AN EFFICIENT ANDROID MALWARE DETECTION USING GA, SVM AND ANN.

[1]Medikonda Naveen Babu, [2]Abhijeet Prakash Yadav, [3]Pasupuleti Krishna Tejasvi, [4]Rapole Manish, [5] Dr. K. Srinivas

UG Scholar[1, 2, 3,4], Associate Professor[5]
Computer Science and Engineering, St. Martins Engineering College, Secunderabad [1,2,3,4]

naveenbabumedikonda4@gmail.com[1], abhijeetprakash25@gmail.com[2], pasupuleti.tejasvi@gmail.com[3], rmanish0308@gmail.com[4], drksrinivascse@smec.ac.in[5]

***Abstract*— Android platform due to open-source characteristic and Google backing has the largest global market share. Being the world's most popular operating system, it has drawn the attention of cyber criminals operating particularly through wide distribution of malicious applications. This project proposes an effectual machine-learning based approach for Android Malware Detection making use of evolutionary Genetic algorithm for discriminatory feature selection. Selected features from Genetic algorithm are used to train machine learning classifiers and their capability in identification of Malware before and after feature selection is compared.**

Keywords—CSV file, Term Frequency, Feature Selection, Machine Learning, SVM, Genetic Algorithm, ANN

I. INTRODUCTION

Android Apps are freely available on Google Play store, the official Android app store as well as third-party app stores for users to download. It is an open-source nature and popularity; malware writers are increasingly focusing on developing malicious applications for Android operating system.Therefore, there is need to perform malware analysis or reverse-engineering of such malicious applications which pose serious threat to Android platforms. Broadly speaking, Android Malware analysis is of two types: Static Analysis and Dynamic Analysis. Static analysis basically involves analyzing the code structure without executing it while dynamic analysis is examination of the runtime behavior of AndroidApps in constrained environment.

Given in to the ever-increasing variants of Android Malware posing zero-day threats, an efficient mechanism for detection of Android malwares is required. In contrast to signature-based approach which requires regular update of signature database, machine-learning based approach in combination with static and dynamic analysis can be used to detect new variants of Android Malware posing zero-day threats. In, broad yet lightweight static analysis has been performed achieving a decent detection accuracy of 94% using Support Vector Machine algorithm.

Nikola Milosevic et al. presented static analysis-based classification through two methodologies: one was permissions based while the other involved representation of the source code as a bag of words. Another approach based on identifying most significant permissions and applying machine learning on it for evaluation has been proposed. MADAM provides a multi-level analysis framework where behaviour of Android Apps is captured up to four levels: from package, user, application to kernel level, achieving detection accuracy as high as 96% with one disadvantage

The main contribution of the work is reduction of feature dimension to less than half of original feature-set using Genetic Algorithm such that it can be fed as input to machine learning classifiers for training with reduced complexity while maintaining their accuracy in malware classification. In contrast to exhaustive method of feature selection which requires testing for 2N different combinations, where N is the number of features, Genetic Algorithm, a heuristic searching approach based on fitness functionhas been used for feature selection.

The optimized feature set obtained using Genetic algorithm is used to train two machine learning algorithms: Support Vector Machine and Neural Network. It is observed that a decent classification accuracy of more than 94% is maintained while working on a much lower feature dimension, thereby, reducing the training time complexity of classifiers. The remaining paper is structured as follows: Section II discusses about the proposed methodology used. Section III presents experimentation results obtained by applying genetic algorithm for feature selection to train machine learning algorithms. Section IV briefs about the general conclusions drawn from the experimentations

RELATED WORK

For the last few years, Android is known to be the most widely used operating system and this rapidly increasing popularity has attracted the malware developer's attention. Android allows downloading and installation of apps from other unofficial marketplaces. This gives malware developers an opportunity to put repackaged malicious applications in third-party app stores and attack Android devices. Many malware analysis and detection systems have been developed which uses static analysis, dynamic analysis, or hybrid analysis to keep Android devices secure from malware [1].

Android users are constantly threatened by an increasing number of malicious applications (apps), generically called malware. Malware constitutes a serious threat to user privacy, money, device, and file integrity. In this paper, we note that, by studying their actions, we can classify malware into a small number of behavioral classes, each of which performs a limited set of misbehaviors that characterize them. These misbehaviors can be defined by monitoring features belonging to different Android levels [2].

With the widespread use of smartphones, the number of malware has been increasing exponentially. Among smart devices, android devices are the most targeted devices by malware because of their high popularity. This paper proposes a novel framework for android malware detection. Our framework uses various kinds of features to reflect the properties of android applications from various aspects, and the features are refined using our existence-based or similarity-based feature extraction method for effective feature representation on malware detection [3].

The alarming growth rate of malicious apps has become a serious issue that sets back the prosperous mobile ecosystem. A recent report indicates that a new malicious app for Android is introduced every 10 s. To combat this serious malware campaign, we need a scalable malware detection approach that can effectively and efficiently identify malware apps. Numerous malware detection tools have been developed, including system-level and network-level approaches. However, scaling the detection for a large bundle of apps remains a challenging task [4].

