

**A Project Report on**  
**Permission Based Detection of Android Malware**

submitted in partial fulfillment for the award of

**Bachelor of Technology**

in

**Computer Science & Engineering**

by

**M. Yasaswini (Y20ACS508)**

**N. Anupama (Y20ACS509)**

**M. Sreevani (Y20ACS504)**

**K. Poojitha(Y20ACS486)**



Under the guidance of  
**Prof. Nagalla. Sudhakar.**

Department of Computer Science and Engineering

**Bapatla Engineering College**

(Autonomous)

(Affiliated to Acharya Nagarjuna University)

**BAPATLA – 522 102, Andhra Pradesh, INDIA**

**2023-2024**

**Department of  
Computer Science & Engineering**



**CERTIFICATE**

This is to certify that the project report entitled **Permission Based Detection Of Android Malware** that is being submitted by M.Yasaswini(Y20ACS508), N.Anupama(Y20ACS509), M.Sreevani(Y20ACS504) and K.Poojitha(Y20ACS486) in partial fulfillment for the award of the Degree of Bachelor of Technology in Computer Science & Engineering to the Acharya Nagarjuna University is a record of bonafide work carried out by them under our guidance and supervision.

Date:

**Signature of the Guide**  
**Dr. N. Sudhakar**  
**Professor**

**Signature of the HOD**  
**Dr. M. Rajesh Babu**  
**Prof. & Head**

## **DECLARATION**

We declare that this project work is composed by ourselves, that the work contained herein is our own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

**M. Yasaswini(Y20ACS508)**

**N. Anupama(Y20ACS509)**

**M. Sreevani(Y20ACS504)**

**K. Poojitha(Y20ACS486)**

## Acknowledgement

We sincerely thank the following distinguished personalities who have given their advice and support for successful completion of the work.

We are deeply indebted to our most respected guide **Dr. N. Sudhakar, Professor**, Department of CSE, for his valuable and inspiring guidance, comments, suggestions and encouragement.

We extend our sincere thanks to **M. Rajesh Babu**, Prof. & Head of the Dept. for extending his cooperation and providing the required resources.

We would like to thank our beloved Principal **Dr. Shaik Nazeer** for providing the online resources and other facilities to carry out this work.

We would like to express our sincere thanks to our project coordinator **Dr. N. Sudhakar**, Prof. Dept. of CSE for his helpful suggestions in presenting this document.

We extend our sincere thanks to all other teaching faculty and non-teaching staff of the department, who helped directly or indirectly for their cooperation and encouragement.

**M. Yasaswini(Y20ACS508)**

**N. Anupama(Y20ACS509)**

**M. Sreevani (Y20ACS504)**

**K. Poojitha(Y20ACS486)**

# Table of Contents

<b>List of Figures.....</b>	<b>viii</b>
<b>List of Tables.....</b>	<b>ix</b>
<b>Abstract.....</b>	<b>x</b>
<b>1 Introduction .....</b>	<b>1</b>
1.1 Background .....	1
1.2 Problem Statement .....	3
1.3 Motivation.....	4
1.3.1 Rising Threat of Android Malware: .....	4
1.3.2 Limitations of Existing Detection Techniques:.....	4
1.3.3 Privacy and Security Concerns: .....	5
1.3.4 The Role of Permissions in Malware Detection: .....	5
1.3.5 Empowering Users with Informed Decision-Making:.....	5
1.4 Objective .....	6
1.5 Significance.....	7
1.6 Existing System .....	8
<b>2 Literature Review .....</b>	<b>10</b>
<b>3 Proposed System .....</b>	<b>12</b>
3.1 Task .....	12
3.1.1 Project Planning and Setup: .....	12
3.1.2 Data Collection and Preparation: .....	12

3.1.3	Feature Engineering: .....	12
3.1.4	Model Selection and Development: .....	13
3.1.5	Training and Evaluation: .....	13
3.1.6	Integration and Deployment: .....	13
3.1.7	Testing and Validation: .....	14
3.1.8	Documentation and Reporting: .....	14
3.2	Dataset.....	14
3.3	Input .....	16
3.4	Output .....	16
<b>4</b>	<b>Algorithms .....</b>	<b>17</b>
4.1	Genetic Algorithm.....	18
4.1.1	Inspiration and evolution: .....	20
4.1.2	Representation: .....	20
4.1.3	Initialization: .....	20
4.1.4	Evaluation: .....	20
4.1.5	Selection:.....	20
4.1.6	Crossover: .....	21
4.1.7	Mutation:.....	21
4.1.8	Replacement:.....	22
4.1.9	Termination: .....	23
4.2	Support Vector Classifier .....	23

4.2.1	Basic Principle .....	23
4.2.2	Margin and support vectors.....	23
4.2.3	Kernel Trick .....	23
4.2.4	Regularization parameter .....	24
4.3	Artificial Neural Network .....	25
4.3.1	Sequential Structure .....	25
4.3.2	Layers.....	25
4.3.3	Activation Function .....	25
4.3.4	Compilation.....	26
4.3.5	Training.....	27
4.3.6	Evaluation and Prediction .....	27
<b>5</b>	<b>System Design .....</b>	<b>28</b>
5.1	Use Case Diagram.....	28
5.2	Class Diagram .....	29
5.3	Activity Diagram .....	30
5.4	Sequence Diagram .....	31
5.5	Collaboration Diagram.....	32
5.6	State Chart Diagram.....	33
<b>6</b>	<b>Implementation .....</b>	<b>34</b>
6.1	Requirements .....	34
6.1.1	Hardware Requirements.....	34

