IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Evaluating Ensemble Bagging Techniques In Software-Defined Networks For Detecting Distributed Denial Of Service Attacks

Ashish Kumar Tiwari[1], Aseem Mishra[2], Vikas Shukla[3]

Axis Institute of Higher Education, Kanpur Axis Institute of Higher Education, Kanpur Axis Institute of Higher Education, Kanpur

Abstract: Software-Defined Networking (SDN) separates control functions from data forwarding, allowing for dynamic updates to network rules and enhancing overall network management and control. However, centralized control in SDN remains a significant challenge for researchers, as it is susceptible to various attacks, including Distributed Denial of Service (DDoS) attacks. To address this issue, this paper proposes a DDoS detection solution that leverages the K-Nearest Neighbor (KNN) machine learning algorithm to identify DDoS attacks within SDN environments. The experiment involved testing the KNN algorithm with different values of *k* to analyze its effectiveness and identify the optimal *k* value that delivers the most accurate results.

Keywords: Software Defined Networking, Distributed Denial Of Service Attacks, Machine Learning Algorithms, K-Nearest Neighbor.

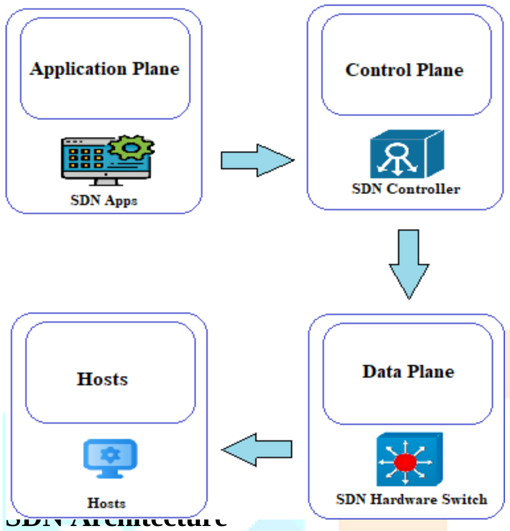
INTRODUCTION

The development of new technologies such as big data and cloud computing has led to an

increase in network traffic and dependency on networks. Particularly in Year 2020, the global COVID-19 epidemic increased the amount of work, education, and leisure that was done online, which increased the demand for network security and reliability. Traditional network architectures are no longer able to meet the needs of users and their problems have become increasingly noticeable. Because of its versatility, programming ability, dynamic nature, and user-friendliness, software-defined networking, or SDN, has become the preferred network technology [1].

By offering a more flexible and manageable solution, Software Defined Networking (SDN) targets to address the drawbacks of traditional networking. SDN allows for greater flexibility and easier management by splitting the control plane from the data plane. Benefits of SDN include centralized management using a controller, a remote device. By separating control and data tasks, network systems may be better managed and updates and alterations can be made more easily, which reduces the possibility of human error. IT managers can modify the infrastructure without any restrictions because to SDN's flexibility to use a variety of network devices from different suppliers. By providing an extensive view of the entire network, the SDN controller enhances management and supervision [2]

Fig-1 depicts the assault surface, emphasizing the locations of entry for different kinds of attacks. In Software-Defined Networking (SDN) setups, the Distributed Denial of Service (DDoS) attack is one of the hardest attach to handle. The goal of DDoS assaults is to overload controller and switch memory, which disables server resources and network bandwidth and disrupts normal user activity. DDoS attacks are very destructive and well-planned. They use a lot of computers to attack a target at the same time. This results in system failure and resource depletion, rendering the intended services unavailable to authorized users.



.

Fig-1: SDN Architecture

The unique structure of Software-Defined Networks (SDNs) that separates the control plane from the data plane makes them vulnerable to DDoS attacks. Controlling the entire network is a critical function of the centralized controller in software-defined networks (SDNs). Distributed denial-of-service (DDoS) attacks target the controller because of its centralization. If an attacker is successful in overloading the controller with requests, they can exhaust its processing power, potentially leading to serious network disruptions, packet loss, and delays in genuine traffic. The flow setup technique in Software-Defined Networks (SDNs) can also be exploited by DDoS assaults, which generate a massive distinct flow requests, overloading the flow tables of switches. The limited flow table capabilities of these switches could result in decreased performance or packet loss. An additional vulnerability can be found in the northbound and southbound APIs, which might be used to disrupt communication channel and link between the controller and network devices if improperly secured.