Their ability to build models adapted to complex non-linear problems has made them a technique widely applied and studied. One of the fields where this technique is currently being applied is in the malware classification problem. The malware classification problem has an increasing complexity, due to the growing number of features needed to represent the behavior of the application as exhaustively as possible [5].

A novel approach is proposed to detect malware in a hybrid and generic way, especially for mobile malware in Android devices. By comparing the patterns reflected by the above individual and sequential of malware and benign apps with each other, we build up a malicious pattern set and a normal pattern set that are used for malware detection and benign app judgment. When we need to detect an unknown app, we use a dynamic method to collect its runtime system calling data in terms of both individual calls and sequential system calls with different depths. Then we extract the target patterns of the unknown app from its runtime system calling data [6].

The widespread adoption of Android devices and their capability to access significant private and confidential information have resulted in these devices being targeted by malware developers. Existing Android malware analysis techniques can be broadly categorized into the static and dynamic analysis. In this paper, we present two machine learning-aided approaches for static analysis of Android malware [7].

The growing amount and diversity of Android malware have significantly weakened the effectiveness of the conventional defense mechanisms, and thus Android platform often remains unprotected from new and unknown malware. To address these limitations, we propose DroidDeep, a malware detection approach for the Android platform based on the deep learning model. Deep learning emerges as a new area of machine learning research that has attracted increasing attention in artificial intelligence [8].

The contributions include: description of the development of a deep-learning- based Android malware detection engine (DroidDetector)that has been put online for user testing and can automatically detect whether an app is a malware or not. In this software, 20 000 apps from the Google Play Store and collect 1760 malware from the well-known Contagio Community and Genome Project [9].

An integrated anti-malware analysis system that considers both malware- centric information and malware creator-centric information for malware analysis. Using malware creator-centric information simplifies the process of malware analysis. Andro-Dumpsys facilitates the detection of new malware including malware variants and zero-day malware. This is further highlighted by experiments using real-world up- to-date malware samples [10].

MONET, an Android malware detection system that consists of a client module and a backend server module. The client module is a lightweight, in-device app for malware behavior monitoring and signature generation using two novel interception techniques, while the backend the server module is responsible for malware signature detection. The system can accurately describe the behaviors of an app to detect and classify malware variants and defend against to obfuscation attacks. The main focus is on classifying malware based on their behavior similarity. Moreover, it has low computational overhead and low demand for battery resources. Monet is Designed and implemented for a runtime behavior signature that can represent both the logic structures and the run time behaviors of an app. Our runtime behavior signature is effective to detect malware variants and transforming malware. It can perform a lightweight, in-device malware detection system, for Android devices [11].

The widespread use of mobile devices in comparison to personal computers has led to a new era of information exchange. The purchase trends of personal computers have started decreasing whereas the shipment of mobile devices is increasing. In addition, the increasing power of mobile devices along with portability characteristics has attracted the attention of users. Not only are such devices popular among users, but theyare favorite targets of attackers [12].

The battle to mitigate Android malware has become more critical with the emergence of new strains incorporating increasingly sophisticated evasion techniques, in turn necessitating more advanced detection capabilities. Thus, a machine learning- based anapproach based on eigenspace analysis for Android malware detection using features derived from static analysis characterization of Android applications. The eigenfaces method measures the extent of similarities and differences amongst a set of faces in a database. It is based on the premise that every face is a linear combination of the basicset of faces called eigenfaces [13].

We first implement a feature extraction tool, AppExtractor, which is designed to extract features, such as permissions or APIs, according to the predefined rules. Then we propose a feature selection algorithm, FrequenSel. Unlike existing selection algorithms which pick features by calculating their importance, FrequenSel selects features by finding the difference in their frequencies between malware and benign apps, because features which are frequently used in malware and rarely used in benign apps are moreimportant to distinguish malware from benign apps [14].

Malicious applications pose a threat to the security of the Android platform. The growing amount and diversity of these applications render conventional defenses largely ineffective and thus Android smartphones often remain unprotected from novel malware. In this paper, we propose DREBIN, a lightweight method for the detection of Android malware that enables identifying malicious applications directly on the smartphone. As the limited resources impede monitoring applications at run-time, DREBIN performs a broad static analysis, gathering as many features of an applicationas possible [15].

II. Existing System

The main contribution of the work is reduction of feature dimension to less than half of original feature-set using Genetic Algorithm such that it can be fed as input to machine learning classifiers for training with reduced complexity while maintaining their accuracy in malware classification. In contrast to exhaustive method of feature selection which requires testing for 2N different combinations, where N is the number of features, Genetic Algorithm, a heuristic searching approach based on fitness function has been used for feature selection. The optimized feature set obtained using Genetic algorithm is used to train two machine learning algorithms: Support Vector Machine and Neural Network. It is observed that a decent classification accuracy of more than 94% is maintained while working on a much lower feature dimension, thereby, reducing the training time complexity of classifiers.