6.1.2	Software Requirements .....	35
6.1.3	Libraries .....	35
6.2	Code .....	37
6.2.1	Importing all necessary packages .....	37
6.2.2	Extracting permissions from input file .....	37
6.2.3	Loading Dataset .....	38
6.2.4	Optimising best feature set using Genetic Algorithm .....	38
6.2.5	SVC as estimator.....	40
6.2.6	splitting data and training ANN .....	41
6.2.7	Splitting data and training SVC .....	42
6.2.8	Loading Pickle file .....	42
6.2.9	Interface using Flask .....	44
<b>7</b>	<b>Result.....</b>	<b>45</b>
7.1	Interface .....	45
7.2	Output when Malware file is input .....	46
7.3	Output when Benign file is input .....	46
7.4	Comparison .....	47
<b>8</b>	<b>Conclusion .....</b>	<b>48</b>
<b>9</b>	<b>References .....</b>	<b>49</b>



## List of Figures

Figure 3.1	List of permissions. ....	15
Figure 3.2	List of permissions(continues).....	15
Figure 4.1	Genetic Algorithm cycle process.....	19
Figure 4.2	Selection, Crossover, Mutation operation in GA.....	21
Figure 4.3	Flow chart for GA. ....	22
Figure 4.4	Illustrates about hyperplane and margin.....	24
Figure 4.5	Represents working of ANN. ....	26
Figure 5.1	Use case Diagram. ....	28
Figure 5.2	Class Diagram ....	29
Figure 5.3	Activity Diagram ....	30
Figure 5.4	Sequence Diagram.....	31
Figure 5.5	Collaboration Diagram ....	32
Figure 5.6	Start Chart Diagram.....	33
Figure 7.1	User Interface ....	45
Figure 7.2	Result when malware APK is input.....	46
Figure 7.3	Result when benign APK is input.....	47

**List of Tables**

Table 7.1 Comparison .....47

## **Abstract**

The proliferation of Android devices has brought unprecedented convenience to users worldwide. However, this widespread adoption has also attracted the attention of malicious actors seeking to exploit vulnerabilities within the Android ecosystem. Traditional methods of detecting Android malware, such as signature-based detection and heuristic analysis, often fall short in identifying evolving and sophisticated threats.

To address this challenge, our project focuses on developing a permission-based detection system tailored specifically for Android malware. Leveraging the Android permission model, which requires applications to declare the permissions they require to access various device resources and data, our system scrutinizes these permissions to discern potentially malicious behaviour. By analysing the requested permissions, our system can identify anomalies and patterns indicative of malicious intent, thus enabling proactive detection of suspicious applications.

By training on a comprehensive dataset comprising labelled instances of malware and benign applications, our system learns to distinguish between legitimate and malicious behaviours based on the permissions requested by an application. This enables our system to continually adapt and improve its detection capabilities, staying ahead of emerging threats in the Android landscape.

Our project aims to provide users with a robust defence mechanism against Android malware, empowering them to make informed decisions about the applications they install on their devices. By enhancing security and privacy protection through permission-based detection, we strive to safeguard the integrity of Android ecosystems and mitigate the risks posed by malicious software.

# 1 Introduction

The rationale behind permission-based detection lies in the inherent link between an application's requested permissions and its potential for malicious activity. Malicious applications often request excessive permissions or request permissions that are unnecessary for their stated functionality. By analysing these permission requests in depth, our system can detect anomalies and patterns indicative of malicious intent, thus providing a valuable layer of defence against Android malware.

## 1.1 Background

As the popularity of Android devices continues to rise, so does the threat posed by malicious software targeting these platforms. Traditional signature-based methods struggle to keep pace with the rapid evolution of Android malware, necessitating the development of more sophisticated detection techniques. One promising approach involves leveraging Android's permission system in conjunction with genetic algorithms (GAs) to enhance malware detection capabilities.

The Android permission system requires applications to declare the permissions they require to access sensitive device resources and user data. This system provides valuable insights into an application's behaviour and potential security risks. Malicious applications often request excessive permissions beyond their functional requirements, which can serve as indicators of suspicious behaviour. By analysing the permissions requested by Android applications, researchers can identify patterns indicative of malicious intent and develop effective detection mechanisms.

Genetic algorithms offer a powerful optimization framework inspired by the process of natural selection. They iteratively evolve solutions to complex problems by simulating the processes of selection, crossover, and mutation. In the context of Android malware detection, genetic algorithms can be employed to optimize feature selection and classification models based on permission data.

The integration of genetic algorithms with permission-based analysis involves encoding permission features into chromosomes and using genetic operators to evolve optimal detection models. Fitness functions evaluate the effectiveness of candidate solutions based on their ability to distinguish between benign and malicious applications using permission data. Through successive generations, genetic algorithms refine detection models, improving their accuracy and adaptability to evolving malware threats.

This hybrid approach addresses several limitations of traditional Android malware detection methods. It can effectively handle high-dimensional feature spaces, capture complex interactions between permissions, and adapt to dynamic changes in malware behaviour. By leveraging the rich metadata provided by Android's permission system and the optimization capabilities of genetic algorithms, researchers can develop robust and adaptive detection models capable of identifying both known and previously unseen malware variants.

Overall, the integration of permission-based analysis with genetic algorithms represents a promising direction for advancing Android malware detection. By harnessing the synergies between these techniques, researchers can develop more effective and resilient detection systems capable of mitigating the evolving threat

landscape of Android malware. Continued research and development in this field are essential to stay ahead of adversaries and protect users from emerging cyber threats.

## **1.2 Problem Statement**

The rapid proliferation of Android devices has revolutionized the way we interact with technology, enabling unprecedented levels of connectivity and productivity. However, this widespread adoption has also exposed users to the growing threat of malware targeting the Android platform. Traditional methods of detecting Android malware, such as signature-based approaches and heuristic analysis, are often inadequate in identifying sophisticated and evolving threats.

The primary challenge lies in the dynamic nature of Android malware, which continually adapts and evolves to evade detection by conventional security measures. Malicious applications leverage various techniques to disguise their true intent and circumvent detection, posing significant risks to user privacy, data security, and overall device integrity. Moreover, the sheer volume and diversity of Android applications available on digital marketplaces make it increasingly challenging for users to discern between legitimate and malicious software.