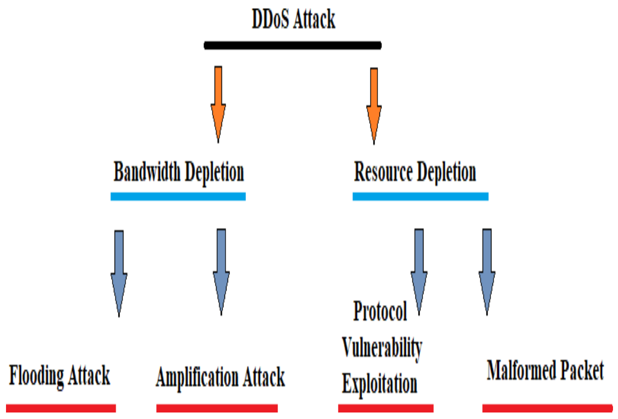


Fig-2: Classification of DDoS Attacks

Distributed Denial-of-service (DDoS) attacks on network bandwidth are designed to exhaust the available resources preventing legal traffic from reaching its destination. Packet floods, which are large volumes of traffic that disregard congestion control signals such as those in the Transmission Control Protocol (TCP) or Explicit Congestion Notification (ECN), are the primary method used to accomplish this purpose. Attacks that deplete bandwidth fall into two categories: Flood Attacks and Amplification Attacks.

Flood Attack: Often referred to as HTTP Flood Attack, is a kind of distributed denial-of-service (DDoS) assault that aims to disturb the web and application servers especially and takes place at Layer 7 of the OSI model. One kind of DDoS attack is the HTTP Flood attack, which overloads a server with valid HTTP GET or POST requests. Because these requests are legitimate and aimed at reachable resources, protecting against HTTP Flood attacks is difficult. The IP address of a client may likewise be hidden in anonymized HTTP flood attacks to avoid discovery.

Amplification Attack: An attacker, a reflector, and a target are all involved in reflection attacks. By impersonating the target's IP address, the attacker can make a request to a reflector—a middlebox or open server, for example—that responds to the target, which in this case is a virtual machine (VM). In order to intensify the attack, the response must exceed the request, which results in a reflected amplification assault. The attacker wants to use as few requests as possible to produce the largest answer. Attackers find a large number of reflectors and craft requests that produce the highest amplification in order to achieve this goal. Reflected amplification attacks are primarily caused by the ability of an attacker to manipulate reflectors to respond to targets by forging the originating IP address.

The purpose of resource depletion attacks is to exhaust vital resources on the target system, which might result in decreased performance or the inability to access services. Attackers try to exhaust specific resources, such as CPU, memory, or disk I/O, by flooding them with damaging traffic or excessive requests.

Protocol Vulnerability Exploitation: Protocol DDoS attacks disrupt services by exploiting specific flaws in network protocols. Typical instances consist of: SYN Flood: Attackers cause half-open connections that consume server resources by sending a lot of SYN queries to a target server without completing the handshake procedure.

Malformed Packet Attack: A malformed packet attack occurs when a target system receives improperly formatted IP packets, which causes the system to malfunction or operate abnormally. By defending against these kinds of assaults, a device can quickly detect and remove packets that are not appropriately constituted.

LITERATURE REVIEW

Numerous scholars have conducted research in the same field utilizing KNN, and some have shown that KNN is on top with results.

Abeer Hakeem et. Al [3]; 2024, Four primary features are used by Authors to train the Logistic Regression classifier to distinguish between attack and normal traffic: the size and quantity of packets, as well as the source and destination MAC addresses. The DDoS Detection solution's efficacy is evaluated and compared to various binary classification approach, such as Support Vector Machine, Random Forest, K-Nearest Neighbor, and Naive Bayes. The test results demonstrate that the logistic regression-based DDoS detection approach outperforms the other widely used classifiers.

