted   0.0  1.0
True
0.0          1761   30
1.0           356   29
```

Figure 10. Accuracy and Confusion for Feature of a software defect prediction system.



	loc	v(g)	ev(g)	iv(g)	n	l	d	i	e
count	10880.000000	10880.000000	10880.000000	10880.000000	10880.000000	10880.000000	10880.000000	10880.00000	1.088000e+04
mean	42.020138	6.347739	3.400037	4.001415	114.391388	0.135352	14.177339	29.43885	3.684563e+04
std	76.608641	13.021924	6.772697	9.118682	249.549291	0.160552	18.712325	34.42332	4.344671e+05
min	1.000000	1.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.00000	0.000000e+00
25%	11.000000	2.000000	1.000000	1.000000	14.000000	0.030000	3.000000	11.86000	1.619400e+02
50%	23.000000	3.000000	1.000000	2.000000	49.000000	0.080000	9.090000	21.92500	2.031020e+03
75%	46.000000	7.000000	3.000000	4.000000	119.000000	0.160000	18.902500	36.78000	1.141615e+04
max	3442.000000	470.000000	165.000000	402.000000	8441.000000	1.300000	418.200000	569.78000	3.107978e+07

Figure 11. Accuracy and Confusion for Feature of a software defect prediction system.

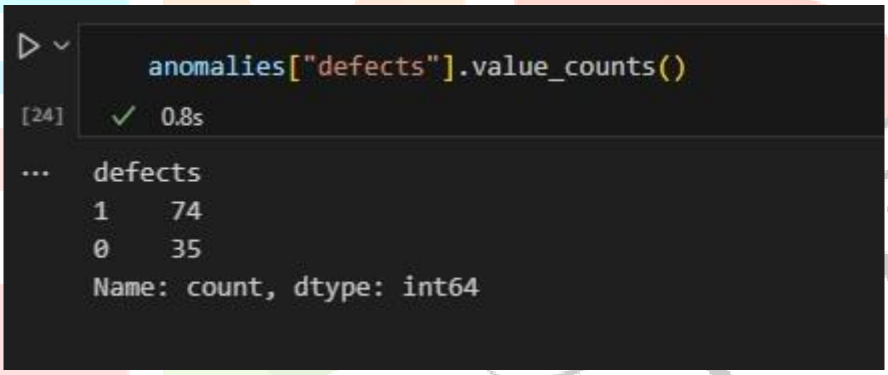


```
anomalies.head()
[23] ✓ 0.9s
```

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	...	lOCode	lOComment	lOBlank	locCod
17	250.0	49.0	34.0	16.0	1469.0	9673.31	0.01	97.00	99.72	938311.06	...	139	92	17	
251	240.0	40.0	6.0	34.0	1098.0	7648.43	0.01	90.63	84.39	693194.02	...	144	55	31	
280	469.0	2.0	1.0	2.0	1186.0	8191.57	0.02	43.85	186.83	359168.94	...	283	130	47	
283	463.0	94.0	52.0	68.0	1756.0	12727.36	0.01	71.81	177.24	913945.71	...	393	34	34	
301	415.0	9.0	1.0	8.0	2057.0	15427.18	0.02	55.06	280.16	849494.97	...	384	2	27	

5 rows × 22 columns

Figure 12. Accuracy and Confusion for Feature of a software defect prediction system.



```
anomalies["defects"].value_counts()
[24] ✓ 0.8s

defects
1    74
0    35
Name: count, dtype: int64
```

Figure 13. Accuracy and Defects for Feature of a software defect prediction system.