Considering these challenges, there is a critical need for innovative approaches to detecting Android malware that can effectively mitigate the risks posed by malicious software while preserving user privacy and device performance. This project aims to address this need by developing a permission-based detection system for Android malware. By leveraging the Android permission model, which governs the access that applications have to device resources and data, our system aims to identify potentially malicious behaviour based on the permissions requested by an application.

The overarching goal of this project is to enhance the security and privacy protection of Android users by providing a proactive defence mechanism against malware threats. By analysing the permissions requested by applications and identifying anomalies indicative of malicious behaviour, our system seeks to empower users to make informed decisions about the software they install on their devices. Through rigorous research and development efforts, we aim to contribute to the advancement of Android security and mitigate the risks posed by malware in the digital landscape.

## **1.3 Motivation**

The development of a permission-based detection system for Android malware emerges as a crucial endeavour, offering a proactive and privacy-preserving approach to combating the ever-evolving threat landscape.

### **1.3.1 Rising Threat of Android Malware:**

The exponential growth of Android devices has made them lucrative targets for malware developers. With millions of apps available on digital marketplaces, the risk of encountering malicious software is higher than ever before. The increasing sophistication of malware variants poses significant challenges to traditional detection methods, necessitating innovative approaches for combating this threat.

### **1.3.2 Limitations of Existing Detection Techniques:**

Signature-based detection methods struggle to keep pace with the rapid evolution of malware, as they rely on predefined patterns or signatures. Heuristic analysis techniques may yield false positives or fail to detect zero-day exploits, leaving devices vulnerable to new and unknown threats. Behavioural analysis approaches, while effective, often

require extensive computational resources and may not scale well to the vast number of apps available.

### **1.3.3 Privacy and Security Concerns:**

Malicious apps can compromise user privacy by accessing sensitive data or performing unauthorized actions on the device. Data breaches and identity theft are serious consequences of malware infections, leading to financial loss and reputational damage. Users need robust security solutions that can proactively identify and mitigate the risks posed by malicious software, safeguarding their personal information and digital assets.

### **1.3.4 The Role of Permissions in Malware Detection:**

Malicious apps can compromise user privacy by accessing sensitive data or performing unauthorized actions on the device. Data breaches and identity theft are serious consequences of malware infections, leading to financial loss and reputational damage. Users need robust security solutions that can proactively identify and mitigate the risks posed by malicious software, safeguarding their personal information and digital assets

### **1.3.5 Empowering Users with Informed Decision-Making:**

Providing users with visibility into an app's permissions empowers them to make informed decisions about the software they install on their devices. Permission-based detection systems can serve as a valuable tool for users to assess the trustworthiness of apps before installation, enhancing their overall security posture. By raising awareness about the risks of Android malware and equipping users with effective detection mechanisms, we can collectively mitigate the impact of malicious software and create a safer digital ecosystem.



## 1.4 Objective

The objective of this project is to develop a permission-based detection method for Android malware using genetic algorithms, with the following goals:

**1. Enhance Detection Accuracy:** Improve the ability to accurately identify malicious Android applications by analysing their requested permissions, thereby reducing false negatives and improving overall detection performance.

**2.Minimize False Positives:** Minimize the occurrence of false positive detections to maintain user trust and minimize disruption to legitimate app usage.

**3.Adaptability to Emerging Threats:** Design the detection method to be adaptive and resilient against evasion techniques employed by malware authors, ensuring effectiveness against newly emerging malware variants.

**4.Scalability:** Develop a scalable solution capable of handling large volumes of Android applications efficiently, enabling rapid and comprehensive analysis of app permissions.

**5.Efficiency:** Implement an efficient detection process that balances computational resources while maintaining high detection accuracy, ensuring practical feasibility for real-world deployment on mobile devices.

By achieving these objectives, the project aims to contribute to the advancement of mobile security by providing an effective and reliable method for detecting Android malware based on permission analysis using genetic algorithms.

## 1.5 Significance

Permission-based detection of Android malware using ANN and SVC, with a feature set optimized by Genetic Algorithms, represents a significant advancement in mobile security. By enhancing accuracy, adaptability, and user empowerment, this project contributes to the ongoing effort to safeguard mobile devices and users' digital experiences.

**1. Advanced Malware Detection:** Android malware poses a significant threat to users' privacy and security, with malicious apps often exploiting permissions to carry out harmful activities. By employing Artificial Neural Networks (ANN) and Support Vector Classification (SVC) algorithms, augmented with a feature set optimized through Genetic Algorithms (GA), this project presents a sophisticated approach to identifying and thwarting Android malware.

**2. Enhanced Accuracy and Efficiency:** Traditional malware detection methods often struggle to keep pace with the evolving tactics of malicious actors. By leveraging ANN and SVC, trained on a feature set refined by Genetic Algorithms, the project enhances accuracy in distinguishing between benign and malicious apps. This precision reduces false positives and negatives, thereby improving the overall efficiency of malware detection systems.

**3. Adaptive Defense Mechanism:** The utilization of ANN and SVC, coupled with Genetic Algorithm optimization, provides an adaptable defense mechanism against emerging Android malware variants. The system can continuously learn and adapt to evolving threats, ensuring robust protection for users' devices and data. This adaptability is crucial in the face of rapidly evolving malware landscapes.

**4. Empowering Users:** Permission-based detection offers users transparency and control over their device security. By analysing app permissions, users gain insights into the behaviour and intentions of installed applications, empowering them to make informed decisions about app trustworthiness. This user-centric approach fosters a safer mobile ecosystem and promotes awareness of potential security risks among users.

**5. Contribution to Mobile Security Research:** The integration of ANN, SVC, and Genetic Algorithms for permission-based Android malware detection represents a significant contribution to mobile security research. The project's methodologies and findings can serve as a foundation for further advancements in malware detection and mitigation strategies. By sharing insights and techniques with the research community, this project contributes to the collective effort to combat mobile threats.

