Journal of Information Systems Engineering and Management
2025, 10(20s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Machine Learning-Based Adaptive Test Sequence Recommendation System for Regression Testing

Srinivasa Rao Kongarana1*, Ananda Rao A2, Radhika Raju P3
[1]*Research Scholar, Department of CSE, College of Engineering, JNTUA, Ananthapur, 515002, AP, India.*
Email: srinivas.cst4@gmail.com
[2]*Professor, Department of CSE, College of Engineering, JNTUA, Ananthapur, 515002, AP, India.*
Email: akepogu@gmail.com
[3]*Assistant Professor (C), Department of CSE, College of Engineering, JNTUA, Ananthapur, 515002, AP, India.*
Email: radhikaraju.p@gmail.com

ARTICLE INFO

Received: 15 Dec 2024

Revised: 29 Jan 2025

Accepted: 12 Feb 2025

ABSTRACT

In software testing, especially regression testing, selecting test sequences manually can take a lot of time and may lead to mistakes, making the process less effective. This study introduces the Adaptive Test Sequence Recommendation System (ATSRS), which uses machine learning to suggest the best test sequences. The system analyzes how different parts of the software interact and uses this information to make its recommendations. It continuously learns and improves by fine-tuning its data using linear. The ATSRS is trained with this improved data using a boosting method that combines several simple decision tree models into a stronger one. The experimental study has shown that the system is very accurate, with over 98% precision in recommending the right test sequences. This high level of accuracy makes regression testing more effective and efficient. By providing relevant and effective test sequences, the ATSRS helps improve the overall process of software testing.

Keywords: Machine Learning, Optimum Regression Testing, Adaptive Test Sequence Recommendation System, Adaptive random testing.

I. INTRODUCTION

Software testing is crucial in the software development lifecycle because it ensures the quality and reliability of software. Among the different types of testing, regression testing is especially important after software updates [1]. It helps to identify and fix bugs and regressions that could have been introduced during the updates. However, manual regression testing can be very time-consuming and prone to errors. These issues make the testing process less effective and more costly [2]. Researchers have tried to solve these problems by creating automated systems that select and recommend test cases using machine learning. These systems analyze the features of the software and suggest the best test sequences to use. Despite these efforts, there is still a significant challenge: adapting the test sequences to dynamic and constantly changing software systems. This is where the current study aims to make a difference. This study introduces the Machine Learning-Based Adaptive Test Sequence Recommendation System (ATSRS). The ATSRS is designed to recommend the best test sequences by learning from valid test sequences and the interactions in software use cases. One of the key features of the ATSRS is its ability to continuously learn and adapt using linear regression. This continuous learning allows the system to improve its recommendations over time, making it more effective in dynamic software environments.

The objective of this article is to introduce an adaptive system that can adjust to changes in software and provide accurate recommendations for regression testing. By using a machine learning model that evolves with the software, the ATSRS offers a solution that goes beyond traditional, rule-based approaches. This is achieved by integrating a continuous learning model with an adaptive boosting classifier, which combines multiple simple decision tree models to create a stronger, more accurate model. To train and fine-tune the recommendation system, the ATSRS uses data samples that have been adjusted through linear regression. This process creates enhanced data groups, which help to train the weak classifiers more precisely. The resultant recommendation model can provide precise and efficient

test sequences optimized for regression testing. The performance evaluation of the ATSRS has shown promising results, with the system achieving above 98% in precision, sensitivity, and accuracy. By utilizing the relationships and interactions between use cases and a continuous learning methodology, the ATSRS provides an enhanced and intelligent solution for test sequence recommendation.

The remaining sections of this article are structured as follows: Related Works are thoroughly reviewed in Section 2 of this article. The approach and architecture of the ATSRS are presented in Section 3, along with techniques for feature extraction, data preprocessing, and machine learning. Section 4 provides a description of the experimental design and the findings of the performance assessment. Section 5 analyzes the research's conclusions and ramifications, and ends the paper by outlining potential future research avenues.

II. RELATED WORK

The "Efficient test case generation for thread-safe classes" by Lili Bo et al. [3] presents an automatic and efficient approach for generating test cases for thread-safe classes. The objective is to find concurrency bugs by combining bug-driven and coverage-guided techniques. The approach uses static analysis to remove non-concurrent method pairs and employs bug-driven grouping and iterative generation of concurrent test cases. The evaluation on 20 thread-safe classes demonstrates significant improvement in efficiency without compromising bug finding capacities. The approach is automatic, efficient, and effective in finding concurrency bugs in thread-safe classes.

The "Search-based algorithm with scatter search strategy for automated test case generation of NLP toolkit" by Fangqing Liu et al. [4] proposes a scatter search strategy to enhance search-based algorithms for automated test case generation of NLP toolkit. The strategy enables quick coverage of paths requiring specific input variables, saving testing time and consumption. Experimental results show improved search ability and effective coverage of required paths with a small number of test cases. The strategy provides time-saving benefits and enhances search-based algorithms for NLP toolkit test case generation.

The "Adaptive Test Case Allocation, Selection and Generation Using Coverage Spectrum and Operational Profile" by Antonia Bertolino et al. [5] introduces an adaptive software testing strategy using the coverage spectrum and operational profile. The strategy, called covrel+, combines white-box coverage measures with operational profile information to select and generate effective test cases for improved reliability. Experimental results demonstrate the strategy's ability to outperform operational testing in achieving reliability targets and detecting hard-to-detect faults. The paper provides a replication package for independent verification and replication of the experiments.

Ines Hajri et al.'s article "Automating system test case classification and prioritization for use case-driven testing in product Lines" [6] suggests an automated method for classifying and ranking system test cases in the development of product lines. Automated support organizes pertinent test cases and ranks them according to risk. The industrial product family in the automotive sector demonstrates the useful advantages of the suggested method for categorizing and ranking system test cases.

The "A Proactive Approach to Test Case Selection — An Efficient Implementation of Adaptive Random Testing" by Michael Omari et al. [7] introduces adaptive random testing, a proactive method of test case selection. The algorithm chooses test cases through random and adaptive sampling during testing. The algorithm finds experimentation errors. The proposed algorithm may enhance the choice of software testing test cases, according to the proactive approach and experimental evidence.

Adaptive random testing (ART) employs two-point partitioning in "A Novel Test Case Generation Approach for Adaptive Random Testing of Object-Oriented Software Using K-Means Clustering Technique" by J Chen et al. [8]. The approach aims to improve test case selection effectiveness. Experimental results show the effectiveness of the proposed approach in detecting faults. The two-point partitioning technique differentiates it from traditional methods relying solely on random sampling. The article suggests that the proposed approach enhances test case selection in software testing.