G. Gnana Priya et. Al [4]; 2024, the accuracy of two machine learning methods, K-Nearest Neighbors (KNN) and Logistic Regression (LR), was assessed by the author. The test results for the two methods show differences in precision. The KNN algorithm's accuracy is almost 99%, whereas the accuracy of logistic regression is roughly 91%. They came to the conclusion that KNN performs better than logistic regression as a result of the analysis.

Usman Haruna Garba et. Al [5]; 2024, By using traditional machine learning models including Logistic Regression, SVM, Decision Trees, and K-Nearest Neighbor, the authors presented a method for quickly detecting and resolving DDoS attacks in Software-Defined Networking (SDN) smart home networks. The Decision Tree method demonstrated a 99.57% detection accuracy for attacks, demonstrating the effectiveness of machine learning algorithms in differentiating between malicious and legitimate traffic.

Dhanush Pakala et. Al [6]; 2024, Several feature selection strategies for machine learning in DDoS detection are evaluated by the author research utilizing the NSL-KDD dataset in conjunction with the Linear SVM, Quadratic SVM, and K-Nearest Neighbours (KNN) algorithms. It highlights the critical role that feature selection has in improving the SDN controller's performance and classification accuracy. The testing results show that the KNN classifier detects DDoS attacks with a 98.41% detection rate.

Abdinasir Hirsi et. Al [7]; 2024, Mininet was used to build the dataset that was used to train the model. The advantages and disadvantages of each algorithm are displayed through a comparative examination. With an F1-Score of 0.95, precision of 0.91, recall of 0.89, and accuracy of 97.31%, the SVM model performed well. 99.28% accuracy, 0.97 precision, 0.96 recall, and 0.97 F1-Score were attained by the DT model. The accuracy of the KNN model was 96.26%, while its precision, recall, and F1-Score were all 0.95. These results highlight how machine learning may be used to improve DDoS detection and strengthen software-defined network security.

ALGORITHM AND METHODOLOGY USED

We will examine the KNN algorithm, evaluating its accuracy by varying K values while taking standards and suggestions into account.

KNN Algorithm: K-Nearest Neighbor is one of the elemental and underlying supervised machine learning algorithms. The KNN algorithm classifies new instances into the category that most closely resembles the available categories based on the assumption that new data or cases are comparable to existing cases. The KNN method classifies a new data point based on its affinity to all of the previous data. This suggests that the KNN method makes it simple to classify newly available data into the proper category. Although it is mainly utilized for classification problems, the KNN technique can be practiced to both regression and classification tasks. KNN is an algorithm type that examines data without making any assumptions. Because it does not learn from the training set right away, it is recognized as a lazy learner algorithm. Rather, during the classification process, it stores the dataset and applies an action to it. Ámid the training phase, the KNN algorithm only retains the dataset. It assigns new data to a category that closely matches the new data when it receives it [8].

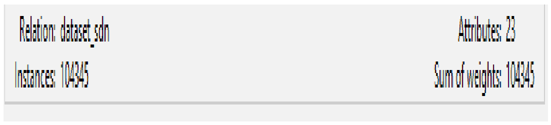
In the KNN algorithm, the k value denotes the number of neighboring points that will be taken into account while classifying a specific query point. If k equals 1, for example, the instance will belong to the same class as its lone neighbor. Regulating k can be challenging since varying values may cause overfitting or under- fitting. Although lower values of k will typically be close to the real values (low bias), they can produce a broad range of results (high variance). Conversely, results with greater values of k can be more consistent (lower variance), but they might also deviate more from the true values (high bias). The choice of k will mostly depend on the input data because higher values of k are likely to produce better results for data with more noise or outliers. usually, it is advised to choose an odd value for k in order to avoid classification ties, and cross-validation techniques can help you choose the ideal value for your dataset [9].

The following considerations should be made while determining the value of K in the K-NN algorithm:

There is no set way to identify the ideal value for "K," therefore we have to experiment with different values to discover the optimum one. According to domain experts, the most popular value for K is 5. Outlier effects can occur in the model when K is set to an extremely small value, such as K=1 or K=2. Greater K values are advantageous, but they can also cause problems.

To determine the initial K value of the KNN method, experts also recommend the Square Root of N rule, where N is the total count of data points in the dataset [10].