**6. Industry Relevance and Impact:** The development of a permission-based malware detection system using state-of-the-art techniques holds immense relevance for the cybersecurity industry. As mobile threats continue to evolve, innovative solutions like ANN, SVC, and Genetic Algorithm optimization are instrumental in staying ahead of malicious actors. The project's outcomes can inform industry practices and inspire the integration of advanced detection methods into commercial security products and services.

## **1.6 Existing System**

The existing system is a permissions-based malware detection system (PerDRaML) that determines the App's maliciousness based on the usage of suspicious permissions. The system uses a multi-level-based methodology; first extract and identify the significant features such as permissions from dataset. Further, employ various machine learning models to categorize the Apps into their malicious or benign categories.

Through extensive experimentations, the proposed method successfully identifies the 5× most significant features to predict malicious Apps. The proposed method outperformed the existing techniques by achieving high accuracies of malware detection i.e., 89.7% with Support Vector Machine, 89.96% with Random Forest, 86.25% with Rotation Forest, and 89.52% with Naïve Bayes models. Moreover, this method optimized up to ~77% of the feature set as compared to the recent approaches, while improving the evaluation metrics such as precision, sensitivity, accuracy, and F-measure.

The research consists of the following major components:

1. Collecting Malicious and Benign APKs.
2. Constructing/Identifying the Features Set.
3. Filtering, Finalizing, and Extracting the Permissions (Features) Dataset.
4. Employing the Supervised ML algorithms to Classify the Android Malware.

## 2 Literature Review

We examine the existing malicious App detection schemes. Nisha et al. [1] proposed the detection of repackaged Android malware using mutual information and chi-square techniques for the selection of features. Random forest classifier was able to achieve the highest accuracy of 91.76% among employed classifiers. However, Nisha et al.'s technique is focused on the 88 uniquely identified permissions for the analysis, which can be further optimized to include only the harmful ones.

Similarly, Sandeep [2] extracted information from the applications and performed Exploratory Data Analysis (EDA). The EDA approach focuses on the detection of malware using deep learning techniques during the installation process. Sandeep's detection framework employed several options like permissions to mirror the behaviours of the applications. It achieved 94.6% accuracy when Random Forest was used as the classifier. The approach uses 331 features for the classification which can further be optimized. Li et al. [3] suggested a permission-based detection system called SIGPID on the permission usage analysis.

Li et al. employed a multi-level data pruning technique for the selection of features. Using three levels of pruning—Permission Ranking with Negative Rate (PRNR) Support Based Permission Ranking (SPR), and Permission Mining with Association Rules (PMAR)—they could identify 22 significant permissions. Similarly, Wang et al. [4] performed a Multilevel Permission Extraction (MPE) approach where they focused on automatically identifying the permission interaction that helps to distinguish between the benign and the malicious applications. Their dataset included

9736 applications from each category set—benign and malicious—and experimental results show that a detection rate of 97.88% was achieved.

Similarly, Sun et al. [5] used static analysis in the study and proposed a collection of program characteristics consisting of sensitive API calls and permissions and performed evaluations using the Extreme Learning Machine (ELM) technique. Sun et al.'s aim consists of detection which involves minimal human intervention. Sun et al. implemented an automated tool called WaffleDetector. The proposed tool showed ~97.14% accuracy using the ELM technique.

However, the results are based on a small dataset of only 1049 applications from third-party or unofficial stores. Zhu et al. [6] proposed a low-cost system for malware detection by extracting a set of system events, permissions, and sensitive APIs and calculating the permission rate according to the key features. Zhu et al.'s approach employed the ensemble Rotation Forest (RF) to construct a model for the classification of malicious and benign APKs.

The approach Symmetry 2022, 14, 718 5 of 19 yielded over 88% detection accuracy which is almost 3.3% higher when compared with SVM classifier. However, Zhu et al.'s technique is evaluated on the limited APKs dataset. Further, the proposed feature set can be optimized and evaluated with other ML classifiers to improve detection accuracy.

## **3 Proposed System**

Permission-based analysis focuses on scrutinizing the permissions requested by Android applications as a primary indicator of potential malicious behaviour. By analysing the permissions requested by an app, such as access to sensitive data or device resources, security systems can identify suspicious or potentially harmful applications.

### **3.1 Task**

The preliminary task for developing this project is detect whether the Application is malicious or benign when user uploaded a APK file.

#### **3.1.1 Project Planning and Setup:**

1. Define project objectives, goals, and deliverables.
2. Establish a timeline and allocate resources for the project.
3. Set up development environments and tools required for the project.

#### **3.1.2 Data Collection and Preparation:**

1. Collect a diverse dataset of Android applications, including both benign and malicious samples.
2. Extract metadata and permission information from each application.
3. Preprocess the data, including cleaning, filtering, and feature extraction.

#### **3.1.3 Feature Engineering:**

1. Identify relevant features from the permission data.

2. Explore techniques for feature selection and dimensionality reduction to improve model performance.

#### **3.1.4 Model Selection and Development:**

1. Research and select appropriate algorithms for permission-based malware detection.
2. Develop models using selected algorithms, considering factors like scalability, interpretability, and accuracy.
3. Experiment with different model architectures and hyperparameters to optimize performance.

#### **3.1.5 Training and Evaluation:**

1. Split the dataset into training, validation, and testing sets.
2. Train the models using the training data and validate.
3. Evaluate model performance using metrics like accuracy, precision, recall.

#### **3.1.6 Integration and Deployment:**

1. Integrate the trained models into a detection system or application for real-time malware detection.
2. Implement mechanisms for dynamic permission analysis at runtime.
3. Optimize the detection system for efficiency, scalability, and compatibility with Android devices.



### **3.1.7 Testing and Validation:**

1. Conduct thorough testing of the detection system to ensure reliability and accuracy.
2. Test the system against a variety of benign and malicious applications, including known and unknown samples.
3. Perform validation against ground truth labels or expert analysis to verify detection accuracy.