Disadvantages of Existing System:

1. Less Accuracy.

III. Proposed System

Two set of Android Apps or APKs: Malware/Good ware is reverse engineered to extract features such as permissions and count of App Components such as Activity, Services, Content Providers, etc. These features are used as feature vector with class labels as Malware and Good ware represented by 0 and 1 respectively in CSV format. To reduce dimensionality of feature-set, the CSV is fed to Genetic Algorithm to select the most optimized set of features. The optimized set of features obtained is used for training two machine learning classifiers: Support Vector Machine and Neural Network. In the proposed methodology, static features are obtained from AndroidManifest.xml which contains all the important information needed by any Android platform about the Apps. Androguard tool has been used for disassembling of the APKs and getting the static features.

Advantages of Proposed System:

1. Accuracy is more

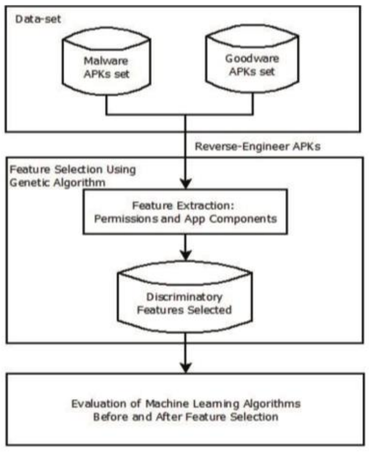

Data-set
Malware APKs set
Goodware APKs set
Reverse-Engineer APKs
Feature Selection Using Genetic Algorithm
Feature Extraction: Permissions and App Components
Discriminatory Features Selected
Evaluation of Machine Learning Algorithms Before and After Feature Selection

Fig. 1: Architecture of the proposed system

IV. Implementation

In this chapter the problem formulation, the related fundamentals and method of simulation would be discussed. To run this project, we need to work on python environment hence we implemented the code with the help of PyCharm.

A. PyCharm IDE

PyCharm is an integrated development environment (IDE) used in computer programming, specifically for the Python language. It is developed by the Czech company JetBrains. It provides code analysis, a graphical debugger, an integrated unit tester, integration with version control systems (VCSes), and supports web development with Django as well as Data Science with Anaconda. PyCharm is cross-platform, with Windows, macOS and Linux versions. The Community Edition is released under the Apache License, and there is also Professional Edition with extra features – released under a proprietary license.

- Coding assistance and analysis, with code completion, syntax and error highlighting, linter integration, and quick fixes.
- Project and code navigation: specialized project views, file structure views and quick jumping between files, classes, methods, and usages.
- Python refactoring: includes rename, extract method, introduce variable, introduce constant, pull up, push down and others.
- Support for web frameworks: Django, web2py and Flask.
- Integrated Python debugger Integrated unit testing, with line-by-line code coverage.
- Google App Engine Python development.
- Version control integration: unified user interface for Mercurial, Git, Subversion, Perforce and CVS with change lists and merge.

It competes mainly with several other Python-oriented IDEs, including Eclipse's PyDev, and the more broadly focused Komodo IDE. PyCharm provides API so that developers can write their own plugins to extend PyCharm features. Several plugins from other JetBrains IDE also workwith PyCharm. There are more than 1000 plugins which arecompatible with PyCharm.

- PyCharm Professional Edition has several license options, which differ in their features, price, and terms of use.
- PyCharm Professional Edition is free for open-source projects and for some educational uses.
- An Academic license is discounted or free.
- PyCharm Community Edition is distributed under Apache 2 license, with full source code available on GitHub.

The template is designed for, but not limited to, six authors. A minimum of one author is required for all conference articles. Author names should be listed starting from left to right and then moving down to the next line. This is the author sequence that will be used in future citations and by indexing services. Names should not be listed in columns nor group by affiliation. Please keep your affiliations as succinct as possible (for example, do not differentiate among departments of the same organization).

B. Python

Python is an interpreted high-level general-purpose programming language. Its design philosophy emphasizes code readability with its use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.

Guido van Rossum began working on Python in the late 1980s, as a successor to the ABC programming language, and first released it in 1991 as Python 0.9.0. Python 2.0 was released in 2000 and introduced new features, such as list comprehensions and a cycle-detecting garbage collection system. Python 3.0 was released in 2008 and was a major revision of the language that is not completely backward-compatible. Python 2 was discontinued with version 2.7.18 in2020.

C. User

The user should upload the dataset which contains all the features of the applications, this file sent as the input to the machine learning model, now after uploading the dataset we train and test the data, the dataset is divided into 70:30 ratio and 30% is sent for testing purpose. The user has to run the model with svm, svm with genetic algorithm, ann, ann with genetic algorithm. After running by all the algorithms, we can generate time complexity and accuracy graph, user observes that svm with genetic algorithm is the best algorithm.

