Tuijin Jishu/Journal of Propulsion Technology
ISSN: 1001-4055
Vol. 45 No. 2 (2024)

# Evaluation and Analysis of ML Techniques in Test Case Prioritization

**Tapas Kumar Choudhury[1], Subhendu Kumar Pani[2], Jibitesh Mishra[3]**
[1, 2] *Biju Patnaik University of Technology, Rourkela, Odisha, India*
[3] *Odisha University of Technology and Research, Bhubaneswar, Odisha, India*

***Abstract:-*** Software testing refers to checking of software whether the software is working properly and it is defect-free. Software testing is an expensive process. Nowadays as the demand for software is increasing testing of software is very essential and software bugs could be expensive for the company. Regression testing is the type of software testing which clarifies that the program code has not affected the existing program. It is partially or fully re-executing of an already run test case to confirm that the given program is free of fault or bugs. Most regression techniques are usually emphasized in the program code to prioritize test cases. The need for regression testing is when we have changed the code and we need to confirm that the modified code does not affect the other part of the code. Regression testing is large and it requires an intelligent method to categorise the important test case faster so that maximum fault can be detected at the earliest. To ensure early fault detection and addressing crucial issues in the program, we use the concept of test case prioritization. Due to the shortage of time and constraints of resources prioritization of the test case has become an important factor now. Here we use the Average Percentage of Fault Detected (APFD) Value helps to find out how quick the fault is detected in the given test suit. APFD provide a value between 0 and 1. 1 is the best possible point and the higher the APFD value, the failures are detected and executed first and it proves that the algorithm used is fast and efficient for test case prioritization. In this paper, we use 5 basic ML (Machine Learning) based algorithms in the regression-based model. We train different datasets using these algorithms and obtain results on the test case prioritization and analyse the performance of these algorithms.

***Keywords****:* Software testing, Regression testing, Test case prioritization, Machine Learning.

## 1. Introduction

Today's world is built on the strong foundations of computer software. It has become the backbone of our modern society in which every process, be it lengthy and time-consuming or very convoluted is made easy and simplistic with the help of software. But before the software is released in the real world, it must be tested thoroughly beforehand to prevent any mishaps in the future. Software testing refers to checking of software whether the software is working properly and it is defect-free. Software testing is an expensive process. Since the demand for software is increasing, testing software has become very essential and software bugs could prove to be costly and damaging for the company [4,7]. The benefits of software testing are that it is very cost-effective, provides security, improves the quality of the software product, and provides customer satisfaction.

Software testing has many phases and one of the most important phases includes testing the software using specific test cases. The test cases are used to detect faults in the software. A test case is generally defined as a set of instructions which helps in getting an error in the system by producing a failure. The aim of the test case should be to get a program which has no errors, and if any error is found in the program, it is solved quickly. But first, we need to perform some major processes to make our test case suite dependable and flawless [8]. These processes include Test case generation, test case selection, etc. which makes the test case suite veracious and easy to work with. The test suite generated afterwards may be correct but still, it may take a long time to work with. But with the increasing size and complexity of modern-day software, testing must be focused on important aspects of the application having high importance so that the most important resource, i.e. time is utilized properly.

To ensure early fault detection and addressing crucial issues in the program, we use the concept of Test Case Prioritization (TCP) [2]. TCP refers to the prioritization of the test case in the given test suit based on various factors like functionality, features, critical module, and coverages. Test case prioritization generally pivots and focuses on the important test cases which are more likely to detect errors in the system at the earliest. Due to the shortage of time and constraints of resources prioritization of the test case has become an important factor now. In today's world as the size of the software is increasing the test suit size is also increasing and due to that, the maintenance of the test suit has become very expensive. So, it is important to prioritize the test cases, so that when the need for regular or critical maintenance arises, and we must perform proper testing within a short timeframe, we can easily execute the higher priority test cases first and check the system's operability and efficiency. TCP is a great technique for increasing the efficiency of the test suite.

## 2. Related Works

For evaluation and analysis of ML algorithms in TCP we came across different research papers. The application of Machine Learning in TCP explains the most used metric here for classifying algorithms as most suitable for prioritization of test cases is the APFD [1]. The first observations show that algorithms having high APFD values are suitable for the prioritization of test cases. Secondly, we come across a limitation of ML, which is that they do not work well with small data sets. They require long data sets to produce effective results. This means more data processing with huge storage capabilities. The Machine Learning-Driven Test Case Prioritization Approaches for Black-Box Software Testing demonstrates the application of an individual set of application of the features which are used to classify unknown test cases [2]. The objective here is to find test cases which have high chances of finding faults in the software. It works on questions such as the dependency of features of test cases on different Machine Learning based algorithms and whether it is possible to train the system to achieve satisfactory goals. To identify new failures in the software, there is a requirement to use static data which already contains a static set of test cases in it. This feature was clearly implemented in the above report.