### **3.1.8 Documentation and Reporting:**

1. Document the project workflow, including methodologies, algorithms, and implementation details.
2. Prepare a comprehensive report summarizing the project objectives, methodology, results, and conclusions.
3. Create user documentation or guides for deploying and using the permission-based malware detection system.

## **3.2 Dataset**

The model is trained with labelled binary dataset where permissions are considered as features. If a particular permission is allowed, then the value is to 1 or else the value is set to 0.

The dataset is download from GitHub repository of a developer in which there are up to 400 features with 1700 distinguish values to those permissions. The target value or the class label of the dataset is either malware or benign. Some of the

permissions included in the dataset are:

```
Final > app > static > permissions.txt
1  android.permission.ACCESS_ALL_DOWNLOADS
2  android.permission.ACCESS_BLUETOOTH_SHARE
3  android.permission.ACCESS_CACHE_FILESYSTEM
4  android.permission.ACCESS_CHECKIN_PROPERTIES
5  android.permission.ACCESS_CONTENT_PROVIDERS_EXTERNALLY
6  android.permission.ACCESS_DOWNLOAD_MANAGER
7  android.permission.ACCESS_DOWNLOAD_MANAGER_ADVANCED
8  android.permission.ACCESS_DRM_CERTIFICATES
9  android.permission.ACCESS_EPHEMERAL_APPS
10 android.permission.ACCESS_FM_RADIO
11 android.permission.ACCESS_INPUT_FLINGER
12 android.permission.ACCESS_KEYGUARD_SECURE_STORAGE
13 android.permission.ACCESS_LOCATION_EXTRA_COMMANDS
14 android.permission.ACCESS_MOCK_LOCATION
15 android.permission.ACCESS_MTP
16 android.permission.ACCESS_NETWORK_CONDITIONS
17 android.permission.ACCESS_NETWORK_STATE
18 android.permission.ACCESS_NOTIFICATIONS
19 android.permission.ACCESS_NOTIFICATION_POLICY
20 android.permission.ACCESS_PDB_STATE
21 android.permission.ACCESS_SURFACE_FLINGER
22 android.permission.ACCESS_VOICE_INTERACTION_SERVICE
23 android.permission.ACCESS_VR_MANAGER
```

Figure 3.1 List of permissions.

```
Final > app > static > permissions.txt
23 android.permission.ACCESS_VR_MANAGER
24 android.permission.ACCESS_WIFI_STATE
25 android.permission.ACCESS_WIMAX_STATE
26 android.permission.ACCOUNT_MANAGER
27 android.permission.ALLOW_ANY_CODEC_FOR_PLAYBACK
28 android.permission.ASEC_ACCESS
29 android.permission.ASEC_CREATE
30 android.permission.ASEC_DESTROY
31 android.permission.ASEC_MOUNT_UNMOUNT
32 android.permission.ASEC_RENAME
33 android.permission.AUTHENTICATE_ACCOUNTS
34 android.permission.BACKUP
35 android.permission.BATTERY_STATS
36 android.permission.BIND_ACCESSIBILITY_SERVICE
37 android.permission.BIND_APPWIDGET
38 android.permission.BIND_CARRIER_MESSAGING_SERVICE
39 android.permission.BIND_CARRIER_SERVICES
40 android.permission.BIND_CHOOSER_TARGET_SERVICE
41 android.permission.BIND_CONDITION_PROVIDER_SERVICE
42 android.permission.BIND_CONNECTION_SERVICE
43 android.permission.BIND_DEVICE_ADMIN
44 android.permission.BIND_DIRECTORY_SEARCH
45 android.permission.BIND_DREAM_SERVICE
```

Figure 3.2 List of permissions(continues).

### **3.3 Input**

Users will be prompted to upload an APK file containing the Android application they want to analyse. The system should provide an interface for users to browse and select the APK file from their local storage. Users will be given the option to choose between the Artificial Neural Network (ANN) or Support Vector Classifier (SVC) model for malware detection. This selection will determine which machine learning algorithm will be used to analyse the uploaded APK file.

### **3.4 Output**

Once the user uploads the APK file and selects the desired model, the system will analyse the permissions requested by the application. The selected machine learning model (ANN or SVC) will then predict whether the application is malware or a safe application based on its permission set.

The prediction result will be displayed to the user, indicating whether the uploaded APK file is classified as malware or safe. Optionally, the system can provide a confidence score along with the prediction result. The confidence score indicates the level of certainty of the model in its prediction. Higher confidence scores suggest a more reliable prediction, while lower scores may indicate uncertainty. It also provides SDK version, application name and file size for the uploaded APK file

## 4 Algorithms

The model designed for the detection of Android malware based on the permissions requested by applications. Leveraging the power of Artificial Neural Networks (ANN), Support Vector Machines (SVC), and Genetic Algorithm optimization, the model aims to provide accurate and efficient malware detection capabilities, thereby enhancing the security of Android devices.

It utilizes a hybrid architecture consisting of both ANN and SVC classifiers, allowing for complementary strengths in pattern recognition and classification. The ANN component comprises multiple layers of neurons, including input, hidden, and output layers, with activation functions such as ReLU (Rectified Linear Unit) or sigmoid to introduce non-linearity. The SVC component employs a kernel-based approach to separate data points in a high-dimensional feature space, maximizing the margin between classes.

This model incorporates feature engineering techniques to extract relevant information from the permission data, including frequency of permissions, permission combinations, and presence of rare or sensitive permissions. Feature selection methods, such as genetic algorithms or recursive feature elimination, are applied to identify the most discriminative permissions and reduce dimensionality.

Model is trained on a labelled dataset of Android applications, where each application is represented by its permission profile and labelled as benign or malicious. The model's performance is evaluated using standard evaluation metrics such as accuracy, precision, recall, F1-score.