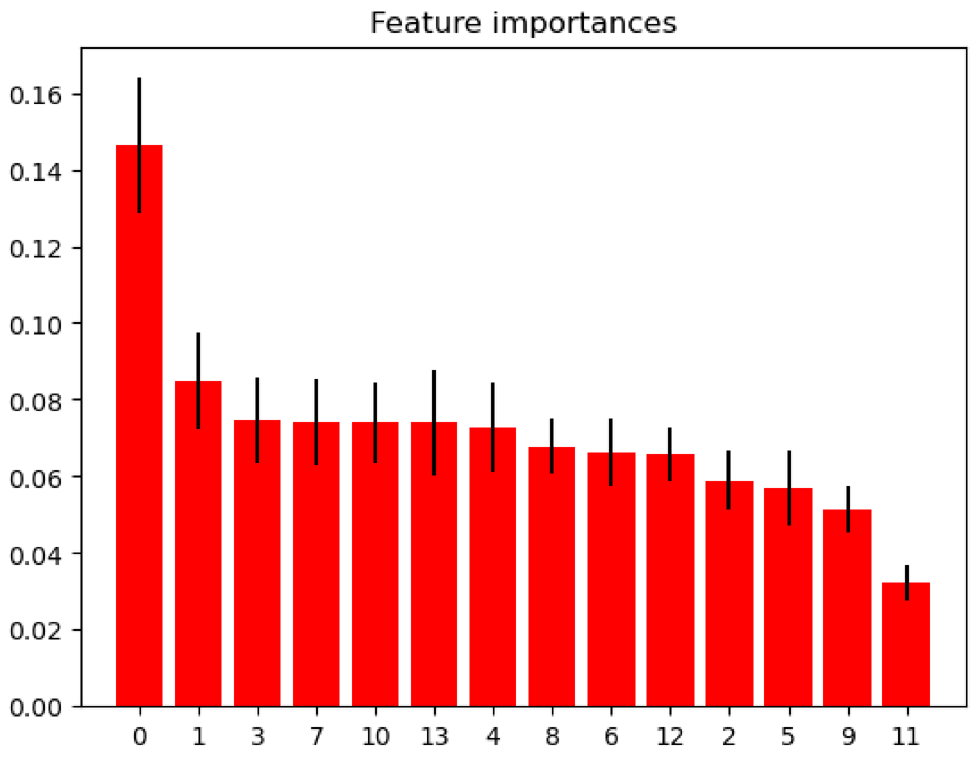


Figure 14. Feature of a software defect prediction system.

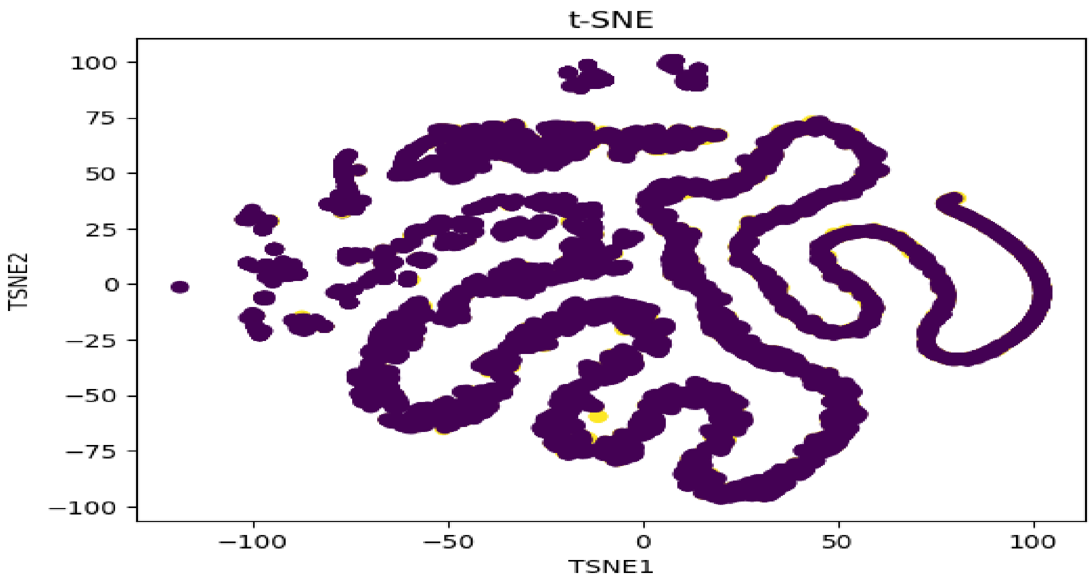


Figure 15. T-distributed neighbor embedding (t-SNE) of a software defect prediction system.