The "Dynamic Test Case Selection using Machine Learning" by Jesper Larsson et al. [9] focuses on building machine learning models for dynamic test case selection. The models aim to reduce testing time and resource consumption while ensuring fault detection. Experimental results demonstrate the models' ability to find failures while running a fraction of test categories. The Support Vector Regression model proves to be the most efficient, finding 100% of

failures within 90% of test categories. The paper proposes an efficient solution to reduce testing efforts and ensures fault-free code deployment. However, the generalization and maintenance of the models require consideration.

Fangqing Liu et al. [10] proposes a manifold-inspired search-based algorithm called MISA for automated test case generation based on path coverage. The algorithm finds equivalent mapping subspaces using a test-case-path relationship matrix and generates test cases that cover all possible paths. Experimental results show that MISA consumes significantly fewer function evaluations compared to state-of-the-art algorithms while achieving high path coverage. The algorithm has scalability and outperforms existing methods in terms of function evaluation consumption and path coverage.

Zhan-wei Hui et al. [11] introduce a novel method named MT-ART for metamorphic testing (MT) test case generation. To create the subsequent MT test case, metamorphic relations (MRs) and adaptive random testing (ART) measure distances. In four programs, MT-ART outperforms other ART algorithms in terms of test effectiveness, efficiency, and coverage. Improved research and test case generation techniques result from taking MRs and test case effectiveness into account in MT.

For automated test case generation coverage criteria, Kling, Toon. et al. [12] forecast the best fitness functions. The study shows that branch coverage criteria and exception coverage perform statistically differently. With a f1-score of 0.865 for Mutation Score, machine learning models that use class characteristics can forecast performance. The study looks at coverage criteria and demonstrates how class traits can forecast the best fitness functions.

In order to solve problems in software engineering, Delgado-Pérez, Pedro et al. [13] suggest using the data mining-optimizer DUO approach. In order to enhance software quality and reduce production costs, the paper employs machine learning, search, and optimization algorithms. It suggests using these techniques to solve software engineering issues but doesn't go into detail about how well it works or its benefits or drawbacks.

Andries Reurink et al.'s predictive TCS&P [14] prioritizes test cases based on previous test executions. Machine learning models trained on historical execution data and software repository metrics are used to predict test failures. The resulting model achieves a recall of 95.4% and can be used for skipping tests with low fault-revealing probability, selecting high-probability tests, and prioritizing based on the chance of failure. The technique offers potential productivity gains and fault identification support, although it relies on historical data and may require significant computational resources.

T. Nithya et al. [15] propose two algorithms, simulated annealing and modified simulated annealing, for test case selection in software. The modified simulated annealing algorithm demonstrates improved performance in terms of the number of print tokens for various cost scenarios. The paper contributes efficient algorithms for test case selection, which can reduce cost and time in software testing. Empirical evaluation using the Siemens suite adds credibility to the proposed methods.

Mitchell Olsthoorn et al. [16] propose a variant of NSGA-II called L2-NSGA that uses linkage learning to optimize test case selection. The approach leverages unsupervised clustering algorithms to identify promising patterns among test suites and generates subsets that are cost-effective and detect more faults compared to existing multi-objective evolutionary algorithms. The customizations made to NSGA-II improve its effectiveness for test case selection, providing benefits in terms of cost and fault detection.

Cai et al. [17] present a paper on adversarial example-based test case generation for black-box speech recognition systems. The objective is to enhance the reliability and robustness of speech recognition systems using targeted adversarial examples. The authors propose a family of methods based on the firefly algorithm (F), augmented with gauss mutations and/or gradient estimation (F-GM, F-GE, F-GMGE) to address the specific challenges of generating targeted adversarial test cases. The experimental evaluation is performed on Google Command, Common Voice, and LibriSpeech datasets, with the performance of the adversarial examples evaluated by volunteers. The results demonstrate that these approaches outperform existing methods, effectively improving the success rate of targeted adversarial example generation. The merits of the proposed methods lie in their ability to enhance the reliability and robustness of speech recognition systems, while the absence of efficient techniques for speech recognition systems prior to this work could be considered a demerit.

The Adaptive Test Sequence Recommendation System (ATSRS) is a machine learning model created to suggest a test sequence based on the sequence of relationships and interactions in a particular use case. The ATSRS stands out as

the first of its kind in its particular objective, in contrast to the articles reviewed, which concentrate on various facets of test case generation and selection. It introduces a brand-new continuous learning model based on LR that adjusts the data samples to enhance the test sequence recommendation using adaptive boosting methods. With this method, ATSRS is able to dynamically modify and enhance its recommendations over time, increasing the testing procedure's effectiveness and efficiency, which is a key component of regression testing. ATSRS makes a significant contribution to the field of software testing by addressing the difficulty of recommending an ideal test sequence for complex use cases by utilizing adaptive boosting and continuous learning table 1.

TABLE 1: SUMMARY OF MACHINE LEARNING MODELS AND TEST CASE SEQUENCE SELECTION

Article	Test Case Sequence Selection	Machine Learning Model	Regression Testing	Processing Data Samples	Use cases	Recurrent/ Continuous Learning Model
[3]	✔	X	✔	X	X	X
[4]	✔	X	✔	X	X	X
[5]	✔	X	✔	X	✔	X
[6]	✔	X	✔	X	X	X
[7]	✔	X	✔	X	X	X
[8]	✔	X	✔	X	X	X
[9]	✔	X	✔	X	X	X
[10]	✔	X	✔	X	X	X
[11]	X	X	✔	✔	X	X
[12]	X	X	X	X	X	X
[13]	X	✔	✔	X	X	X
[14]	✔	✔	✔	✔	✔	X
[15]	✔	✔	✔	✔	X	X
[16]	✔	✔	✔	✔	✔	X
[17]	✔	✔	✔	✔	X	X
ATSRS	✔	✔	✔	✔	✔	✔

III. METHODS AND MATERIALS

The Adaptive Test Sequence Recommendation System (ATSRS) uses two main techniques to make accurate recommendations for test sequences in regression testing: adaptive boosting and continuous learning through linear regression.

Adaptive boosting, also known as Adaboost, combines several simple models, called weak classifiers, into one strong model. In this system, the weak classifiers are decision trees. The goal is to create a strong model that can make accurate predictions by learning from many simple models.

To improve the data used for training, the ATSRS uses continuous learning with linear regression. This process fine-tunes the data samples, making them better for training the Adaboost classifier. The system adjusts the data samples

into different groups using linear regression, which helps make the data more precise and useful for training. Once the data samples are fine-tuned, they are used to train the Adaboost classifier. The fine-tuned data allows the weak classifiers, which are the decision trees, to learn more effectively. This ongoing learning process ensures that the system adapts to changes and enhances its ability to recommend the best test sequences. In this way, the combination of adaptive boosting and continuous learning with linear regression improve its recommendations over time. The system becomes better at understanding the data and making accurate test sequence suggestions.