The developed model can be integrated into existing security frameworks or deployed as a standalone tool for real-time malware detection on Android devices. The model's optimized architecture and efficient implementation enable seamless deployment on resource-constrained devices, with minimal computational overhead. It can undergo continuous refinement and updates to adapt to emerging malware threats and evolving permission patterns. Feedback from users and ongoing research in the field of Android security inform model updates and enhancements, ensuring that model remains effective and up to date.

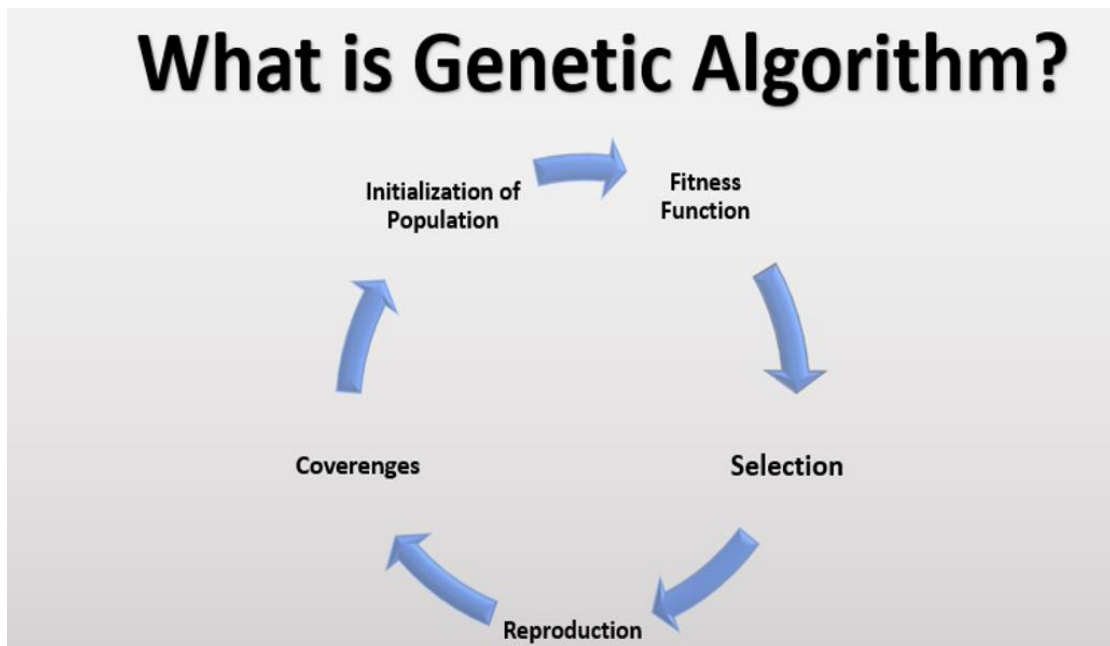
By leveraging the model, users can proactively detect and mitigate potential malware threats on Android devices, thereby enhancing device security and protecting sensitive data from unauthorized access and exploitation.

## **4.1 Genetic Algorithm**

Genetic Algorithms (GAs) are adaptive heuristic search algorithms that belong to the larger part of evolutionary algorithms. Genetic algorithms are based on the ideas of natural selection and genetics. These are intelligent exploitation of random searches provided with historical data to direct the search into the region of better performance in solution space. They are commonly used to generate high-quality solutions for optimization problems and search problems.

Genetic algorithms simulate the process of natural selection which means those species that can adapt to changes in their environment can survive and reproduce and go to the next generation. In simple words, they simulate “survival of the fittest” among individuals of consecutive generations to solve a problem. Each generation consists of a population of individuals and each individual represents a point in search space and

possible solution. Each individual is represented as a string of character/integer/float/bits. This string is analogous to the Chromosome.



**Figure 4.1 Genetic Algorithm cycle process.**

Genetic algorithms are based on an analogy with the genetic structure and behaviour of chromosomes of the population. Following is the foundation of GAs based on this analogy –

- 1.Genetic algorithm starts with a random population of candidate solutions.
- 2.The solutions are evaluated based on their fitness, and the best ones are selected for reproduction.
- 3.The genes of the selected solutions are combined through crossover to create new solutions.
- 4.The genes of the new solutions are randomly mutated to maintain diversity.
- 5.The process is repeated until a stopping criterion is met, such as a maximum number of generations or a satisfactory fitness level.

Here's an overview of genetic algorithms:

#### **4.1.1 Inspiration and evolution:**

Genetic algorithms are inspired by the process of natural selection and Darwinian evolution. They mimic the principles of survival of the fittest, reproduction, and genetic variation to search for optimal solutions to complex problems.

#### **4.1.2 Representation:**

In a genetic algorithm, potential solutions to the optimization problem are encoded as chromosomes, typically represented as binary strings or arrays of real numbers. Each chromosome represents a candidate solution in the search space.

#### **4.1.3 Initialization:**

The genetic algorithm starts with an initial population of chromosomes, which are randomly generated or initialized. The population size is typically determined based on the problem's complexity and computational resources.

#### **4.1.4 Evaluation:**

Each chromosome in the population is evaluated using a fitness function, which quantifies how well the solution performs with respect to the problem's objective. The fitness function guides the search process by providing a measure of the quality of each solution.

#### **4.1.5 Selection:**

The selection process determines which chromosomes are chosen to proceed to the next generation based on their fitness scores. Selection methods such as roulette wheel

selection, tournament selection, or rank-based selection are commonly used to favour fitter chromosomes while maintaining diversity in the population.

#### 4.1.6 Crossover:

During the crossover (recombination) phase, pairs of selected chromosomes are combined to produce offspring. Crossover points are randomly selected along the chromosomes, and genetic material is exchanged between parent chromosomes to create new solutions.

#### 4.1.7 Mutation:

Mutation introduces genetic diversity into the population by randomly altering individual genes or bits within chromosomes. Mutation helps prevent premature convergence to suboptimal solutions and allows the genetic algorithm to explore new regions of the search space.