D. Feature Selection

Feature selection is the process of reducing the number of input variables when developing a predictive model. It is desirable to reduce the number of input variables to both reduce the computational cost of modeling and, in some cases, to improve the performance of the model.

E. Machine Learning

a) Support Vector Machine

Support vector machine is one of most common used supervised learning techniques, which is commonly used for both classification and regression problems. The algorithm works in such a way that each data is plotted as point in n dimensional space with the feature values represents the values of each co-ordinate.

b) ANN

"Artificial neural network" refers to a biologically inspired sub-field of artificial intelligence modeled after the brain. An Artificial neural network is usually a computational network based on biological neural networks that construct the structure of the human brain. Similar to a human brain has neurons interconnected to each other, artificial neural networks also have neurons that are linked to each other in various layers of the networks. These neurons are known as nodes.

V. Proeject Planning

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests. Each test type addresses a specific testing requirement. The below mentioned are some of the various types of testing:

A. Unit Testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application. It is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

<u>Test strategy and approach</u>

Field testing will be performed manually, and functional tests will be written in detail.

Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

Features to be tested

- Verify that the entries are of the correct format.
- No duplicate entries should be allowed.
- All links should take the user to the correct page.

B. Integration Testing

Integration tests are designed to test integrated software components to determine if they run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects. The task of the integration test is to check that components or software applications, e.g., components in a software system or – one step up – software applications at the company level – interact without error.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

C. Functional Testing

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified.

D. *White Box Testing*

The box testing approach of software testing consists of black box testing and white box testing. We are discussing here white box testing which also known as glass box is testing, structural testing, clear box testing, open box testing and transparent box testing. It tests internal coding and infrastructure of a software focus on checking of predefined inputs against expected and desired outputs. It is based on inner workings of an application and revolves around internal structure testing. In this type of testing programming skills are required to design test cases. The primary goal of white box testing is to focus on the flow of inputs and outputs through the software and strengthening the security of the software.

The term 'white box' is used because of the internal perspective of the system. The clear box or white box or transparent box name denote the ability to see through the software's outer shell into its inner workings.

Developers do white box testing. In this, the developer will test every line of the code of the program. The developers perform the White-box testing and then send the application or the software to the testing team, where they will perform the black box testing and verify the application along with the requirements and identify the bugs and sends it to the developer.

The developer fixes the bugs and does one round of white box testing and sends it to the testing team. Here, fixing the bugs implies that the bug is deleted, and the feature is working fine on the application.

E. *Black Box Testing*

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box. You cannot "see" into it. The test provides inputs and responds to outputs without considering how the software works.

F. *Acceptance Testing*

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

VI. PROJECT OUTPUT

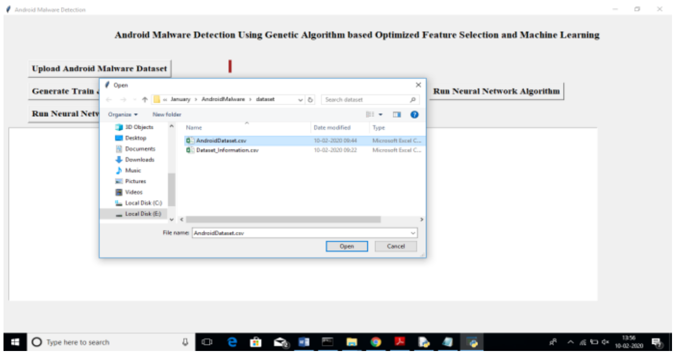
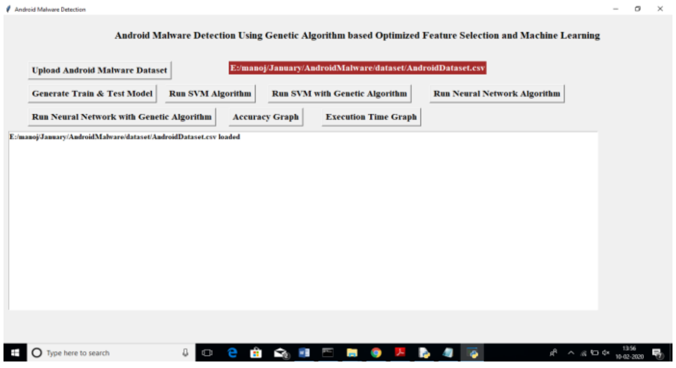
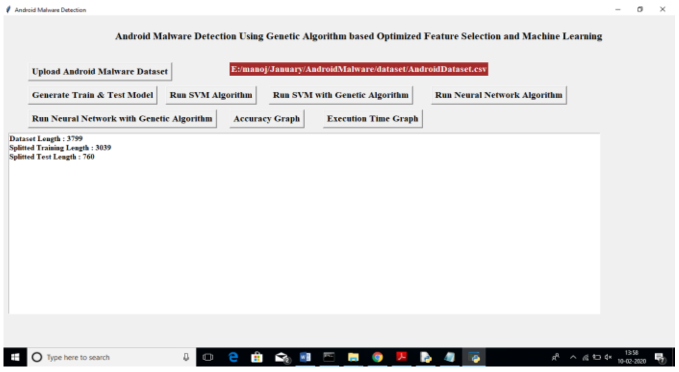