In System-Level Test Case Prioritization Using Machine Learning we get the comparisons made with unsophisticated approaches to achieve optimization in Test Case Prioritization. We came across the Black Box testing method, which is an approach to developing test cases with the knowledge of the internal structure of software [3,6,9]. Hence the techniques to vanquish the choices of test cases were included in it. It is checked whether this technique modifies the concept of prioritization as compared to basic workflow. In the Search Algorithms for Regression Test Case Prioritization which reads the at the size of the written piece of the program does not entirely affect the complexity of the Test Case Prioritization but somehow it determines the size of the search space [7]. However, it is seen that it takes longer to identify the wellness of any test case for a larger piece of the program. Hence it is the search space which is ultimately affected by the size of the program. In this report, the Additional Greedy Algorithm and the Optimal Algorithms appear to be the best approaches for Test Case Prioritization. Test case selection and prioritization using machine learning [8,17]. It contained 2 major key findings as Reinforced Learning, Supervised Learning and Unsupervised learning is mostly employed by Machine Learning based algorithms and the Machine Learning based algorithms usually depend upon several features that are easy to be found out with appropriate data values contained within it. Further, it depends upon whether these features are easy to compute with the naive approaches of Machine Learning algorithms. This paper focuses majorly on how supervised and unsupervised learning are differently affected by Machine Learning based algorithms. Next, we investigated applications of Machine Learning in software testing [14,15]. It tells us that machine learning does not help the fully automated solutions in the process of prioritization of test cases. It does show clearly how decision making is done, but not exactly how the self-acting and self-regulating solutions value the test case specifications. In this process, we mostly get impractical solutions which are not always that effective to be counterproductive in nature. So, we took the idea of how to make the fully automated specifications work freely towards the Test Case Prioritization and implemented the same in our report.

The prioritization of Fault Exposing Potential has been described by different authors [11,16]. This has a clear advantage over other methods which even states that additional executions of any statement do not add more values to the execution of statements which were performed earlier in the process of decision making of Prioritization of Test Cases. However, this even states additional Fault Exposing Potential requires several

information about the unprioritized test cases to prioritize them in the process of TCP. The development process of software is about developing in a systematic method. It takes a long time and is the most important phase of the development. It focuses more on the Regression testing phase, which is all about checking whether any changes made in the process of software development affects the previous qualities of the software which were present from the time of development in the model [8,9] . In a survey about TCP in different scenarios focuses on Heuristic approach in which the algorithms used have a prior knowledge about how far the goal state is from the current state [10,13]. Thus, this focuses more on algorithm which have certain idea about the domain. An algorithm, named MLPrior, which assigns priority to tests that are more likely to be misclassified. MLPrior effectively prioritizes test inputs by leveraging the unique characteristics of conventional machine learning classifiers, such as their interpretability and carefully crafted dataset features. Two key tenets form the foundation of MLPrior: To begin with, there is a greater likelihood of incorrect categorization for tests exhibiting increased sensitivity to mutations. Second, tests located nearer the decision boundary of the model have a higher probability of being misclassified. Utilizing these concepts, theydevelop mutation rules specifically for classical machine learning models and their associated datasets. It concurrently generates mutation to indirectly quantify the separation between a test and the decision boundary [18]. SO-SDC-Prioritizer and MO-SDC-Prioritizer are two black-box TCP techniques. These methods work well for SDCs and are dependent on a set of static road characteristics. Before testing, these characteristics can be eliminated from the driving scenarios. The recommended road features and test execution costs are used in both methods to determine the test cases' distances (diversity), which are then ranked using GAs. The SO-SDC-Prioritizer combines test diversity and execution costs into a single fitness function, thereby utilizing a single-objective optimization to achieve this goal. However, MO-SDC-Prioritizer uses a common multi-objective GA (NSGA-II) to prioritize tests based on two search objectives (test differences and test execution costs) [19].

## 3. Methods

The prioritization of test cases, machine learning and regression testing has helped to increase the efficiency or importance of the software. Machine learning is a branch which tries to learn new thing likes the way humans do by focusing mainly on the data and algorithm. It attempts to make a model which is trained repeatedly, slowly, and steadily improving itself in learning newer things and processes. Regression testing is the type of software testing which clarifies that the program code has not affected the existing program. It is partially or fully re-executing of an already run test case to confirm that the given program is free of fault or bugs [1,2,3]. The need for regression testing is when we have changed the code and we need to confirm that the modified code does not affect the other part of the code. Most regression techniques are usually emphasized in the program code to prioritize test cases. The process for TCP can be performed by various algorithms. Recently, various studies have shown the use of ML and regression-based algorithms for the process of TCP. Various concepts ranging from the fields of soft computing to artificial intelligence to neural network have been used to derive the process and algorithm for performing TCP. But, the use of so many different algorithms also raise the question of the efficiency and efficacy of the algorithms used. The efficiency of the algorithms for TCP can be measured using various parameters. But the most widely used parameter to measure the efficiency and working of the algorithm is the **APFD** value [5]**.** APFD helps to find out how quick the fault is detected in the given test suit. Faster the fault is detected quicker than algorithms. APFD provide a value between the range 0 and 1. 1 is the apex point and the higher the APFD value it shows the Test cases which provide us failure are executed first. It provides us that this algorithm is good and fast for test case prioritization. The formula (equ-1) to calculate APFD value is