The ATSRS architecture diagram shown in figure 1 comprises four primary layers: Input, Data Preparation, Learning, and Recommendation. The Input Layer includes Use Cases and Interactions, capturing essential testing scenarios. The Data Preparation Layer involves Data Cleaning, Feature Extraction, and LR-based Data Tuning to refine and prepare the data for analysis. The Learning Layer utilizes Weak Classifiers (Decision Trees) and Adaptive Boosting to train models on the prepared data, enhancing the system's learning capabilities. Finally, the Recommendation Layer comprises the Test Sequence Recommendation module and outputs the Recommended Test Sequences. Cross-layer connections illustrate the direct influence and contributions of interactions, feature extraction, and weak classifiers on the recommendation process, with thick dashed lines emphasizing these relationships.

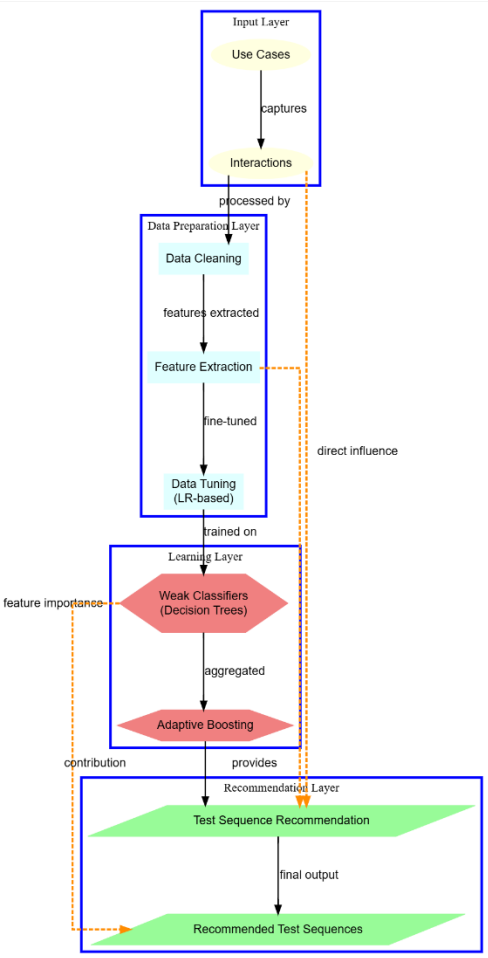

Figure 1: Architecture of proposed model ATSRS

A. The features

To recommend suitable test sequences for interactions in a use case diagram, the following features are very important:

1. **Order**: The interaction order determines the sequence of interactions in the test sequence. It makes sure that the test sequence follows the logical flow of the use case diagram.

2. **Preconditions**: These are the conditions that must be met before an interaction can take place. They include things like user authentication, system setup, or data availability.

3. **Inputs/Triggers**: These are the specific inputs or triggers needed to start actions or behaviors. This feature captures the necessary inputs or triggers for each interaction.

4. **Expected Results**: Each interaction in the test sequence produces expected outputs or outcomes. This feature captures those expected results for verification.

5. **Dependencies**: Some interactions depend on previous ones or certain conditions. This feature addresses these dependencies to make sure the interactions are executed properly.

6. **Decision Points**: Decision points are branches or choices that shape the flow of interactions. This feature ensures that different paths and scenarios are covered.

7. **Alternate Paths**: These are exceptional or alternate flows shown in the use case diagram. This feature ensures that both primary and alternate interaction paths are covered.

8. **Post-conditions**: The expected system state or behavior after each interaction is defined by post-conditions. This feature validates the expected system state after each interaction.

By using these features, the recommendation system can suggest test sequences that match the interactions in the use case diagram. This enhances the effectiveness and coverage of testing.

B. The Data

The dataset represents specific interactions within a system such that each record of the dataset represents on interaction. Each record includes several pieces of information about these interactions. The Test Sequence ID is a unique identifier for the test sequence linked to the record. The Use Case ID specifies the use case that corresponds to the record. The Interaction Order ID indicates the sequence of interactions within the use case. Preconditions are the conditions that must be met before an interaction can occur. Inputs or Triggers are the events or actions that start the interaction. Expected Outputs describe what the interaction should achieve. Dependencies show the connections between an interaction and other parts of the system. Decision Points are crucial moments where choices are necessary during the interaction. Alternate Flows consider alternative circumstances or routes. Postconditions outline the expected state of the system after the interaction.

By providing detailed information about each interaction, this dataset helps analyze and improve the system's functionality, decision-making, and overall design.

C. Data Preparation Model

The data preparation model uses a linear regression model to divide the records in the dataset into groups that are highly correlated based on the test sequences. The process begins by organizing the records according to their test sequences. Each record is assigned to a group based on its test sequence ID. The linear regression model then analyzes these groups to identify patterns and correlations within the data. This helps in fine-tuning the data for better training of the Adaboost classifier. By creating highly correlated groups, the system can improve its ability to recommend accurate test sequences for regression testing. The steps of this data preparation model are shown in figure 2.

The flow diagram shown in figure 2 for the "Data Preparation Model" illustrates the process starting with the initialization of a counter variable, followed by the consideration of records annotated with each test sequence (TS) and their assignment to set S_i. The model iterates through each test sequence, selecting optimal feature patterns for S_i, removing test sequence annotations, and training a linear regression model using these patterns. The trained model then predicts the annotated test sequences, identifying true positive records, which are added to a highly correlated set. The counter variable is incremented, and a new set S_i is created from the remaining records. This loop continues until S_i is empty, after which the process repeats for each test sequence in the set, concluding the data preparation process.

The approach iterates through each test sequence ts_k in the set TS and performing the following steps:

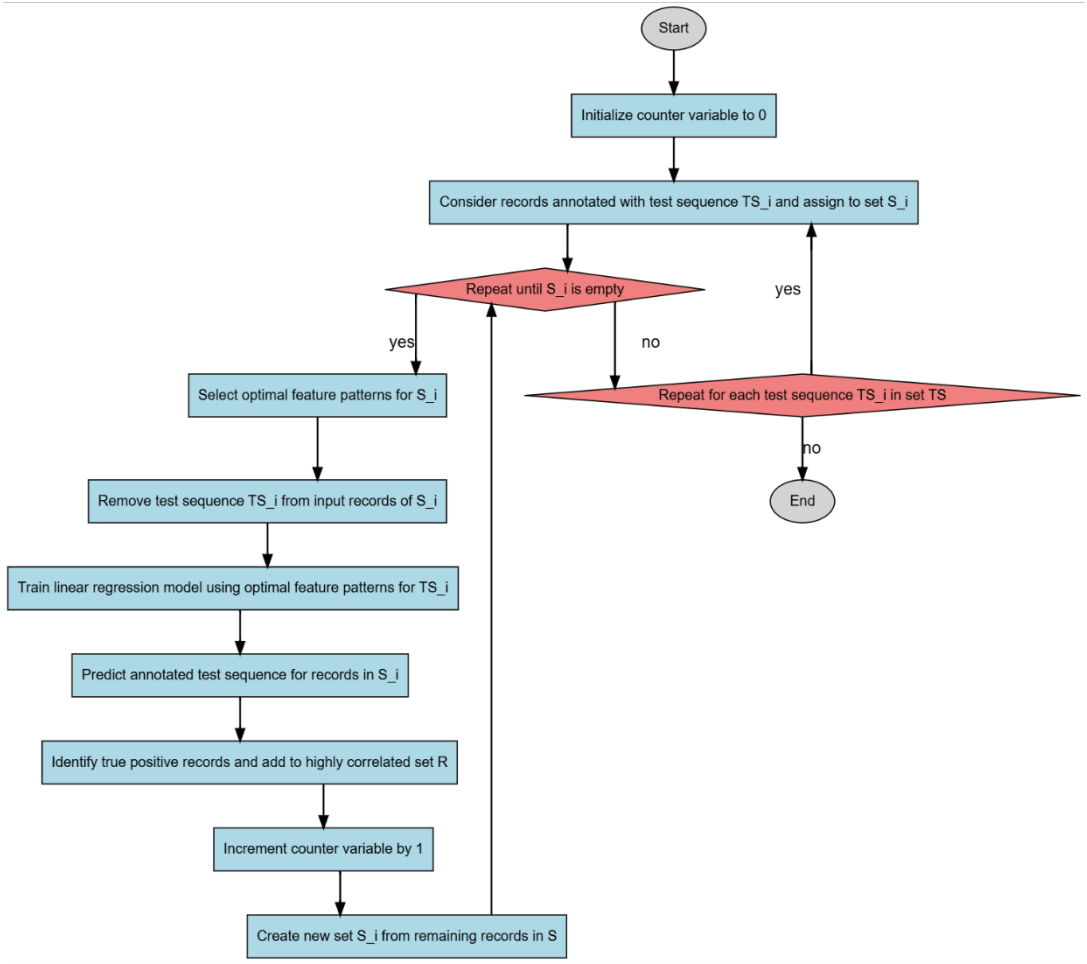

Figure 2: Flow diagram of the Data preparation model.

Given:

- Dataset D consisting of records with annotated test sequences.
- Set TS containing unique test sequences present in the dataset D.
- Optimal feature patterns selection approach to select optimal feature patterns for each test sequence.

1. For each test sequence ts_k in the set TS, perform the following steps:

a. Initialize a counter variable c to 1.

b. Create a set $D_k(c)$ to store the records annotated with test sequence ts_k. While $D_k(c)$ is not empty, do the following:

c. c. Remove test sequence ts_k from the input records of set $D_k(c)$.

d. d. Train a linear regression model using the optimal feature patterns selected for test sequence ts_k.

e. Use the trained linear regression model to predict the annotated test sequence for the records in set $D_k(c)$.

f. Identify the true positive records, which match the predicted test sequence, and add them to the highly correlated perfect records set $D_k(c)$.

g. Increment the counter c by 1.

h. Create a new set $D_k(c)$ consisting of the remaining records in $D_k(c)$ and apply the optimal feature patterns selection approach to select optimal feature patterns for this set.

2. Repeat the above steps for each test sequence ts_k in the set TS.

D. Linear Regression

The training and test sequence prediction using linear regression can be described as follows.

a) Training Phase:

1. Given a training dataset with input features X and corresponding target test sequence id Y.

2. The linear regression model aims to find the coefficients β that minimize the sum of squared errors (SSE) between the predicted test sequence id $\hat{Y}$ and the actual target test sequence id Y.

3. The linear regression equation can be represented as: $\hat{Y} = \beta_0 + \beta_1 X_1 + \beta_2 X_2 .... + \beta_n * X_n$, where $\hat{Y}$ is the predicted test-case and $X_1, X_2, ..., X_n$ are the input features.

4. The objective is to estimate the optimal coefficients $\beta_0, \beta_1, \beta_2, ..., \beta_n$ that minimize the SSE:

$$SSE = \sum \left(Y_i \hat{Y}_i \right)^2 \text{, for } i = 1 \text{ to } m \text{ where } m \text{ is the number of training samples.}$$

5. To estimate the optimal coefficients, the model uses the normal equation:

$\beta = \left(X^T X \right)^{-1} * X^T Y$, where β is the vector of coefficients, X^T is the transpose of the input feature matrix X, and $\left(X^T * X \right)^{-1}$ is the inverse of the matrix product.

b) Prediction Phase:

1. Given a new set of input features X_{test} for which the test case needs to be predicted.

2. Using the trained linear regression model with the estimated coefficients β, the predicted test case Y_{pred} can be calculated as:

$$Y_{pred} = \beta_0 + \beta_1 X_{1_{test}} + \beta_2 X_{2_{test}} + ... + \beta_n * X_{n_{test}}$$

The model utilizes the training phase to estimate the optimal coefficients of the linear regression equation, and then applies those coefficients to predict the test sequence for new input features in the prediction phase. The objective is to minimize the difference between the predicted values and the actual target values using the normal equation.

E. Optimal Feature Patterns Selection

The optimal feature patterns selection technique aims to identify the most relevant feature patterns for each test sequence in the dataset. This section presents a systematic breakdown of the steps involved in this process.

Data Preparation To prepare the data, a set TS is created to contain unique test sequences annotated to one or more records within the dataset D. Additionally, a set named P is generated to encompass all conceivable patterns of feature IDs derived from the dataset D.

Pattern Extraction Pattern extraction involves iterating through each pattern, denoted as p_i, within the set P. Within this iterative process, unique patterns of values that represent the feature IDs of p_i are extracted and stored in a set referred to as V_{p_i}.

Support Calculation Support calculation is a crucial step in the process. For every p_i in the set P, a nested loop is initiated to iterate through each pattern labeled as v_j. within the set V_{p_i}. An aggregate variable, v_j, is initialized to a starting value of 0. While iterating through these patterns v_j, the support denoted as $s\left(v_j, ts_k\right)$ for v_j concerning each test sequence ts_k found in the set TS is calculated. The aggregate variable v_j is continually updated by summing up the support values $s\left(v_j, ts_k\right)$ for each ts_k.

Weighted Support Computation The subsequent step involves the computation of weighted support for each pattern v_j concerning every test sequence ts_k. This calculation necessitates dividing the support $s(v_i, ts_k)$ by the aggregate support $as(v_j)$ to obtain the weighted support. The aggregate weighted support $aws(v_j)$ is then updated by adding up the weighted support values $ws(v_i, ts_k)$ for each ts_k.