Android Malware Detection
Android Malware Detection Using Genetic Algorithm based Optimized Feature Selection and Machine Learning
Upload Android Malware Dataset
Generate Train
Run Neural Netw
Run Neural Network Algorithm
Open
AndroidDataset.csv
Dataset_Information.csv
File name: AndroidDataset.csv
Open
Cancel
Type here to search

Fig. 4.1: Upload Data Set


Android Malware Detection
Android Malware Detection Using Genetic Algorithm based Optimized Feature Selection and Machine Learning
Upload Android Malware Dataset
E:/manoj/January/AndroidMalware/dataset/AndroidDataset.csv
Generate Train & Test Model
Run SVM Algorithm
Run SVM with Genetic Algorithm
Run Neural Network Algorithm
Run Neural Network with Genetic Algorithm
Accuracy Graph
Execution Time Graph
Type here to search

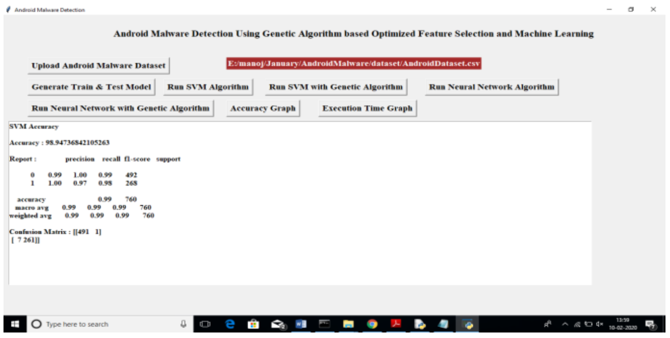
```
E:/manoj/January/AndroidMalware/dataset/AndroidDataset.csv loaded
```

Fig. 4.2: Generate Train & Test Model


Android Malware Detection
Android Malware Detection Using Genetic Algorithm based Optimized Feature Selection and Machine Learning
Upload Android Malware Dataset
E:/manoj/January/AndroidMalware/dataset/AndroidDataset.csv
Generate Train & Test Model
Run SVM Algorithm
Run SVM with Genetic Algorithm
Run Neural Network Algorithm
Run Neural Network with Genetic Algorithm
Accuracy Graph
Execution Time Graph
Type here to search

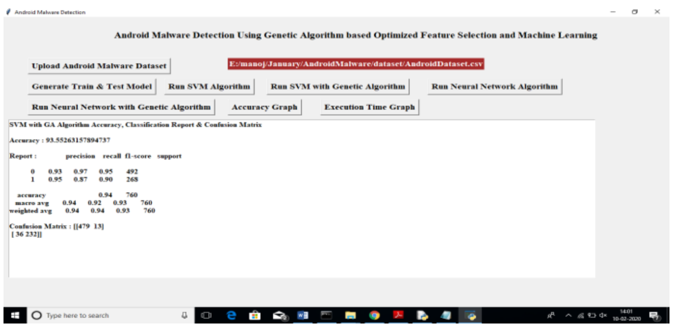
```
Dataset Length : 3799
Splitted Training Length : 3039
Splitted Test Length : 760
```

Fig. 4.3: Data Set Partition


Android Malware Detection
Android Malware Detection Using Genetic Algorithm based Optimized Feature Selection and Machine Learning
Upload Android Malware Dataset
E:/manoj/January/AndroidMalware/dataset/AndroidDataset.csv
Generate Train & Test Model
Run SVM Algorithm
Run SVM with Genetic Algorithm
Run Neural Network Algorithm
Run Neural Network with Genetic Algorithm
Accuracy Graph
Execution Time Graph
Type here to search

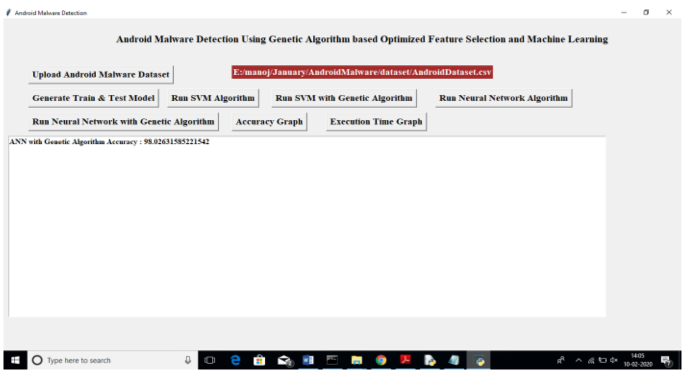
```
SVM Accuracy

Accuracy : 98.94736842105263

Report :           precision    recall  f1-score   support

           0       0.99      1.00      0.99       492
           1       1.00      0.97      0.98       268

    accuracy                           0.99       760
   macro avg       0.99      0.99      0.99       760
weighted avg       0.99      0.99      0.99       760

Confusion Matrix : [[491   1]
 [  7 261]]
```