This study uses the DDoS SDN dataset, which is a secondary dataset with three categories, 104,345 rows, and 23 columns. The "label" is used as the dataset's target variable. In this case, "1" denotes benign traffic while "0" denotes malicious traffic. Fig-3 displays the dataset information from the WEKA tool, which was used for the experiment.



```
Relation: dataset_sdn                    Attributes: 23
Instances: 104345                        Sum of weights: 104345
```

Fig-3: Dataset Information RESULT AND

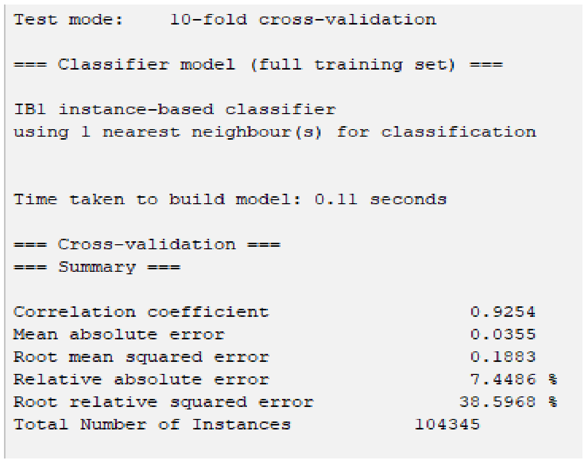
CONCLUSION

Testing was conducted using a range of K values. As recommended by many domain experts, we choose to use odd values of K and discovered that the results were mostly unchanged. Although accuracy was above 90% in the majority of cases, our highest accuracy was 94.58% with a K value of 5. We also investigated the expert-recommended square root of N formula (K=323) , which we were able to accomplish with 88.29% accuracy. Table-1 shows the test result.

Table-1: Test Result

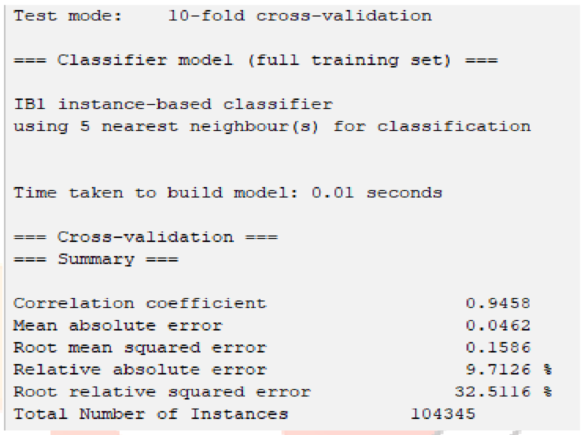
Value of k (Odd)	Accuracy	Mean Absolute Error
1	92.54%	3.55%
5	94.58%	4.62%
9	94.56%	5.19%
13	94.26%	5.58%
17	94.02%	5.89%
21	93.82%	6.12%
25	93.67%	6.31%
29	93.53%	6.48%
33	93.37%	6.63%
37	93.21%	6.77%
323	88.29%	11.49%

Fig-4 depicts the testing with different K values on WEKA tool.



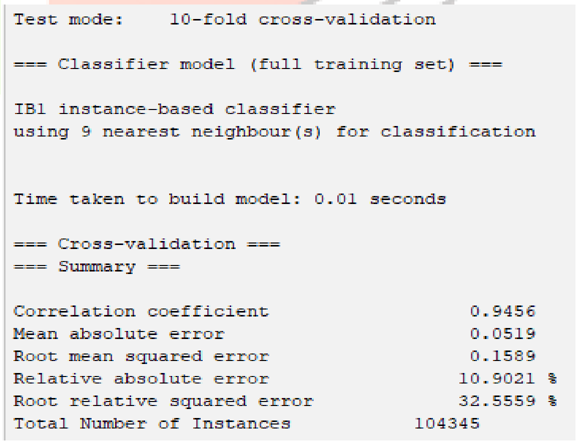
```
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

IB1 instance-based classifier
using 1 nearest neighbour(s) for classification


Time taken to build model: 0.11 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.9254
Mean absolute error                      0.0355
Root mean squared error                  0.1883
Relative absolute error                  7.4486 %
Root relative squared error             38.5968 %
Total Number of Instances             104345
```