Threshold Determination The determination of the weighted support threshold is the subsequent task. For each pattern v_j, the weighted support threshold $wst(v_j)$ is established. This is achieved by first calculating the average of the weighted supports for v_j across all test sequences present in TS. Subsequently, the standard deviation of the weighted supports is computed. Finally, the weighted support threshold $wst(v_j)$ is set as the summation of the average and the standard deviation.

Optimal Feature Patterns Selection Lastly, the technique iterates over each test sequence ts_k within the set TS. Within this iteration, an evaluation is made to ascertain whether the weighted support $ws(v_j, t_k)$ for v_j exceeds or equals the respective weighted support threshold $wst(v_j)$. If this condition is met, v_j is considered an optimal feature pattern for ts_k. Consequently, v_j is added to the set $p(ts_k)$, which represents the collection of optimal feature patterns specifically tailored to ts_k.

Let:

- D be the dataset containing records with annotated test sequences.
- TS be the set of unique test sequences annotated to one or more records in D.
- P be the set of all possible feature ID patterns.
- p_i be a pattern in the set P.
- Vp_i be the set of unique patterns of values representing the feature IDs of pattern p_i.
- As (v_j) be the aggregate support of pattern v_j.
- $s(v_j, ts_k)$ be the support of pattern v_j for test sequence ts_k.
- $ws(v_i, ts_k)$ be the weighted support of pattern v_i for test sequence ts_k.
- $aws(v_j)$ be the aggregate weighted support of pattern v_j.
- $wst(v_i)$ be the weighted support threshold of pattern v_i.
- $ofp(ts_k)$ be the set of optimal feature patterns for test sequence ts_k.

The optimal feature patterns selection algorithm can be defined as follows:

1. Initialization:
   - Set $TS = \{unique\,test\,sequence\,in\,D\}$.
   - Generate all possible patterns of feature IDs and store them in P.
2. For each pattern p_i in P:
   - Extract unique patterns of values representing the feature IDs and store them in Vp_i.
3. For each pattern v_j in Vp_i:

- Initialize as $(v_j) = 0$.

4. For each pattern v_j in Vp_i:

- For each test sequence ts_k in TS:

- Calculate the support $s(v_j, ts_k)$.

- Update as v_j by adding $s(v_j, ts_k)$.

5. For each pattern $v_j \, in Vp_i$:

- Initialize $aws(v_j) = 0$.

- For each test sequence ts_k in TS:

- Calculate the weighted support $ws(v_i, ts_k) = s(v_i, ts_k) / as(v_j)$.

- Update $aws(v_j)$ by adding $ws(v_i, ts_k)$.

6. For each pattern v_j in Vp_i:

- Calculate the weighted support threshold $wst(v_j)$ as the sum of the average of the weighted supports and the standard deviation.

7. For each test sequence ts_k in TS:

- For each pattern v_j in Vp_i:

- If $ws(v_j, ts_k) >= wst(v_i)$, then add v_j to of $p(ts_k)$.

The above steps outline the steps involved in selecting the optimal feature patterns for each test sequence based on their supports, weighted supports, and thresholds. It provides a formal representation of the algorithm and enables a systematic approach to feature pattern selection.

F. Test Sequence Recommendation System

The Test Sequence Recommendation System effectively leverages the Adaboost classification technique [18], a powerful algorithm renowned for its ability to amalgamate multiple weak classifiers into a robust ensemble classifier. Within this framework, each weak classifier takes the form of a finely-tuned decision tree model. The process commences with the creation of distinct datasets tailored to each specific test sequence. For every test sequence, multiple datasets are meticulously assembled, and a set of weak classifiers is honed through rigorous training sessions employing these datasets.

These weak classifiers, in the form of decision tree models, are expertly trained to adeptly classify the test sequences by drawing insights from the provided datasets. Once these weak classifiers are trained to perfection, the system artfully deploys them to suggest the most fitting test sequences. These recommendations are carefully tailored to the intricate interactions and relationships that characterize the use case diagrams.

The "Adaptive Test Sequence Recommendation System" unfolds in two distinct phases, each playing a pivotal role:

Initialization Phase In this initial phase, the system sets the stage for the subsequent recommendation process. It evaluates the total count of test sequences, represented as K, with each test sequence representing a distinct scenario or requirement requiring thorough testing. Additionally, the system determines the total number of datasets containing highly correlated records, aptly designated as C. These datasets are repositories of records meticulously annotated with corresponding test sequences.

Training Phase The heart of the system beats within the Training Phase, where it focuses its energies on meticulously training the weak classifiers. For each test sequence ts_k, ranging systematically from 1 to K, the system embarks on the training journey. Within this phase, the system examines each dataset, named $D_k(c)$, where c ranges from 1 to C, corresponding to their respective test sequences ts_k. The steps are methodical:

1. The system conducts a precise count, determining the total number of records within each dataset, elegantly denoted as N.
2. Each record within this dataset emerges as a noteworthy entity, represented as a feature vector bearing the distinguished label X_i.
3. Further, each of these records proudly carries a label, aptly christened as y_i, indicating its alignment or non-alignment with the test sequence ts_k.
4. The pièce de résistance within this phase involves the training of a weak classifier, gracefully christened as $h_c(k)$, leveraging the formidable AdaBoost algorithm. This process draws upon the feature vectors X_i and their corresponding labels y_i from the dataset $D_k(c)$.

Test Sequence Recommendation Phase In the Test Sequence Recommendation Phase, the spotlight shines brightly on the selection of the most fitting test sequence, a task requiring precision. The system embarks on this journey for each interaction and its symbiotic relationships.

1. The feature vector, symbolically encapsulated as X, emerges as the bridge connecting interactions and relationships, laying the foundation for the ensuing recommendation.
2. The system methodically traverses each test sequence ts_k, spanning gracefully from 1 to K. Within this realm, each weak classifier, elegantly named $h_c(k)$, is assigned a weight denoted as w_k, judiciously weighed based on its past performance during training.
3. Armed with these weighted classifiers, the system meticulously computes the weighted predictions, symbolically represented as $p_k = h_c(k)(X) \cdot w_k$.
4. The moment of truth arrives as the system singles out the test sequence ts_k boasting the highest weighted prediction value p_k, declaring it the most suitable test sequence tailored for the intricate tapestry of interactions and relationships.

The system sets the stage with meticulous parameter initialization, refines its arsenal of weak classifiers through AdaBoost training on datasets brimming with highly correlated records, and concludes its performance by elegantly recommending the most befitting test sequence, meticulously selected based on the weighted predictions of its well-trained classifiers.

1. Initialization:

- Assign equal weights (w_i) to each training example in the dataset $D_k(c)$ for each test sequence ts_k.
- Define the number of weak classifiers c, representing the total number of datasets generated for each test sequence.