Fig. 4.4: Run SVM Algorithm


Android Malware Detection
Android Malware Detection Using Genetic Algorithm based Optimized Feature Selection and Machine Learning
Upload Android Malware Dataset
E:/manoj/January/AndroidMalware/dataset/AndroidDataset.csv
Generate Train & Test Model
Run SVM Algorithm
Run SVM with Genetic Algorithm
Run Neural Network Algorithm
Run Neural Network with Genetic Algorithm
Accuracy Graph
Execution Time Graph
Type here to search

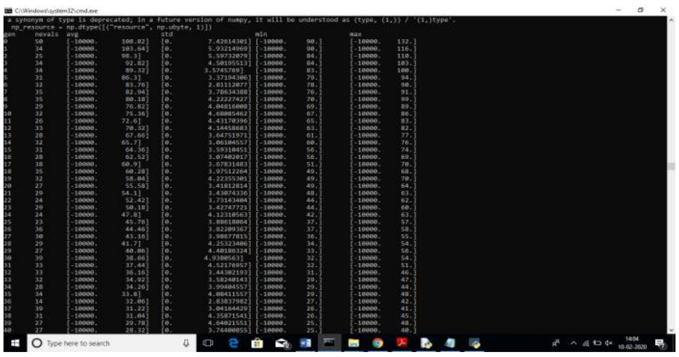
```
SVM with GA Algorithm Accuracy, Classification Report & Confusion Matrix

Accuracy : 93.55263157894737

Report :          precision    recall  f1-score   support

        0       0.93      0.97      0.95       492
        1       0.95      0.87      0.90       268

   accuracy                         0.94       760
  macro avg       0.94      0.92      0.93       760
weighted avg      0.94      0.94      0.93       760

Confusion Matrix : [[479  13]
 [ 36 232]]
```

Fig. 4.5: Run SVM with Genetic Algorithm


Android Malware Detection
Android Malware Detection Using Genetic Algorithm based Optimized Feature Selection and Machine Learning
Upload Android Malware Dataset
E:/manoj/January/AndroidMalware/dataset/AndroidDataset.csv
Generate Train & Test Model
Run SVM Algorithm
Run SVM with Genetic Algorithm
Run Neural Network Algorithm
Run Neural Network with Genetic Algorithm
Accuracy Graph
Execution Time Graph
Type here to search

```
ANN with Genetic Algorithm Accuracy : 98.02631585221542
```