a) K=1



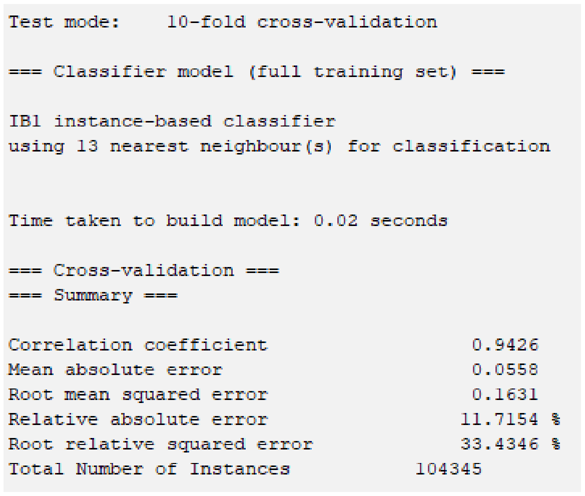
```
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

IB1 instance-based classifier
using 5 nearest neighbour(s) for classification


Time taken to build model: 0.01 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.9458
Mean absolute error                      0.0462
Root mean squared error                  0.1586
Relative absolute error                  9.7126 %
Root relative squared error             32.5116 %
Total Number of Instances             104345
```

b) K=5



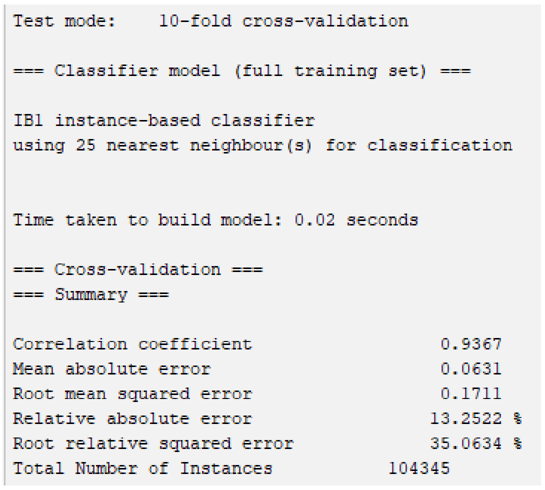
```
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

IB1 instance-based classifier
using 9 nearest neighbour(s) for classification


Time taken to build model: 0.01 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.9456
Mean absolute error                      0.0519
Root mean squared error                  0.1589
Relative absolute error                 10.9021 %
Root relative squared error             32.5559 %
Total Number of Instances             104345
```

c) K=9



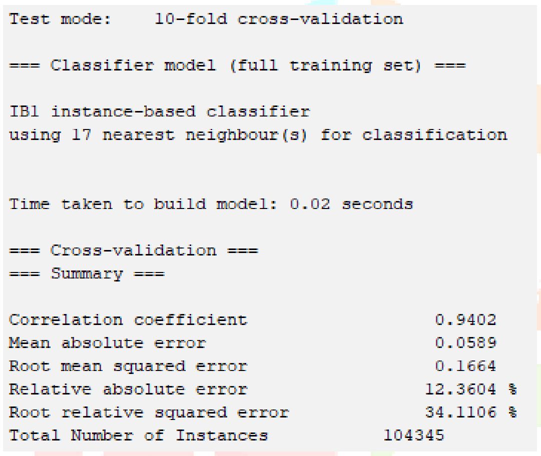
```
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

IB1 instance-based classifier
using 13 nearest neighbour(s) for classification


Time taken to build model: 0.02 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.9426
Mean absolute error                      0.0558
Root mean squared error                  0.1631
Relative absolute error                 11.7154 %
Root relative squared error             33.4346 %
Total Number of Instances           104345
```