$$\textbf{APFD}= 1 - \frac{\sum_{i=1}^{n} \mathrm{TFi}}{nm} + \frac{1}{2n} \text{----------} \quad (1)$$

Here, T = test suit undergoing evaluation

m = No. of faults it contains

n = Total No. of test cases

$TF_i$ = position of the first in T that expose the fault 1 and so on.

For this paper, we researched various previous studies conducted on similar topics related to TCP, regression testing and ML techniques. We found out that there are various algorithms that can be used for the process of TCP and from this set we have selected quite a few of them like k-nearest neighbour (KNN), support vector machine (SVM), Genetic Algorithm, and Hill Climbing Algorithm and Random search Algorithm to analyse and evaluate them on various parameters to check their efficiency.

## 4. Proposed Model

A test management system stores the data we need to carry out a test case prioritization. There are no explicit information types, data sets, or framework space limitations. In any event, we anticipate working with a specified set of requirements, a specified set of test cases, and a set of failures that have been made public. Each testcase in the hypothesis is related to at least one requirement, such as how a particular capability will function. Failures, which have been uncovered by a testcase, ought to be connected in like manner. Nonetheless, by and by this discernibility isn't continuously given. For our methodology, we expect to be that the creators stay up with the latest and give recognizability between related artifacts.

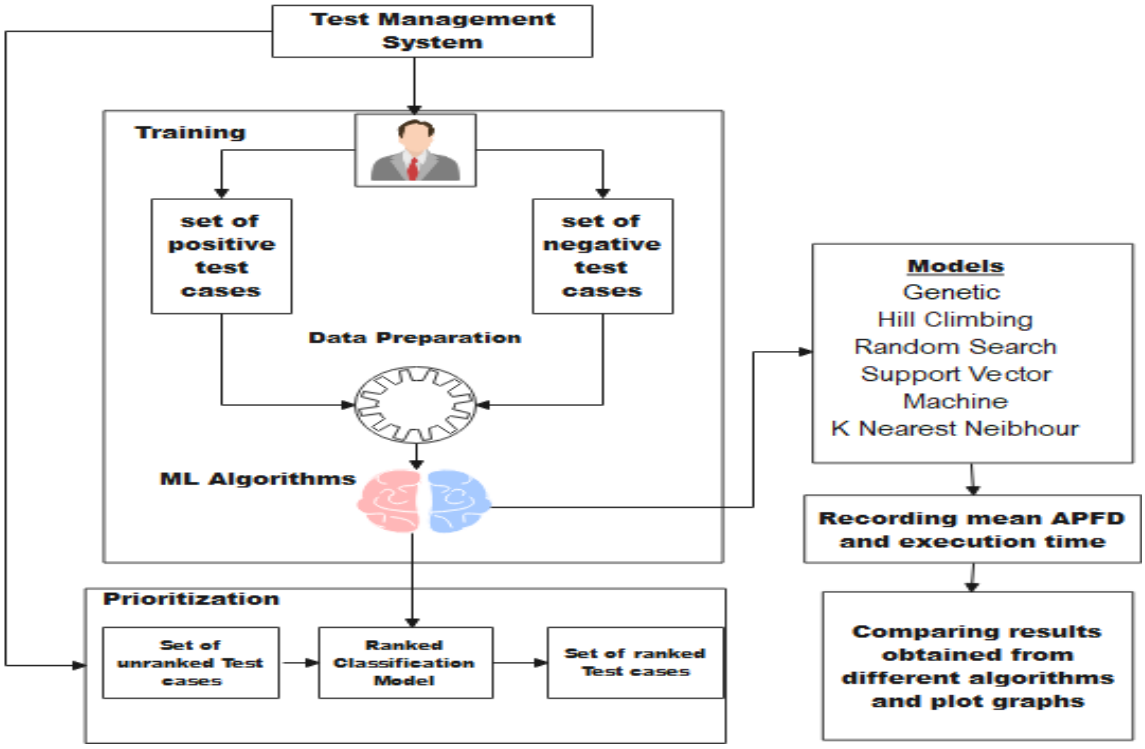


**Fig-1: Proposed Model for evaluation and analysis of ML technique in TCP**

A test master needs to choose a preparation set for the ML calculation. Specifically, we require the master to choose a bunch of positive test cases, i.e., test cases which are of high significance, utilized routinely or are for some other explanation significant for the ongoing rendition under test. To supplement this step, the master gives a set of negative testcases (Fig-1). These are off low significance, e.g., as the specific usefulness has not been changed for some variants or is of generally safe. Our methodology depends on the possibility of master information. Subsequently, we do not further confine this step, however let the analyzer make a choice about the significance of specific testcases. While this is a physically performed step, we require the master to choose a subset of test cases.