Fig. 4.8: Run ANN with Genetic Algorithm



```
 a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
gen     nevals  avg                     std                     min                     max
0       50      [-10000.     108.82]    [0.       7.42614301] [-10000.    90.]       [-10000.   132.]
1       34      [-10000.     103.64]    [0.       5.93214969] [-10000.    90.]       [-10000.   116.]
2       25      [-10000.     98.3]      [0.       5.59732079] [-10000.    84.]       [-10000.   110.]
3       34      [-10000.      92.82]    [0.       4.50195513] [-10000.    84.]       [-10000.   103.]
4       34      [-10000.      89.32]    [0.      3.5745769]   [-10000.    83.]       [-10000.   100.]
5       31      [-10000.     86.3]      [0.       3.37194306] [-10000.    79.]       [-10000.    94.]
6       32      [-10000.      83.76]    [0.       2.81112077] [-10000.    78.]       [-10000.    90.]
7       35      [-10000.      82.94]    [0.       3.78634388] [-10000.    76.]       [-10000.    91.]
8       35      [-10000.      80.18]    [0.       4.22227427] [-10000.    70.]       [-10000.    99.]
9       29      [-10000.      76.82]    [0.       4.04816008] [-10000.    69.]       [-10000.    89.]
10      32      [-10000.      75.36]    [0.       4.68085462] [-10000.    67.]       [-10000.    86.]
11      26      [-10000.     72.6]      [0.       4.43170396] [-10000.    65.]       [-10000.    83.]
12      33      [-10000.      70.32]    [0.       4.14458683] [-10000.    63.]       [-10000.    82.]
13      28      [-10000.      67.66]    [0.       3.64751971] [-10000.    61.]       [-10000.    77.]
14      32      [-10000.     65.7]      [0.       3.06104557] [-10000.    60.]       [-10000.    76.]
15      31      [-10000.      64.36]    [0.       3.59310451] [-10000.    56.]       [-10000.    74.]
16      28      [-10000.      62.52]    [0.       3.07402017] [-10000.    56.]       [-10000.    69.]
17      38      [-10000.     60.9]      [0.       3.67831483] [-10000.    51.]       [-10000.    70.]
18      35      [-10000.      60.28]    [0.       3.97512264] [-10000.    49.]       [-10000.    68.]
19      32      [-10000.      58.04]    [0.       4.22355301] [-10000.    49.]       [-10000.    70.]
20      27      [-10000.      55.58]    [0.       3.41812814] [-10000.    49.]       [-10000.    64.]
21      29      [-10000.     54.1]      [0.       3.43074336] [-10000.    48.]       [-10000.    63.]
22      24      [-10000.      52.42]    [0.       3.73143404] [-10000.    44.]       [-10000.    62.]
23      29      [-10000.      50.18]    [0.       3.42747721] [-10000.    44.]       [-10000.    60.]
24      24      [-10000.     47.8]      [0.       4.12310563] [-10000.    42.]       [-10000.    63.]
25      23      [-10000.      45.76]    [0.       3.88618064] [-10000.    37.]       [-10000.    57.]
26      36      [-10000.      44.46]    [0.       3.82209367] [-10000.    37.]       [-10000.    58.]
27      30      [-10000.      43.16]    [0.       3.98677815] [-10000.    36.]       [-10000.    55.]
28      29      [-10000.     41.7]      [0.       4.25323406] [-10000.    34.]       [-10000.    54.]
29      27      [-10000.      40.06]    [0.       4.40186324] [-10000.    33.]       [-10000.    56.]
30      39      [-10000.      38.66]    [0.      4.9380563]   [-10000.    32.]       [-10000.    54.]
31      33      [-10000.      37.44]    [0.       4.52176957] [-10000.    32.]       [-10000.    51.]
32      33      [-10000.      36.16]    [0.       3.44302193] [-10000.    31.]       [-10000.    46.]
33      32      [-10000.      34.92]    [0.       3.58240143] [-10000.    29.]       [-10000.    47.]
34      28      [-10000.      34.26]    [0.       3.99404557] [-10000.    29.]       [-10000.    44.]
35      34      [-10000.     33.8]      [0.       4.08411557] [-10000.    29.]       [-10000.    48.]
36      14      [-10000.      32.06]    [0.       2.83837982] [-10000.    27.]       [-10000.    42.]
37      39      [-10000.      31.22]    [0.       3.04164429] [-10000.    26.]       [-10000.    41.]
38      31      [-10000.      31.04]    [0.       4.35871541] [-10000.    26.]       [-10000.    45.]
39      27      [-10000.      29.78]    [0.       4.64021551] [-10000.    25.]       [-10000.    48.]
40      27      [-10000.      28.32]    [0.       3.74400855] [-10000.    25.]       [-10000.    40.]
```

Fig. 4.6: SVM with Genetic Algorithm Backend

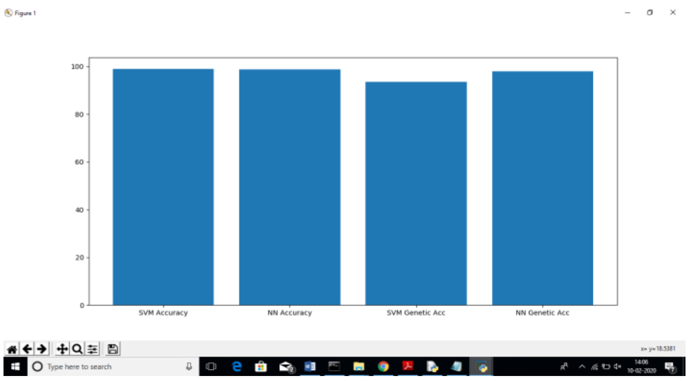
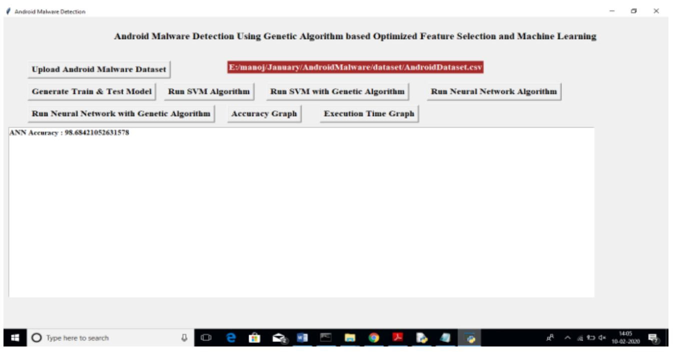


Fig. 4.9: Accuracy graph


Android Malware Detection
Android Malware Detection Using Genetic Algorithm based Optimized Feature Selection and Machine Learning
Upload Android Malware Dataset
E:/manoj/January/AndroidMalware/dataset/AndroidDataset.csv
Generate Train & Test Model
Run SVM Algorithm
Run SVM with Genetic Algorithm
Run Neural Network Algorithm
Run Neural Network with Genetic Algorithm
Accuracy Graph
Execution Time Graph
Type here to search

```
ANN Accuracy : 98.68421052631578
```