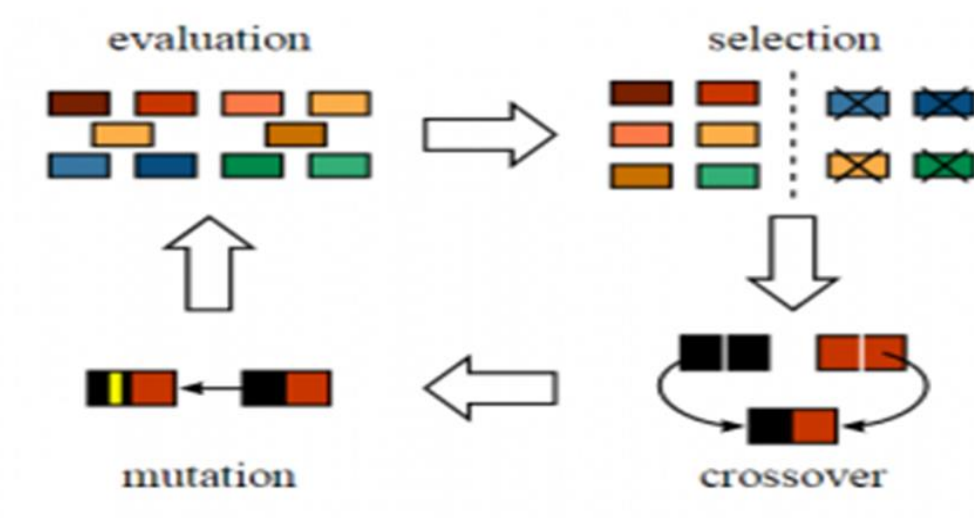


Figure 4.2 Selection, Crossover, Mutation operation in GA.



#### 4.1.8 Replacement:

Replacement involves determining which individuals from the current population will be replaced by offspring in the next generation. Various replacement strategies can be used, such as elitism, generational replacement, or steady-state replacement.

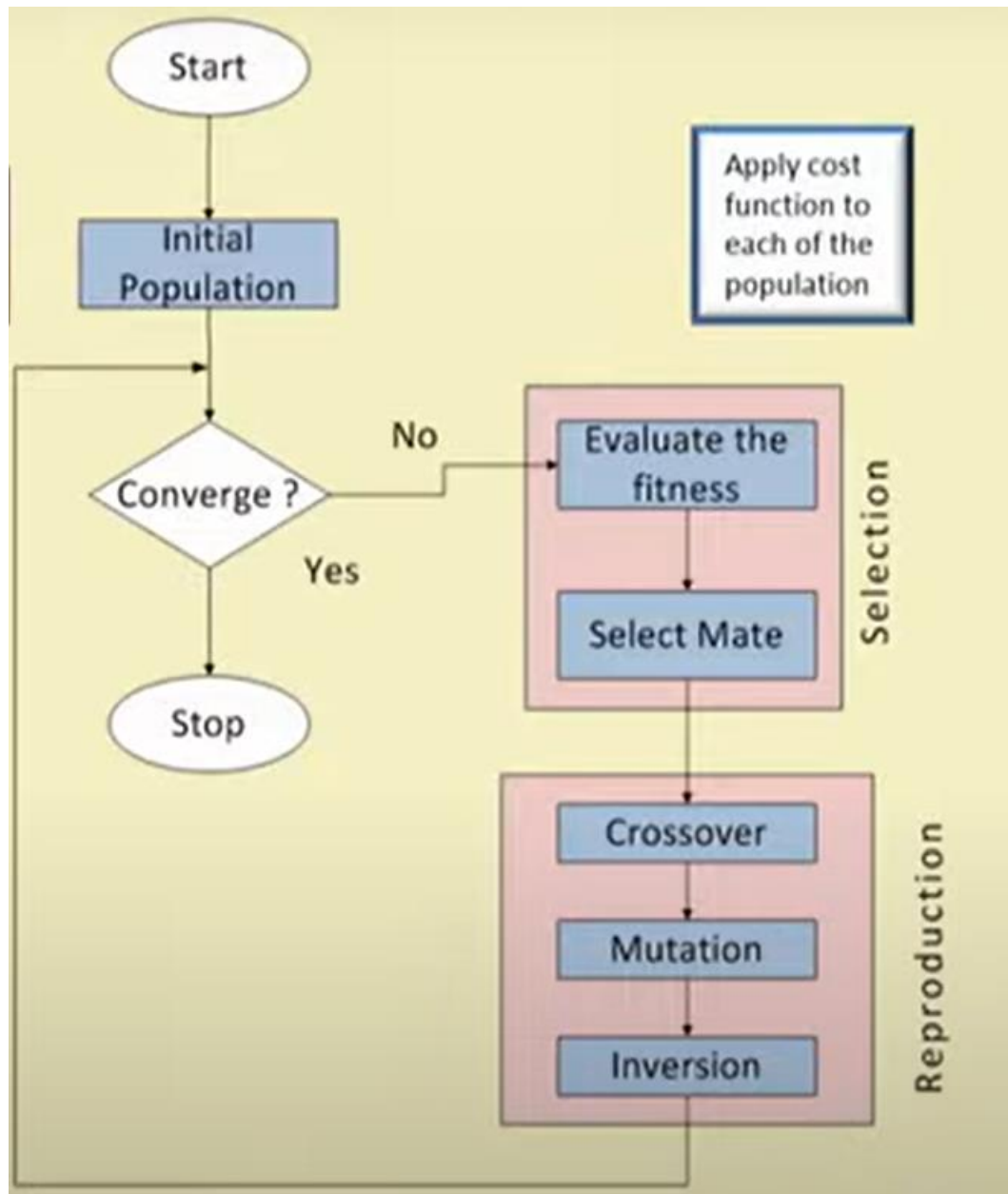


Figure 4.3 Flow chart for GA.

### **4.1.9 Termination:**

The algorithm terminates when one or more termination conditions are met, indicating that the optimization process should stop. Common termination conditions include reaching maximum number of generations, achieving a satisfactory solution quality.