d) K=13



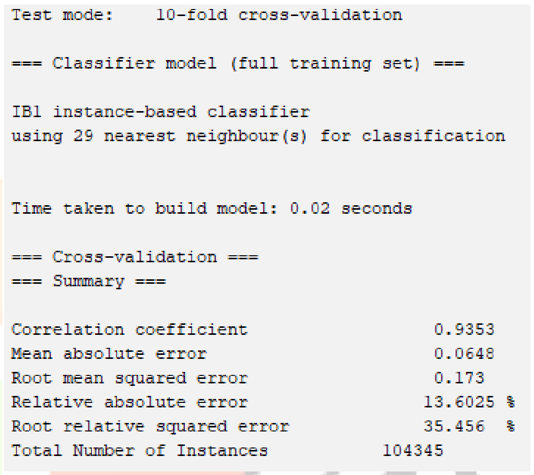
```
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

IB1 instance-based classifier
using 25 nearest neighbour(s) for classification


Time taken to build model: 0.02 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.9367
Mean absolute error                      0.0631
Root mean squared error                  0.1711
Relative absolute error                 13.2522 %
Root relative squared error             35.0634 %
Total Number of Instances           104345
```

g) K=25



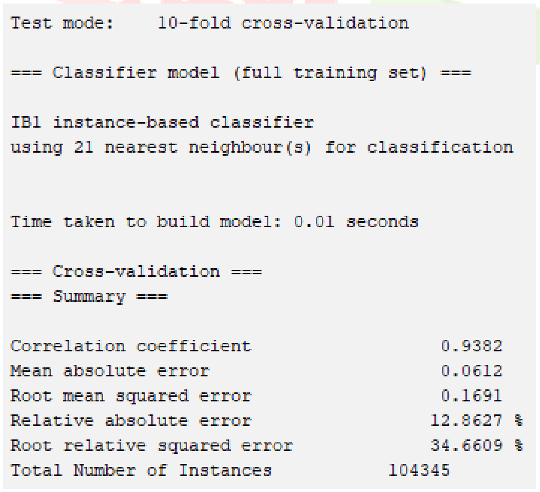
```
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

IB1 instance-based classifier
using 17 nearest neighbour(s) for classification


Time taken to build model: 0.02 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.9402
Mean absolute error                      0.0589
Root mean squared error                  0.1664
Relative absolute error                 12.3604 %
Root relative squared error             34.1106 %
Total Number of Instances           104345
```

e) K=17



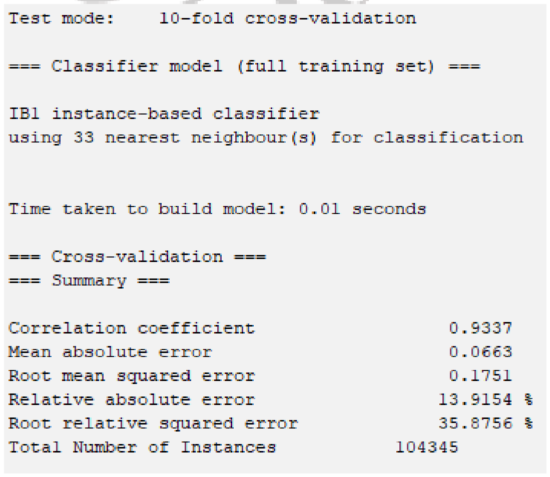
```
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

IB1 instance-based classifier
using 29 nearest neighbour(s) for classification


Time taken to build model: 0.02 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.9353
Mean absolute error                      0.0648
Root mean squared error                  0.173
Relative absolute error                 13.6025 %
Root relative squared error             35.456  %
Total Number of Instances           104345
```

h) K=29



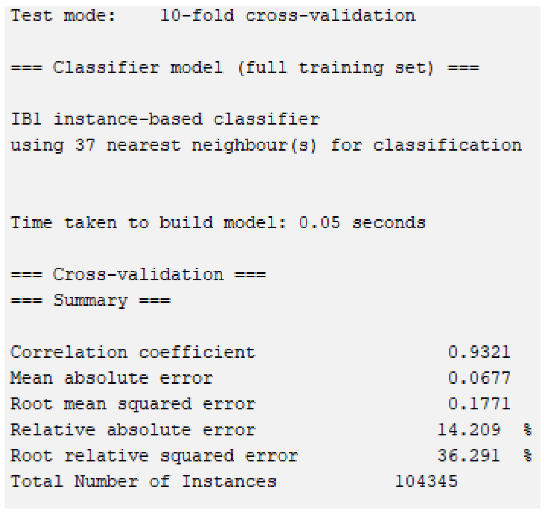
```
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

IB1 instance-based classifier
using 21 nearest neighbour(s) for classification


Time taken to build model: 0.01 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.9382
Mean absolute error                      0.0612
Root mean squared error                  0.1691
Relative absolute error                 12.8627 %
Root relative squared error             34.6609 %
Total Number of Instances           104345
```