2. For $t = 1$ to c:

- Train a decision tree classifier (h_t) using the current weights assigned to the training examples in $D_k(c)$.
- Calculate the weighted error (ε_t) of the decision tree classifier, which measures its performance on the training examples. $(\varepsilon_t) = \sum w_i * I(y_i \neq h_t(x_i))$ where: $-w_i$ Weight assigned to training example i. $-y_i$: True

label of training example i . $h_t(x_i)$: Prediction of the decision tree classifier for training example i . $-I(Condition)$: Indicator function that returns 1 if the condition is true, and 0 otherwise.

- Compute the weight (α_t) for the decision tree classifier, indicating its importance in the ensemble.

$$\alpha_t = 0.5 * ln\left(\frac{(1-\varepsilon_t)}{\varepsilon_t}\right)$$

- Update the weights of the training examples for the next iteration, giving higher weights to misclassified examples and lower weights to correctly classified examples.

$$w_i = w_i * exp\left(-\alpha_t * y_i * h_t(x_i)\right)$$

- Normalize the weights to ensure they sum up to one. $w_i = \frac{w_i}{Z}$ where: $-Z$: Normalization factor to ensure that the weights sum up to one.

3. Combine the weak classifiers:

- Assign a weight (β_t) to each weak classifier, which is calculated based on the weight (α_t) and the error (ε_t) .

$$\beta_t = \frac{\alpha_t}{\sum \alpha_t}$$

4. Test Sequence Recommendation:

- For a given use case with interactions and corresponding relationships:
- Construct the feature vector (x) representing the interaction and relationships.
- Compute the final weighted prediction of the ensemble classifier by summing the predictions of all weak classifiers multiplied by their respective weights. $Ensemble(x) = sign\left(\sum \beta_t * h_t(x)\right)$

Output the recommended test sequence based on the highest weighted prediction value.

IV. EXPERIMENTAL STUDY

The objective of the experimental investigation was to compare the performance of the suggested model, ATSRS (Test Sequence Recommendation System), with existing model, L2-NSGA [16]. The study made use of a dataset available in Kaggle [19] of 400 entries, each of which represented an interaction and its related use cases. These data came from 87 different use cases in total. This dataset is further augmented to best fit size of 400 records

A number of characteristics were present in each record in the dataset, including the Use Case ID, Interaction Order ID, Preconditions, Inputs or Triggers, Expected Outputs, Dependencies, Decision Points, Alternate Flows, and Post Conditions. Each record also included one of the 160 test sequence IDs, designating its connection to a particular test sequence.

Python [20] was used as the programming language for the experimental implementation. The model assessment process used a four-fold cross-validation strategy. Four subsets of the dataset were created, and the models were repeatedly trained and tested on each subset, ensuring that each subset was used for both training and testing purposes during the cross-validation procedure.

Out of the entire 400 records, 300 were utilized to train the models, while the remaining 100 records were used to test their performance. It is important to note that while all records were classified as positives, the dataset was not divided into positive and negative cases. The focus of the test sequence recommendation system, which informed this choice, is on providing pertinent test sequences to users. Metrics like true negatives and false negatives were therefore

not regarded as being important in this context. False negatives could happen if the recommendation system doesn't suggest a test sequence the user would find interesting, but real negatives might happen if the system properly foresees the user's lack of interest in a certain test sequence.

A. Performance Analysis

This section presents a thorough evaluation of the performance of two separate models: the proposed ATSRS model and the current L2-NSGA model. For this analysis, the results from 4-fold cross-validation for both models are used as a starting point. Let's get into the specifics and results of this close study.

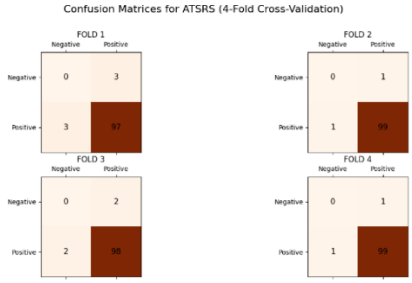

Figure 3: Confusion matrices of the 4 fold cross validation performed on proposed model ATSRS

The confusion matrices that shown in figure 3 obtained from the 4-fold cross-validation conducted on the ATSRS model offer a comprehensive assessment of its classification performance. These matrices encompass four vital components: True Positives (TP) representing accurate positive classifications, False Positives (FP) denoting incorrect positive predictions, True Negatives (TN) indicating the absence of correct negative classifications, and False Negatives (FN) representing missed positive classifications. Remarkably, the ATSRS model consistently demonstrated stability across all folds, achieving high True Positive rates while maintaining low levels of both False Positives and False Negatives. This consistency underscores the model's reliability in recommending test sequences, making it a valuable tool for precision-sensitive applications.

According to table 2, ATSRS model consistently demonstrated robust and consistent performance across various essential metrics, without exhibiting significant fluctuations or inconsistencies.

TABLE 2: INPUTS AND OUTCOMES OF THE 4 FOLD CROSS VALIDATION PERFORMED ON PROPOSED MODEL ATSRS

	ATSRS			
	FOLD 1	FOLD 2	FOLD 3	FOLD 4
input positives	100	100	100	100
input negatives	0	0	0	0
labeled as positives	100	100	100	100
labeled as negatives	3	1	2	1
true positives	97	99	98	99
false positives	3	1	2	1
true negatives	0	0	0	0
false negatives	3	1	2	1
Precision	0.97	0.99	0.98	0.99
Sensitivity	0.972	0.988	0.977	0.989
Accuracy	0.97	0.99	0.98	0.99
F-measure	0.971	0.989	0.9785	0.9895
false alarming	0.03	0.01	0.02	0.01

Precision, a metric crucial for assessing the accuracy of positive predictions, consistently presented commendable results. Precision values ranging from 0.97 to 0.99 indicate the model's remarkable ability to make highly accurate positive predictions while minimizing the chances of false positives. This precision ensures that the recommended test sequences are reliably aligned with the actual positive cases. Sensitivity, another critical measure, signifies the

model's proficiency in correctly identifying actual positive instances. The ATSRS model exhibited strong sensitivity values, consistently ranging from 0.972 to 0.989 across the four folds. These high sensitivity scores underscore the model's capacity to effectively capture true positive cases, thereby reducing the likelihood of false negatives. The overall accuracy of the ATSRS model remained consistently high, with values of 0.97, 0.99, 0.98, and 0.99 for the four folds, respectively. This demonstrates the model's exceptional correctness in its predictions, suggesting that it can be relied upon for accurate test sequence recommendations. The F-Measure, a metric that combines precision and sensitivity, consistently approached unity across all folds. This balanced performance highlights the ATSRS model's capacity to strike a harmonious equilibrium between precision and recall, further substantiating its efficacy. Furthermore, the model's low false alarming rate, which ranged from 0.01 to 0.03 across the folds, is indicative of its ability to make predictions with a minimal rate of false positives. This feature is particularly valuable as it reduces unnecessary alarms and ensures that recommendations are relevant and meaningful. The ATSRS model's consistent and robust performance across precision, sensitivity, accuracy, F-Measure, and false alarming rate underscores its reliability and suitability for recommending test sequences across diverse scenarios.