- The training data is utilized as contribution for a machine learning algorithm. Our methodology is viable to different ML algorithms, which need to satisfy the following necessities:
- Supervised, as we need to imitate the choices made by test masters based on two classes: to test and not to test.

- Ready to adapt to huge component spaces with inadequate information. Result is a ranked classification model, i.e., an info esteem is furnished with a yield esteem, addressing its priority.

After the training dataset is chosen, we remove highlights for testcases in view of their meta-information, e.g., title, number of connected prerequisites or execution span. Furthermore, we parse their printed depiction, a curiosity among regression testing strategies. After the training data is changed into a include portrayal, the ML algorithm processes a ranked classification model. A short time later, we utilize this model to focus on prioritize, unknown testcases. The outcome is an order list of testcases as indicated by their priority position. The objective is to distinguish significant testcases with a higher probability to track failures.

There are various possible algorithms that can be used for test case prioritization. Previous studies conducted on a similar topic have used some kind of ML-based algorithm with the help of regression testing. They have shown that using these algorithms have better effects on the prioritization process and give much better and optimal answers compared to other simple non-machine learning based algorithm. In this paper, we have tried to explain the widely used algorithm that are also part of previous studies being conducted. We have selected a set of 5 such ML algorithms having very diverse qualities and processing styles to implement and analyze. These are Hill Climbing, Genetic, Random Search, KNN and SVM algorithms. We have also added some other algorithms like Greedy and 2-optimal algorithm in our survey.

### 4.1 Hill Climbing Algorithm

Hill Climbing algorithm is an advanced search algorithm that moves forward to ascend to the top of a mountain or to find the best solution to a problem. If a neighbouring value is higher than the current value or status, it makes a move. Problems such as math problems use this algorithm for development purposes. With a large set of input values, it gives a solution. This is not global maxima; however, this could be the local maxima. The hill-climbing algorithm looks good only in the immediate vicinity and does not exceed much beyond the point. That is why the so-called greedy local search algorithm. This algorithm depends on the availability of a good heuristic value. If a good heuristic is found, Hill Climbing is used. The term ‘heuristic’ used here means that although a complete solution to the problem is not available but can provide a good solution in a timely manner. In the hill climb algorithm, there are two parts of the node: state and value.

There are 3 types of Hill Climbing algorithm such as

#### 4.1.1 Simple Hill Climbing*:*

It is an effective and easy way to implement the algorithm. This method only checks the value of the neighbour node status at a time and selects a node that enhances the current cost and sets it as the current state. This method only checks for the next condition and finds that the next condition is better than the current one, then it must make its move. One of the benefits of this is it consumes less time.

**Algorithm:**

1: We evaluate the state, for a goal state we return success and stop.

2: Until a solution is found, looping is done or until there is no new operator left to apply in the algorithm.

3: An operator is selected and applied to the current state from this.

4: We go for evaluating a new state and its checked:

a) Executing => goal state => return success, quit.

b) Better state => new state = current state.

c) Not better => return to step 2.

5: Exit.

**4.1.2 Steepest-Ascent Hill climbing:**

It is a variation of the simple algorithm of hill climbing. First, it scans nodes and finally selects the best node as the next node. As it searches many neighbourhoods, it is a very time-consuming process.

**Algorithm:**

1: First, condition is checked => goal condition => return success and stop.

2: Loop till the current situation is unchanged.

a) For each value of operator that satisfies present state.

b) An opr. is applied to generate a state.

c) State's evaluation is done.

d) If it is a goal state, then return it and quit, or else compare it to the S state.

e) Better state => S = new state.

3: Exit.

**4.1.3 *Stochastic* Hill Climbing:**

In this search algorithm, it selects one neighbor node at random and decides if it is to choose this as the current state or look for another state.

**Algorithm:**

1: First state = goal state => return success.

2: Repeat until no change in solution.

a) Choose a state.

b) All the neighbor states are generated by applying a successor function to the present state.

c) Choose a single state better than the present state.

d) Present state = goal state =>return success else return to step 2.b.

3: End.

**4.2 Random Search Algorithm**

AI is all about building rational agents as they act rationally. So, these agents are called rational agents and problem-solving agents. Some kinds of searches are performed by these agents to achieve their tasks. There are several states for a search problem:

i) **State space-** It is the set of all possible states where you can be.

ii) **Start state-** The search begins from this state.

iii) **Goal state-** A function looks at the current state and determines if it is the goal state or not.