Fig. 4.7: Run ANN

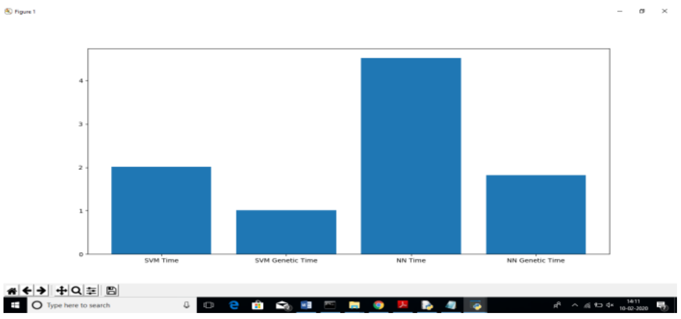


Fig. 4.10: Execution Time Graph

VII. Conclusion

As the number of threats posed to Android platforms is increasing day to day, spreading mainly through malicious applications or malwares, therefore it is especially important to design a framework which can detect such malwares with accurate results. Where signature-based approach fails to detect new variants of malware posing zero- day threats, machine learning based approaches are being used. The proposed methodology attempts to make use of evolutionary Genetic Algorithm to get most optimized feature subset which can be used to train machine learning algorithms in most efficient way.

From experimentations, a decent classification accuracy of more than 94% is maintained using Support Vector Machine and Neural Network classifiers while working on lower dimension feature-set, thereby reducing the training complexity of the classifiers. Further work can be enhanced using larger datasets for improved results and analyzing the effect on other machine learning algorithms when used in conjunction with Genetic Algorithm

VIII. Future Enhancement

In future enhancements we would like to make the software run at runtime and built strategies that would have capabilities to detect new malware which attack the application and pry on user's sensitive information.

References

[1]. S. Arshad, M. A. Shah, A. Wahid, A. Mehmood, H. Song, and H. Yu, ―SAMADroid: A Novel 3- Level Hybrid Malware Detection Model for Android Operating System,‖ IEEE Access, vol. 6, pp. 4321–4339, 2018.

[2]. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, ―MADAM: Effective and Efficient Behaviorbased Android Malware Detection and Prevention,‖ IEEE Trans. Dependable Secur. Comput., vol. 15, no. 1, pp. 83–97, 2018.

[3]. TaeGuen Kim, BooJoong Kang,Mina Rho,Sakir Sezer

,Eul Gyu Im―A Multimodal Deep Learning Method for Android Malware Detection using Various Features 21 August 2018.

[4]. J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-An, and H. Ye, ―Significant Permission Identification for Machine-Learning-Based Android Malware Detection,‖ IEEE Trans. Ind. Informatics, vol. 14, no. 7, pp. 3216–3225, 2018.

[5]. N. Milosevic, A. Dehghantanha, and K. K. R. Choo, ―Machine learning aided Android malware classification,‖ Comput. Electr. Eng., vol. 61, pp. 266–274, 2017.

[6]. Alejandro Martín García, David Camacho,Valery Naranjo―Evolving Deep Neural Networks architectures for Android malware classification June 2017.

[7]. X. Wang, Y. Yang, Y. Zeng, C. Tang, J. Shi, and K. Xu, "A Novel Hybrid Mobile Malware Detection System Integrating Anomaly Detection With Misuse Detection," Proc. 6th Int. Work. Mob. Cloud Comput. Serv., pp. 15–22, 2016.

[8]. Y. Liu, Y. Zhang, H. Li, and X. Chen, "A hybrid malware detecting scheme for mobile Android applications," in 2016 IEEE International Conference on Consumer Electronics (ICCE), 2016,

[9]. J. W. Jang, H. Kang, J. Woo, A. Mohaisen, and H. K. Kim, "AndroDumpsys: Anti-malware system based on the similarity of malware creator and malware centric information," Comput. Secur., vol. 58, pp. 125–138, 2016.

[10]. Z. Yuan, Y. Lu, and Y. Xue, "Droiddetector: android malware characterization and detection using deep learning," Tsinghua Sci. Technol., vol. 21, no. 01, pp. 114–123, 2015

[11]. H.-Y. Chuang and S.-D. Wang, "Machine Learning Based Hybrid Behavior Models for Android Malware Analysis," in 2015 IEEE International Conference on Software Quality, Reliability and Security, 2015.

[12]. S.Y. Yerima, S. Sezer, I. Muttik, Android malware detection: An eigenspace analysis approach, in Science and Information Conference (SAI), 2015, IEEE.

[13]. D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, ―Drebin: Effective and Explainable Detection of Android Malware in Your Pocket,‖ in Proceedings 2014 Network and Distributed System Security Symposium, 2014.

[14]. U. Pehlivan, N. Baltaci, C. Acarturk, and N. Baykal, "The analysis of feature selection methods and classification algorithms in permission based Android malware detection," in 2014 IEEE Symposium on Computational Intelligence in Cyber Security (CICS), 2014

[15]. Dafang Zhang,Wenjia Li,Kai Zhao―A Deep Learning Approach to Android Malware Feature Learning Detection 2014.