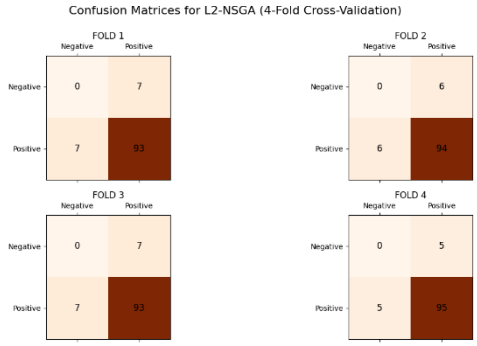

Figure 4: Confusion matrices of the 4 fold cross validation performed on existing model L2-NSGA

The confusion matrices that shown in figure 4 derived from the 4-fold cross-validation of the L2-NSGA model provide a comprehensive assessment of its classification performance. These matrices encompass essential elements, including True Positives (TP) representing accurate positive predictions, False Positives (FP) indicating incorrect positive classifications, True Negatives (TN) denoting the absence of correct negative predictions, and False Negatives (FN) representing missed positive instances. Across all four folds, the L2-NSGA model consistently demonstrated reliable performance, with strong True Positive rates and limited occurrences of both False Positives and False Negatives. This consistent and balanced performance underscores the model's capability to make precise and trustworthy recommendations for test sequences, establishing its suitability for applications that prioritize precision.

TABLE 3: INPUTS AND OUTCOMES OF THE 4 FOLD CROSS VALIDATION PERFORMED ON EXISTING MODEL L2-NSGA

L2-NSGA				
	FOLD 1	FOLD 2	FOLD 3	FOLD 4
input positives	100	100	100	100
input negatives	0	0	0	0
labeled as positives	100	100	100	100
labeled as negatives	7	6	7	5
true positives	93	94	93	95
false positives	7	6	7	5
true negatives	0	0	0	0
false negatives	7	6	7	5
precision	0.93	0.94	0.93	0.95
sensitivity	0.928	0.942	0.933	0.948
accuracy	0.93	0.94	0.93	0.95
f-measure	0.929	0.941	0.9315	0.949

false alarming	0.07	0.06	0.07	0.05

The thorough analysis of the performance of the existing L2-NSGA model, based on the results obtained from 4-fold cross-validation that shown in table 3.

Precision, a pivotal metric that assesses the accuracy of positive predictions, consistently yielded respectable outcomes across all four folds. The precision values ranged from 0.93 to 0.95, signifying the model's ability to provide accurate positive predictions while minimizing the chances of false positives. Sensitivity, another vital measure, reflects the model's effectiveness in correctly identifying actual positive instances. The L2-NSGA model consistently displayed strong sensitivity values, ranging from 0.928 to 0.948 across the four folds, which underscores the model's capacity to effectively capture true positive cases, thereby reducing the risk of false negatives. Overall accuracy, a fundamental metric, maintained consistently high levels, with values of 0.93, 0.94, 0.93, and 0.95 for the four folds, respectively. This highlights the model's exceptional correctness in its predictions, indicating that it can be relied upon for precise and accurate test sequence recommendations. The F-Measure, which combines precision and sensitivity, consistently approached unity across all folds. This balanced performance illustrates the L2-NSGA model's ability to strike a harmonious balance between precision and recall, further affirming its effectiveness. Additionally, the model demonstrated a low false alarming rate, ranging from 0.05 to 0.07 across the folds. This signifies the model's ability to make predictions with a minimal rate of false positives, reducing unnecessary alarms and ensuring that recommendations are relevant and valuable.

B. Comparative Analysis

The comparative analysis between the proposed model ATSRS and the existing model L2-NSGA across multiple critical performance metrics offers valuable insights into their respective capabilities and suitability for various applications.

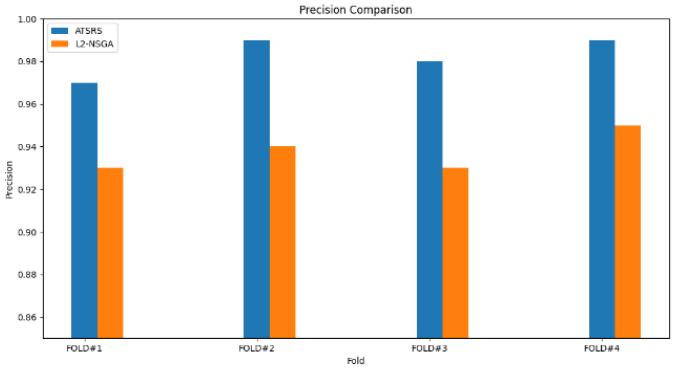

Figure 5: Comparison of Precision: ATSRS vs. L2-NSGA

Starting with precision that shown in figure 5, ATSRS consistently exhibits a high accuracy in classifying positive cases across all four folds, with individual fold precision values of 0.97, 0.99, 0.98, and 0.99. The average precision of ATSRS stands at an impressive 0.9825. On the other hand, L2-NSGA also maintains a reasonable level of precision, with an average of 0.9375 across folds. The deviation in precision for both models is relatively low, indicating their stability and reliability in correctly identifying positive cases.

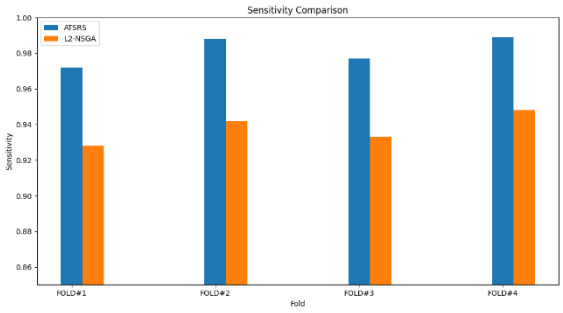

Figure 6: Comparison of Sensitivity: ATSRS vs. L2-NSGA

Sensitivity, which measures the models' ability to capture true positive instances, reflects a similar trend. According to figure 6, ATSRS consistently achieves high sensitivity values in each fold, with an average sensitivity of 0.9815. In comparison, L2-NSGA exhibits a slightly lower average sensitivity of 0.93775. The deviation in sensitivity remains moderate, emphasizing the models' consistent performance in identifying positive cases.