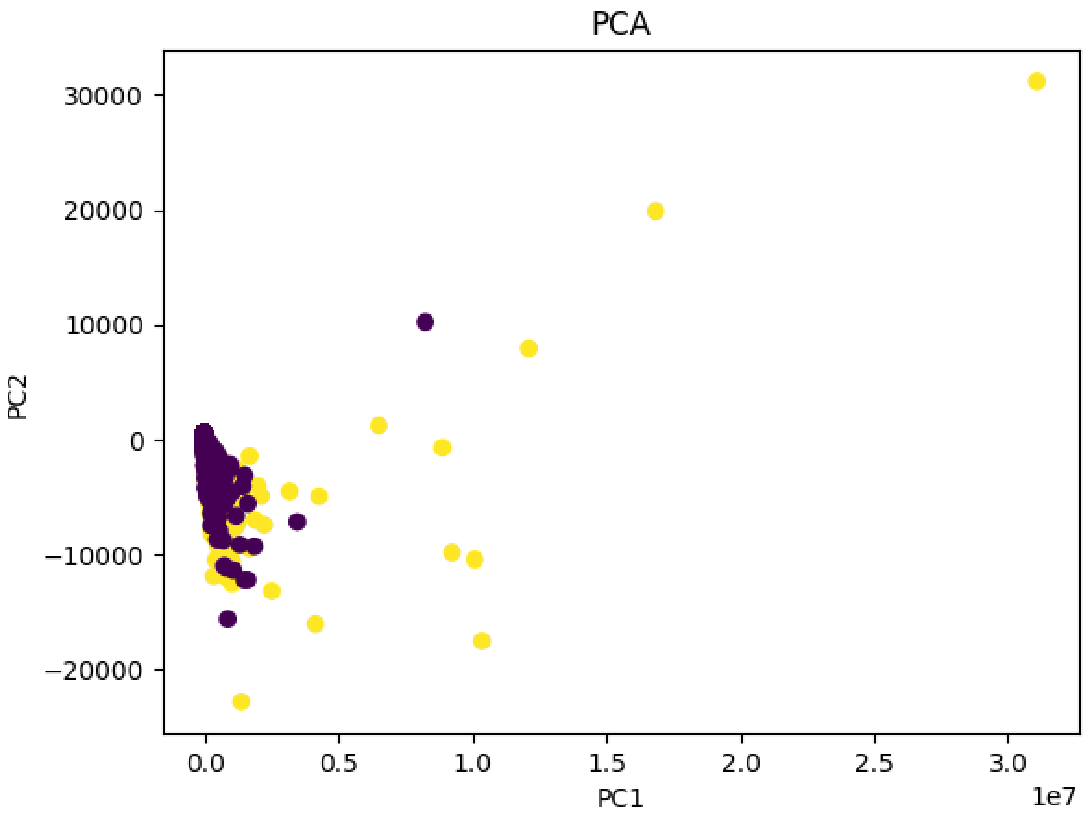


Figure 16. Principal Component Analysis test of a software defect prediction system.

VII. CONCLUSION

In summary, software bug prediction systems prove invaluable for software development teams to proactively identify and address potential bugs in their projects Use historical bug data and apply machine learning techniques to establish patterns in data across software modules or objects f capability analysis Providing improved accuracy, efficiency, and resource allocation compared to traditional methods, the system enables teams to focus on their trial and error effort where necessary, ultimately improving software quality, reducing defects Accuracy, efficiency, flexibility, scalability, user-friendliness Designed with these objectives in mind, the system leads software development teams needs Integrate UML diagrams, including usage diagrams, class diagrams to help visualize and understand the structure and structure of the system With modules related to data a collection, preprocessing, feature extraction, model training, prediction generation, and resource ho allocation , the framework provides solutions for fault forecasting and management Furthermore, visualization and reporting help stakeholders interpret fault forecasting and they are used properly. Overall, a well-designed software bug prediction system contributes significantly to software quality, resource efficiency, and the overall effectiveness of the software development process by addressing potential bugs the immediate action.