Search algorithms can be classified as follows:

1. Uninformia search algorithm

These do not have any idea about the domain name, closeness to the goal, etc. It only contains ideas about how to traverse the data structure and reach the goal node. It is of several types such as: Breadth first search, Depth First Search, Uniform Cost Search, Bidirectional Search.

2. Informia search algorithm

It is something which has domain knowledge. This search is guided by problem information. It is also called a heuristic search. It is of several types such as the Best First Search and A*Search.

**4.3 Genetic Algorithm**

Genetic Algorithms are a part of evolutionary algorithms that are inspired by the evolution theory of the great naturalist Charles Darwin. It is based on natural selection and the genetic information of a population. It begins with a highly diverse population which is nothing but a set of solutions to a specific problem and simulates the process based on "survival of the fittest". The process includes selecting some solutions using fitness, crossover, and mutation steps to generate new solutions. The solutions are checked to see whether they can adapt and survive environmental changes and reproduce the next generation. Each generation contains a population of individual members which symbolises a point in the search area. It is considered as a viable solution to the problem. This point is represented in computational terms as a string of any data type like float, int or a bit. This string is analogous to chromosomes.

In genetic algorithm. Individuals compete among themselves for resources and mating to survive changes in their surroundings. The "fittest" individuals who are left mate to produce new offspring in which their fit genes are passed on. These propagated genes result in offspring which are even better than their parents. In this way, the next successive generations are more adaptable to their vicinity and environment. Thus, the GA creates highly optimized and high-quality solutions for search and optimization-based problems.

The process starts with a fitness function where a fitness score is given based upon the individual's ability to compete for the resource. Individuals having optimal fitness scores are preferred for mating and producing better offspring. Some individuals die to make room for the new fitter arrivals (having better average genes and better partial solutions) as the room for the population is static. Eventually, better solutions in successive generations replace the least fit solutions making the overall solution more and more optimized. The algorithm also consists of crossover functions where two individuals are taken with their randomly chosen crossover sites. The genes or the genetic information at such crossover sites are exchanged, thereby producing an entirely new individual having different characteristics. In mutation function, random genes are inserted into the randomly selected offspring. In this way, the diversity of the population is maintained and we are steered away from premature convergence.

The advantages of the genetic algorithm include robustness, and the ability to provide optimization over a large search space state. GA also does not break in the presence of noise and little change in input. GA has many applications in various fields including mutation testing, code-breaking, neural network, and automating designs. GA has been used in prioritizing test cases due to its working and usefulness in handling large search spaces.

**4.4 K-Nearest Neighbor**

It is based on the supervised learning model which is used to determine the category of a new data point/case. KNN stores all the available information and uses it to classify the belongingness of the new data point/ case to a well-described category by finding out the similarity of it to the respective categories. KNN does not make any assumptions beforehand and hence, it is a non-Parametric algorithm. It does not learn immediately from the training dataset; it instead stores the available data and acts at the time of classification.

First, we decide the value of 'K' randomly (preferably not a very small value such as 1 or 2, rather a very optimal number which is in the middle range) and then find out the 'Euclidean Distance' of K neighbors. Euclidean distance is the geometrical distance between any two data points We select the K nearest neighbours from the calculated data and find out the number of neighbours from each category. The new data point/case is assigned to the category having the greatest number of nearest neighbors shown in Fig-2. In this way, the model is ready to perform various classification and regression tasks.

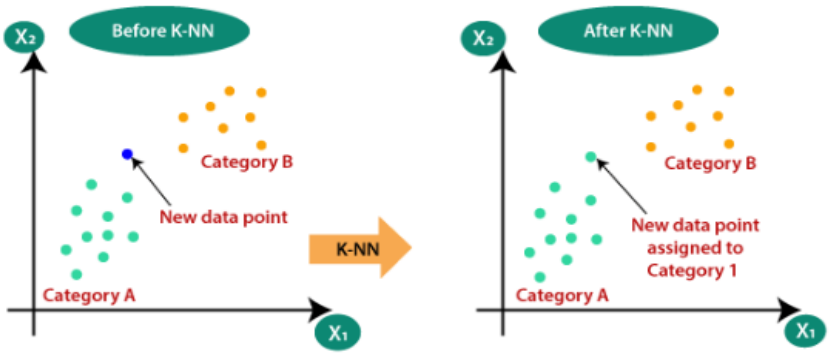


**Fig-2: K-Nearest Neibour classification**

The advantages of the KNN algorithm include robustness to noisy datasets, ease of implementation, and effectiveness for large datasets. But the computation cost is high as we must calculate the distance between two data points regularly and determine the value of 'K' which can get complex.