f) K=21



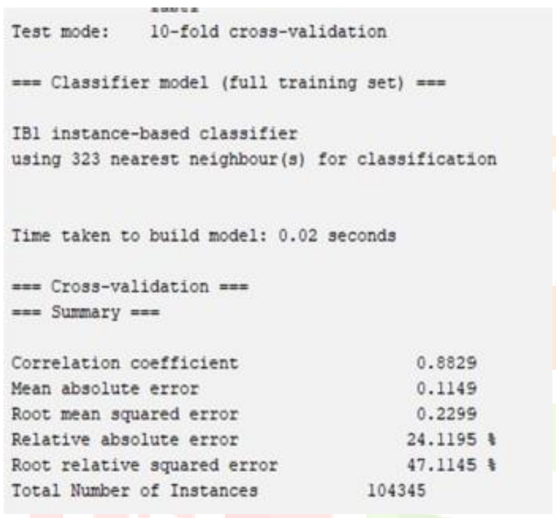
```
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

IB1 instance-based classifier
using 33 nearest neighbour(s) for classification


Time taken to build model: 0.01 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.9337
Mean absolute error                      0.0663
Root mean squared error                  0.1751
Relative absolute error                 13.9154 %
Root relative squared error             35.8756 %
Total Number of Instances           104345
```

i) K=33



```
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

IB1 instance-based classifier
using 37 nearest neighbour(s) for classification


Time taken to build model: 0.05 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.9321
Mean absolute error                      0.0677
Root mean squared error                  0.1771
Relative absolute error                 14.209  %
Root relative squared error             36.291  %
Total Number of Instances           104345
```

j) K=37



```
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

IB1 instance-based classifier
using 323 nearest neighbour(s) for classification


Time taken to build model: 0.02 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.8829
Mean absolute error                      0.1149
Root mean squared error                  0.2299
Relative absolute error                 24.1195 %
Root relative squared error             47.1145 %
Total Number of Instances           104345
```

k) K=323 (Square root on N, Where N=104345)

Fig-4: Testing With Different K Values

Crucial findings from this experiment are as:

- The Mean Absolute Error (MAE) also rises as K values increase, suggesting that the results have greater error as K increases.

- Contrary to what we discovered in many studies and articles, results essentially remain the same or accuracy reduces as the value of K rises.

- We realized that there is no fixed method to get the ideal or optimal value ok k.

The comparison analysis based on the test results is displayed in Fig. 5.

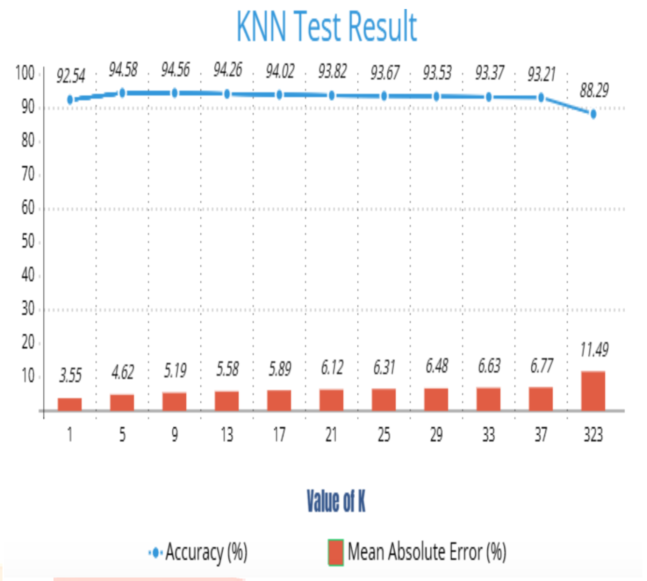


Fig-5: Testing Comparative Analysis