REFERENCES

1. Altuntas¸, E.; Turan, S.L. Awareness of secondary school students about renewable energy sources. *Renew. Energy* **2018**, *116*, 741–748. [CrossRef]
2. Sami, B.S. An Intelligent Power Management Investigation for Stand-alone Hybrid System Using Short-time Energy Storage. *Int. J. Power Electron. Drive Syst. (IJPEDS)* **2017**, *8*, 367. [CrossRef]
3. Waghmare, M.S.S.; Waghmare, A.P. Supervisory Control and Data Acquisition System (Scada) in Construction Industries. *J. Adv. Sch. Res. Allied Educ.* **2018**, *15*, 203–208. [CrossRef]

4. Mon, A.W.; Oo, M.Z.; Kyu, M.T. Design and Implementation of Supervisory Control and Data Acquisition Based Manufacturing System Using PID Control. *Int. J. Sci. Res. Publ. (IJSRP)* **2018**, *8*, 427–431. [CrossRef]
5. Yan, G.; Liu, J.; Huang, B. Limits of control performance for distributed networked control systems in presence of communication delays. *Int. J. Adapt. Control. Signal Process.* **2018**, *32*, 1282–1293. [CrossRef]
6. Gil Noh, S.; Choi, W.Y.; Kook, K.S. Operating-Condition-Based Voltage Control Algorithm of Distributed Energy Storage Systems in Variable Energy Resource Integrated Distribution System. *Electronics* **2020**, *9*, 211. [CrossRef]
7. Ilo, A. Are the Current Smart Grid Concepts Likely to Offer a Complete Smart Grid Solution? *Smart Grid Renew. Energy* **2017**, *8*, 252–263. [CrossRef]
8. Wang, P.; Ma, L.; Xue, K. Multitarget tracking in sensor networks via efficient information-theoretic sensor selection. *Int. J. Adv. Robot. Syst.* **2017**, *14*, 1–9. [CrossRef]
9. Jagannath, R. Detection, estimation and grid matching of multiple targets with single snapshot measurements. *Digit. Signal Process.* **2019**, *92*, 82–96. [CrossRef]
10. Wang, B.; Dehghanian, P.; Zhao, D. Chance-Constrained Energy Management System for Power Grids with High Proliferation of Renewables and Electric Vehicles. *IEEE Trans. Smart Grid* **2019**, *11*, 2324–2336. [CrossRef]
11. Lee, J.-H. Energy-Efficient Clustering Scheme in Wireless Sensor Network. *Int. J. Grid Distrib. Comput.* **2018**, *11*, 103–112. [CrossRef]
12. Stensrud, L.; Ohrn, B.; Loken, R.; Hurzuk, N.; Apostolov, A. Testing of Intelligent Electronic Device (IED) in a digital substation. *J. Eng.* **2018**, *2018*, 900–903. [CrossRef]
13. Mesaric´, P.; Ðukec, D.; Krajcar, S. Exploring the Potential of Energy Consumers in Smart Grid Using Focus Group Methodology. *Sustainability* **2017**, *9*, 1463. [CrossRef]
14. Jokar, P.; Leung, V. Intrusion Detection and Prevention for ZigBee-Based Home Area Networks in Smart Grids. *IEEE Trans. Smart Grid* **2016**, *9*, 1800–1811. [CrossRef]
15. Karaca, O.; Kamgarpour, M. Core-Selecting Mechanisms in Electricity Markets. *IEEE Trans. Smart Grid* **2019**, *11*, 2604–2614. [CrossRef]
16. Satish, M. An Integrated Cloud Based Smart Home Management System. *Int. J. Res. Appl. Sci. Eng. Technol.* **2017**, *5*, 2140–2145. [CrossRef]
17. Sebastian, J.; Hsu, Y.-L. Talking to the home: IT infrastructure for a cloud-based robotic home smart-assistant. *Gerontechnology* **2018**, *17*, 102. [CrossRef]
18. Zuo, L. Energy Harvesting Tiles Could Transform Footsteps into Power. *Sci. Trends* **2018**. [CrossRef]
19. Singh, K.; Kumar, M.N.; Mishra, S. Load Flow Study of Isolated Hybrid Microgrid for Village Electrification. *Int. J. Eng. Technol.* **2018**, *7*, 232–234. [CrossRef]
20. Datta, D.; Sheikh, R.I.; Sarkar, S.K.; Das, S.K. Robust Positive Position Feedback Controller for Voltage Control of Islanded Microgrid. *Int. J. Electr. Components Energy Convers.* **2018**, *4*, 50. [CrossRef]

21. Amri, Y.; Setiawan, M.A. Improving Smart Home Concept with the Internet of Things Concept Using RaspberryPi and NodeMCU. *IOP Conf. Series: Mater. Sci. Eng.* **2018**, *325*, 012021. [CrossRef]