### 4.5 Support Vector Machines

Support Vector Machines (SVM) is a popular supervised machine learning algorithm that can be used for regression and classification purposes. The purpose of the SVM algorithm is to create a best line/decision boundary that can separate the total N-dimensional space into different classes/zones. This best line or decision boundary is described as a hyperplane. Whenever a new data point is added in the future, we can easily put it in the correct category using this hyperplane.

First, we find the extreme points/vectors which are assumed to be closest to a boundary between two separate categories. These extreme points are known as support vectors and so the algorithm is named after it as Support Vector Machine. There may exist many boundaries between the categories, but the hyperplane is selected from the set of boundaries that has the maximum margin shown in Fig-3. i.e. maximum distance between data points. SVM is of mainly two types depending upon the hyperplane: - Linear and Non-Linear SVM.

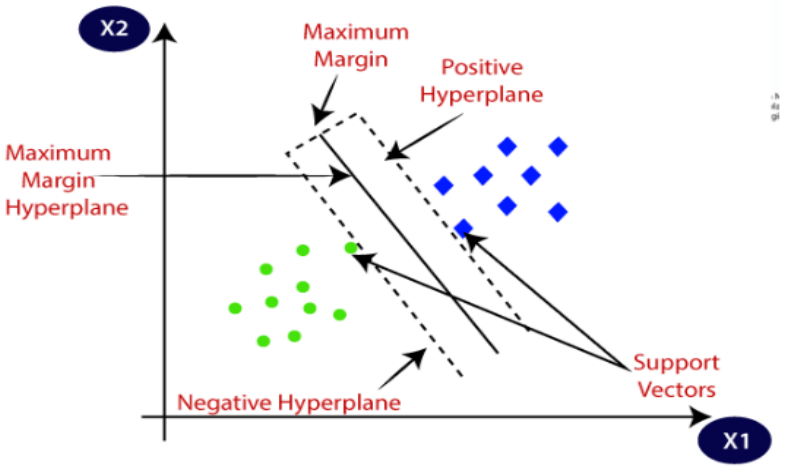


**Fig-3: SVM Classification**

The advantages of the SVM algorithm include effectiveness in high-dimensional cases and memory efficiency. SVM Rank produced good results for black-box regression testing and can compute a ranked classification model even for large feature vectors.

## 5. Results

We start our process from the input dataset which is a fault test case matrix. The dataset is already cleaned from any significant noises and is ready to be worked upon. There are two datasets we have used. The first one is a

large size fault test case and the other one is a small dataset. Both are a form of a sparse matrix where each row is a test case. There are more than 100 such rows in the big fault test case dataset and there are 50 columns as well. The matrix stores value as either 0 or 1. Here, 1 represents an error or a fault that must be detected. Similarly, the small dataset is a smaller matrix having 216 test cases represented as rows and 10 columns. The different dataset helps us in analyzing the effect on the performance of the algorithm (APFD and execution time) for TCP. The dataset is passed through different ML-based algorithms for regression testing. The major algorithms used in this work are the genetic algorithm, hill climbing, SVM, KNN algorithm and the random search algorithm. Each algorithm is run 10 times (as 10 is probably the best-fit number of repetitions to train the data in accordance with the system configuration and limits as well). We record key metrics of each algorithm specifically (like no of generations in genetic algorithm, hill climbing, etc.) at every run. Finally, the fittest APFD value and execution time (seconds) is calculated and recorded for each run as well.

At the end of 10 repeated iterative executions, we calculate the mean execution time and APFD value for each specific algorithm. This process is repeated for each of the other algorithms and at the end, we get this data in a tabular and graphical form as well (Table 1 and Table 2). The graph plotted shows the range of APFD values and its mean obtained for each algorithm which helps us in algorithm analysis and their evaluation. After implementation of the model, we get our desired results in the form of a graph and a table showing the range and mean APFD values of the algorithms implemented. In the graph, algorithms are present on the x-axis and the APFD values are mentioned on the y-axis. The range of APFD values is shaded in green, the mean APFD value is highlighted using a yellow-coloured strip and outlier values are represented by single dots on the graph (Fig-4and Fig-5).