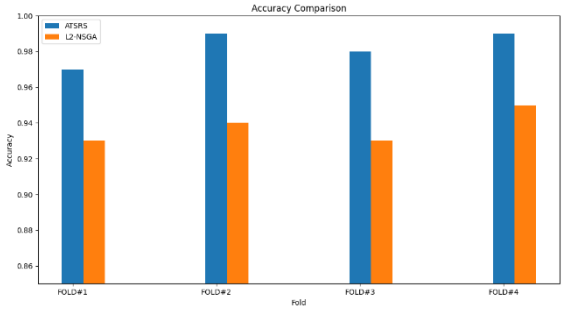

Figure 7: Comparison of Accuracy: ATSRS vs. L2-NSGA

Accuracy, a pivotal metric, demonstrates that both ATSRS and L2-NSGA maintain a commendable level of accuracy. As shown in figure 7, ATSRS records an average accuracy of 0.9825, while L2-NSGA achieves an average accuracy of 0.9375. The stable deviation in accuracy underlines the models' reliability in making accurate predictions.

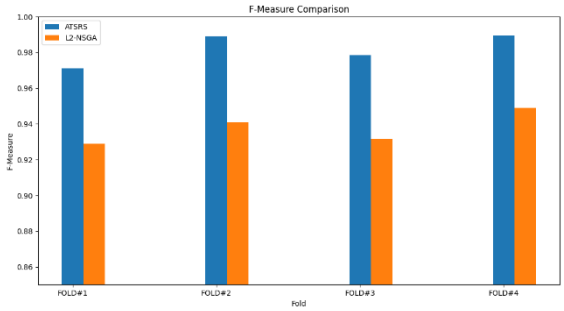

Figure 8: Comparison of F-measure: ATSRS vs. L2-NSGA

The F-measure, which combines precision and sensitivity, further emphasizes ATSRS's superiority. According to figure 8, ATSRS achieves an average F-measure of 0.982, indicating its ability to effectively balance precision and sensitivity. In contrast, L2-NSGA attains an average F-measure of 0.937625, showcasing a slightly lower performance in achieving a balance between precision and sensitivity. The deviation in the F-measure is reasonable for both models.

The comparative analysis reveals that ATSRS consistently outperforms L2-NSGA across precision, sensitivity, accuracy, and F-measure. The low to moderate deviation in these critical metrics highlights the stability and effectiveness of both models in various scenarios. Decision-makers can leverage this analysis to make informed choices based on their specific testing requirements, where ATSRS is favored for its superior performance.

V. CONCLUSION

This article introduces a new method for suggesting test sequences that combines adaptive boosting, decision tree models, and linear regression for continuous learning. Our solution promises to greatly improve the accuracy of test sequence predictions and increase the capacity for system testing, making software testing much more effective. Our study shows the potential of using ensemble methods like AdaBoost when combined with continuous learning approaches. This provides useful insights into how machine learning techniques can be efficiently applied in software testing. We demonstrated that splitting data samples into different groups can enhance the training of weak classifiers and improve the overall accuracy of the recommendations. To further improve the system's performance, future research should focus on exploring additional machine learning algorithms or other ensemble techniques. It

would also be helpful to include more advanced continuous learning models that can adjust over time based on feedback from the testing results. Investigating the possibilities of deep learning and neural networks might lead to new methods for predicting test sequences.

REFERENCES

[1] Kazmi, Rafaqut, et al. "Effective regression test case selection: A systematic literature review." ACM Computing Surveys (CSUR) 50.2 (2017): 1-32.

[2] Pan, Rongqi, et al. "Test case selection and prioritization using machine learning: a systematic literature review." Empirical Software Engineering 27.2 (2022): 29.

[3] Bo, Lili, et al. "Efficient test case generation for thread-safe classes." IEEE Access 7 (2019): 26984-26995.

[4] Liu, Fangqing, et al. "Search-based algorithm with scatter search strategy for automated test case generation of NLP toolkit." IEEE Transactions on Emerging Topics in Computational Intelligence 5.3 (2019): 491-503.

[5] Bertolino, Antonia, et al. "Adaptive test case allocation, selection and generation using coverage spectrum and operational profile." IEEE Transactions on Software Engineering 47.5 (2019): 881-898.

[6] Hajri, Ines, et al. "Automating system test case classification and prioritization for use case-driven testing in product lines." Empirical Software Engineering 25 (2020): 3711-3769.

[7] Omari, Michael, et al. "A Proactive Approach to Test Case Selection—An Efficient Implementation of Adaptive Random Testing." International Journal of Software Engineering and Knowledge Engineering 30.08 (2020): 1169-1198.

[8] Chen, Jinfu, et al. "A Novel Test Case Generation Approach for Adaptive Random Testing of Object-Oriented Software Using K-Means Clustering Technique." IEEE Transactions on Emerging Topics in Computational Intelligence 6.4 (2021): 969-981.

[9] Höstklint, Niklas, and Jesper Larsson. "Dynamic Test Case Selection using Machine Learning." (2021).

[10] Liu, Fangqing, et al. "Manifold-Inspired Search-based Algorithm for Automated Test Case Generation." IEEE Transactions on Emerging Topics in Computing 10.2 (2021): 1075-1090.

[11] Hui, Zhan-wei, et al. "MT-ART: A test case generation method based on adaptive random testing and metamorphic relation." IEEE Transactions on Reliability 70.4 (2021): 1397-1421.

[12] Kling, Toon. "Improving Automatic Test Case Generation by predicting optimal Fitness Functions." (2022).

[13] Delgado-Pérez, Pedro, et al. "Improving Search-based Test Case Generation by means of Interactive Evolutionary Computation." International Summer School on Search-and Machine Learning-based Software Engineering (SMILESENG) (2022): 39.

[14] Reurink, Andries. "Predictive Test Case Selection and Prioritization at Adyen." (2022).

[15] Nithya, T. M., and S. Chitra. "Soft computing-based semi-automated test case selection using gradient-based techniques." Soft Computing 24.17 (2020): 12981-12987.

[16] Olsthoorn, Mitchell, and Annibale Panichella. "Multi-objective test case selection through linkage learning-based crossover." Search-Based Software Engineering: 13th International Symposium, SSBSE 2021, Bari, Italy, October 11–12, 2021, Proceedings 13. Springer International Publishing, 2021.

[17] Cai, Hanbo, et al. "Adversarial example-based test case generation for black-box speech recognition systems." Software Testing, Verification and Reliability: e1848.

[18] An, Tae-Ki, and Moon-Hyun Kim. “A new diverse AdaBoost classifier.” 2010 International Conference on Artificial Intelligence and Computational Intelligence. Vol.1. IEEE, 2010

[19] https://kaggle.com/datasets/a8986d65c340ad63fef142e9ffa2ee1fbce2df7f1504b8b1a947167c1ef88606

[20] Python, Why. "Python." Python Releases Wind 24 (2021)