22. Mahapatra, B.; Nayyar, A. Home energy management system (HEMS): Concept, architecture, infrastructure, challenges and energy management schemes. *Energy Syst.* **2019**, *13*, 643–669. [CrossRef]
23. Al Essa, M.J.M. Home energy management of thermostatically controlled loads and photovoltaic-battery systems. *Energy* **2019**, *176*, 742–752. [CrossRef]
24. Wu, J.; Yang, T.; Wu, D.; Kalsi, K.; Johansson, K.H. Distributed Optimal Dispatch of Distributed Energy Resources Over Lossy Communication Networks. *IEEE Trans. Smart Grid* **2017**, *8*, 3125–3137. [CrossRef]
25. Han, X.; Heussen, K.; Gehrke, O.; Bindner, H.W.; Kroposki, B. Taxonomy for Evaluation of Distributed Control Strategies for Distributed Energy Resources. *IEEE Trans. Smart Grid* **2018**, *9*, 5185–5195. [CrossRef]
26. Afzal, M.; Huang, Q.; Amin, W.; Umer, K.; Raza, A.; Naeem, M. Blockchain Enabled Distributed Demand Side Management in Community Energy System with Smart Homes. *IEEE Access* **2020**, *8*, 37428–37439. [CrossRef]
27. Jeya Mala, D.; Eswaran, M.; Deepika Malar, N. Intelligent vulnerability analyzer—A novel dynamic vulnerability analysis framework for mobile based online applications. *Commun. Comput. Inf. Sci.* **2018**, 805–823. [CrossRef]
28. Miraoui, M.; El-Etriby, S.; Abid, A.Z.; Tadj, C. Agent-Based Context-Aware Architecture for a Smart Living Room. *Int. J. Smart Home* **2016**, *10*, 39–54. [CrossRef]
29. Wang, Y.; Xu, Y.; Tang, Y. Distributed aggregation control of grid-interactive smart buildings for power system frequency support. *Appl. Energy* **2019**, *251*, 113371. [CrossRef]
30. Cormane, J.; Nascimento, F.A. Spectral Shape Estimation in Data Compression for Smart Grid Monitoring. *IEEE Trans. Smart Grid* **2016**, *7*, 1214–1221. [CrossRef]
31. Maitra, S. Smart Energy meter using Power Factor Meter and Instrument Transformer. *Commun. Appl. Electron.* **2016**, *4*, 31–37. [CrossRef]
32. Rocha, H.R.O.; Honorato, I.H.; Fiorotti, R.; Celeste, W.C.; Silvestre, L.J.; Silva, J.A.L. An artificial intelligence based scheduling algorithm for demand-side energy management in Smart Homes. *Appl. Energy* **2021**, *282*, 116145. [CrossRef]
33. Dai, R.; Liu, G.; Wang, Z.; Kan, B.; Yuan, C. A Novel Graph-Based Energy Management System. *IEEE Trans. Smart Grid* **2019**, *11*, 1845–1853. [CrossRef]
34. Chhaya, L.; Sharma, P.; Kumar, A.; Bhagwatikar, G. IoT-Based Implementation of Field Area Network Using Smart Grid Communication Infrastructure. *Smart Cities* **2018**, *1*, 176–189. [CrossRef]
35. Aleksic, S. A Survey on Optical Technologies for IoT, Smart Industry, and Smart Infrastructures. *J. Sens. Actuator Netw.* **2019**, *8*, 47. [CrossRef]
36. Yousif, M. Convergence of IoT, Edge and Cloud Computing for Smart Cities. *IEEE Cloud Comput.* **2018**, *5*, 4–5. [CrossRef]

37. Yaghmaee, M.H.; Leon-Garcia, A.; Moghaddassian, M.; Moghaddam, M.H.Y. On the Performance of Distributed and Cloud-Based Demand Response in Smart Grid. *IEEE Trans. Smart Grid* **2018**, *9*, 5403–5417. [CrossRef]

38. Rahmani, R.; Li, Y. A Scalable Digital Infrastructure for Sustainable Energy Grid Enabled by Distributed Ledger Technology. *J. Ubiquitous Syst. Pervasive Networks* **2020**, *12*, 17–24. [CrossRef]

39. Almehizia, A.A.; Al-Masri, H.M.K.; Ehsani, M. Integration of Renewable Energy Sources by Load Shifting and Utilizing Value Storage. *IEEE Trans. Smart Grid* **2019**, *10*, 4974–4984. [CrossRef]

40. Donaldson, D.L.; Jayaweera, D. Effective solar prosumer identification using net smart meter data. *Int. J. Electr. Power Energy Syst.* **2020**, *118*, 105823. [CrossRef]