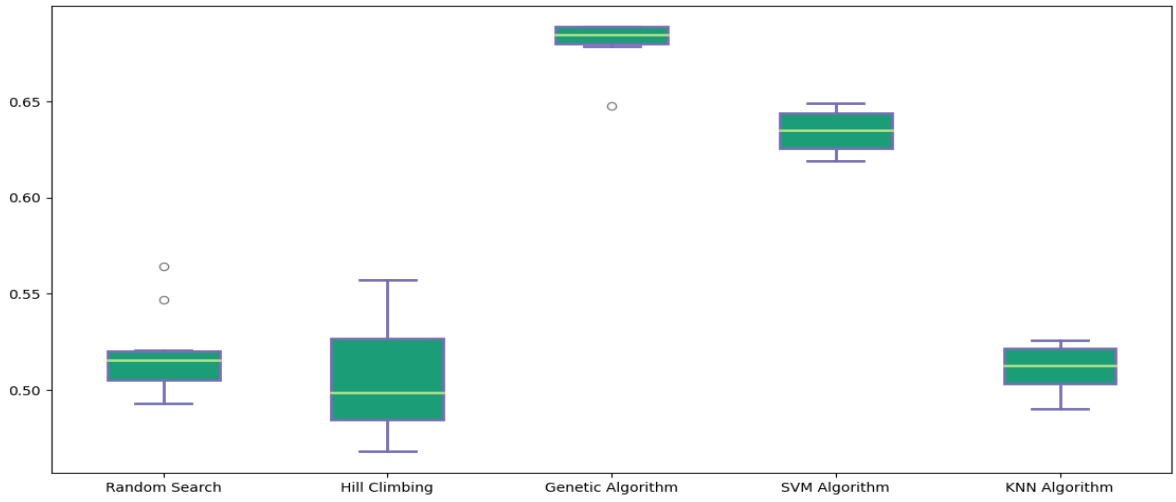


**Fig-4 :- The range of APFD values and its mean obtained for each algorithm for small dataset**

**Table1: Mean Execution Time and mean APFD values for small dataset**

| Algorithm | Mean Execution Time(sec) | Range of APFD values | Mean APFD Value |
|---|---|---|---|
| Random Search | 1.172 | 0.495 - 0.567 | 0.51851 |
| Hill climbing | 0.295 | 0.479 - 0.534 | 0.49840 |
| Genetic Algorithm | 19.991 | 0.673 - 0.692 | 0.68512 |
| SVM | 17.148 | 0.624 - 0.647 | 0.63874 |
| K-NN algorithm | 0.493 | 0.491 - 0.512 | 0.50732 |

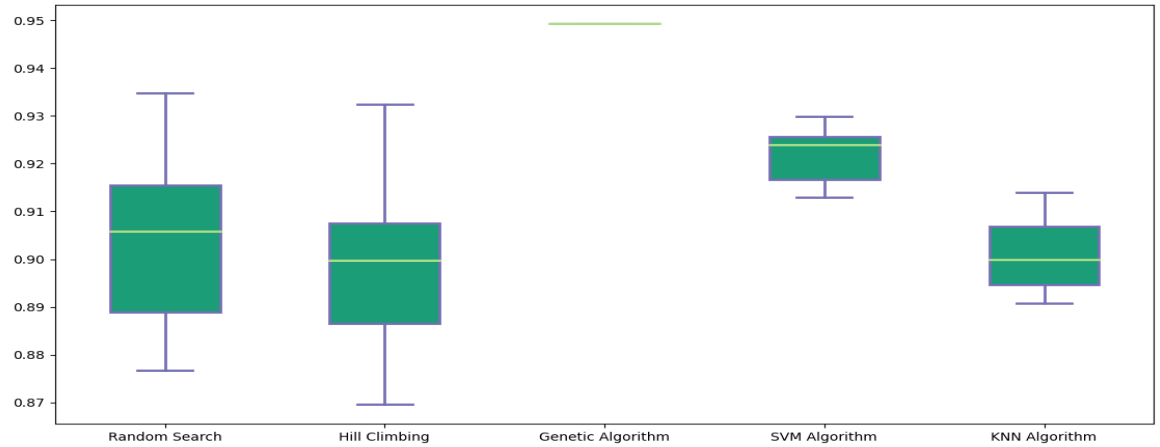


**Fig-5:- The range of APFD values and its mean obtained for each algorithm for big dataset**

**Table 2: Mean Execution Time and mean APFD values for big dataset**

| Algorithm | Mean Execution Time(sec) | Range of APFD values | Mean APFD Value |
|---|---|---|---|
| Random Search | 0.795 | 0.890 - 0.915 | 0.91354 |
| Hill climbing | 0.139 | 0.887 - 0.909 | 0.89576 |
| Genetic Algorithm | 3.692 | 0.946 - 0.950 | 0.94923 |
| SVM | 3.279 | 0.914 - 0.926 | 0.92441 |
| K-NN algorithm | 0.715 | 0.896 - 0.907 | 0.90137 |

**Key Findings:**

1. Genetic algorithm is the best algorithm for accurate early fault detection having an APFD score of 0.685 and 0.949 for small and big datasets.
2. Hill Climbing, Random Search and KNN algorithm turn out to be the least favorable algorithm for TCP as it does not have a good APFD score.
3. Genetic algorithm has a drawback. It has a high execution time of nearly 20 seconds which is significantly higher than of hill climbing and random search (under a second).

## 6. Discussion

After implementing the above-mentioned machine learning algorithms and evaluating their performance based on metrics such as APFD and execution time, we can say that the genetic algorithm has the best fittest APFD value of 0.6851 and 0.9493 in testing big and small datasets. We can conclude that the Genetic algorithm is the best possible machine learning algorithm for test case prioritization. After implementing the algorithm for different sizes of fault-test case matrix (input dataset) we can see that the fittest APFD value of every algorithm increases significantly. The genetic algorithm remains the best algorithm for TCP having a fittest APFD value of 0.9493 instead of the 0.6851 value obtained through the larger dataset. We can also say that the implementation of these algorithms on smaller datasets gives an inconsistent range of data on repeated execution.

We would like to extend the scope of our work in the future by adding more machine learning based algorithms which are widely used for test case prioritization. The probable list of algorithms which can be used are greedy algorithm, neural network-based algorithm, 2-optimal and Bayesian algorithm. Lastly, we would like to elaborate more on the importance of test case prioritization and. other test case-based process like generation and selection in real-life software testing.

**Refrences**

[1] Mece, E. K., Paci, H., & Binjaku, K. (2020). The application of machine learning in test case prioritization-a review. *European Journal of Electrical Engineering and Computer Science*, *4*(1).

[2] Lachmann, R. (2018, June). Machine learning-driven test case prioritization approaches for black-box software testing. In *The European test and telemetry conference*, *Nuremberg, Germany*.

[3] Lachmann, R., Schulze, S., Nieke, M., Seidl, C., & Schaefer, I. (2016, December). System-level test case prioritization using machine learning. In *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)* (pp. 361-368). IEEE.

[4] Dahiya, O., Solanki, K., Rishi, R., Dalal, S., Dhankhar, A., & Singh, J. (2022). Comparative Performance Evaluation of TFC-SVM Approach for Regression Test Case Prioritization. In *Proceedings of International Conference on Communication and Artificial Intelligence* (pp. 229-238). Springer, Singapore.

[5] I. H. Witten, E. Frank, and M. A. Hall. 2017. Data Mining: Practical Machine Learning Tools and Techniques (4th ed.). Morgan Kaufmann Publishers Inc. ISBN: 978-0-12-804291-5.

[6] Shi, T., Xiao, L., & Wu, K. (2020, October). Reinforcement learning based test case prioritization for enhancing the security of software. In *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)* (pp. 663-672). IEEE.

[7] Yadav, D. K., & Dutta, S. (2017). Regression test case prioritization technique using genetic algorithm. In *Advances in computational intelligence* (pp. 133-140). Springer, Singapore.

[8] Khatibsyarbini, M., Isa, M. A., Jawawi, D. N., Shafie, M. L. M., Kadir, W. M. N. W., Hamed, H. N. A., & Suffian, M. D. M. (2021). Trend Application of Machine Learning in Test Case Prioritization: A Review on Techniques. *IEEE Access*.

[9] Marijan, D. (2022). Comparative Study of Machine Learning Test Case Prioritization for Continuous Integration Testing. *arXiv preprint arXiv:2204.10899*.

[10] I. Goodfellow, Y.Bengio, A. Courville, "Deep learning".Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep Learning (vol. 1) Cambridge.

[11] Suleiman, D., Alian, M., & Hudaib, A. (2017, May). A survey on prioritization regression testing test case. In *2017 8th International Conference on Information Technology (ICIT)* (pp. 854-862). IEEE.

[12] Mohanty, S., Acharya, A. A., & Mohapatra, D. P. (2011). A survey on model-based test case prioritization. *International Journal of Computer Science and Information Technologies*, *2*(3), 1042-1047.

[13] Srivastava, P. R. (2008). Test case prioritization. *Journal of Theoretical & Applied Information Technology*, *4*(3).

[14] Briand, L. C. (2008, August). Novel applications of machine learning in software testing. In *2008 The Eighth International Conference on Quality Software* (pp. 3-10). IEEE.

[15] Li, Z., Harman, M., & Hierons, R. M. (2007). Hierons, Search algorithms for regression test case prioritization. In *IEEE Transactions on Software Engineering*. 5

[16] Joachims, T. (2002, July). Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 133-142).

[17] Rothermel, G., Untch, R. H., Chu, C., & Harrold, M. J. (2001). Prioritizing test cases for regression testing. *IEEE Transactions on software engineering*, *27*(10), 929-948.

[18] Dang, X., Li, Y., Papadakis, M., Klein, J., Bissyandé, T. F., & Le Traon, Y. (2024). Test input prioritization for Machine Learning Classifiers. *IEEE Transactions on Software Engineering*.

[19] Chen, Z., Chen, J., Wang, W., Zhou, J., Wang, M., Chen, X., ... & Wang, J. (2023). Exploring better black-box test case prioritization via log analysis. *ACM Transactions on Software Engineering and Methodology*, *32*(3), 1-32.