## **4.2 Support Vector Classifier**

Support Vector Classifier (SVC) is a supervised machine learning algorithm used primarily for classification tasks. It's an extension of the Support Vector Machine (SVM) algorithm, designed specifically for classification problems.

### **4.2.1 Basic Principle**

The key objective of SVC is to maximize the margin, which is the distance between the hyperplane and the nearest data points from each class. Maximizing the margin helps in achieving better generalization and robustness of the model.

### **4.2.2 Margin and support vectors**

Support vectors are the data points closest to the hyperplane and have the most significant influence on determining the optimal hyperplane. The margin is defined as the distance between the hyperplane and the support vectors. SVC aims to maximize this margin. By maximizing the margin, SVC not only separates the classes but also improves the model's ability to classify unseen data accurately.

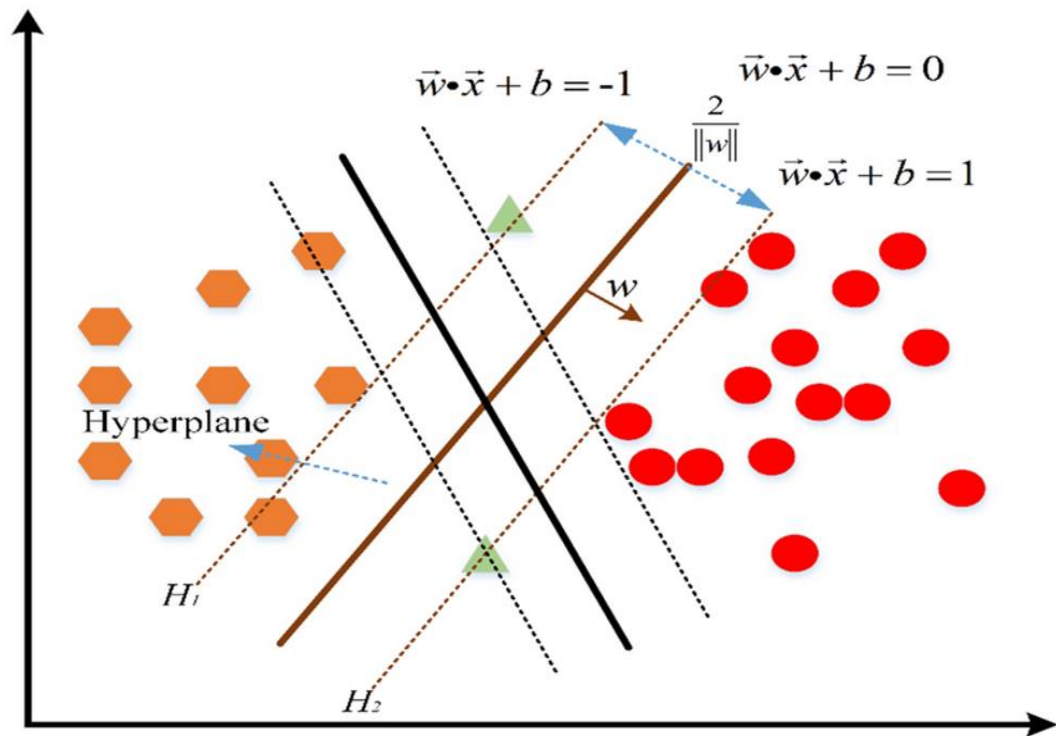
### **4.2.3 Kernel Trick**

SVC can efficiently handle non-linearly separable data by using the kernel trick. The kernel trick implicitly maps the input data into a higher-dimensional space where it becomes linearly separable. Common kernel functions used in SVC include linear,

polynomial, radial basis function (RBF), and sigmoid kernels. These kernels allow SVC to capture complex decision boundaries and achieve better classification performance on non-linear data.

#### 4.2.4 Regularization parameter

SVC introduces a regularization parameter ( $C$ ) to balance between maximizing the margin and minimizing the classification error. A smaller value of  $C$  leads to a wider margin but may allow some misclassification of training examples. A larger value of  $C$  results in a narrower margin but reduces the misclassification of training examples.



**Figure 4.4** Illustrates about hyperplane and margin.

Overall, SVC is a versatile and powerful algorithm for classification tasks, capable of handling both linear and non-linear data with appropriate kernel selection and parameter tuning.

## **4.3 Artificial Neural Network**

An Artificial Neural Network (ANN) Sequential Model is a type of neural network architecture used in machine learning and deep learning for various tasks such as classification, regression, and sequence prediction. The sequential model is a linear stack of layers where each layer is densely connected to the next one. Let's break down the components and workings of an ANN sequential model.

### **4.3.1 Sequential Structure**

The sequential model in Keras, a popular deep learning library, represents a linear stack of layers. This means that each layer in the network feeds its output to the next layer, forming a sequential flow of information. Layers can be added to the model one by one, and Keras automatically connects each new layer to the previous one. This simple structure makes it easy to create and understand complex neural network architectures.

### **4.3.2 Layers**

In Keras Sequential Model, layers are the building blocks of the network. Different types of layers can be added to the model. Each layer performs specific operations on the input data and transforms it into a more useful representation for the next layer.

The first layer added to the model specifies the input shape, which defines the shape of the input data that the model expects.

### **4.3.3 Activation Function**

Activation functions are applied to the outputs of each layer to introduce non-linearity into the model, allowing it to learn complex patterns in the data. Common activation functions include ReLU (Rectified Linear Unit), sigmoid, tanh (hyperbolic tangent),

and softmax for different types of layers and tasks. Activation functions help the model to approximate non-linear functions and make the model more expressive.

#### 4.3.4 Compilation

After adding layers to the sequential model, it needs to be compiled before training. During compilation, the model is configured with the optimizer, loss function, and evaluation metrics. The optimizer determines the optimization algorithm used to update the weights of the network during training. The loss function measures the difference between the predicted output of the model and the true output for a given input.

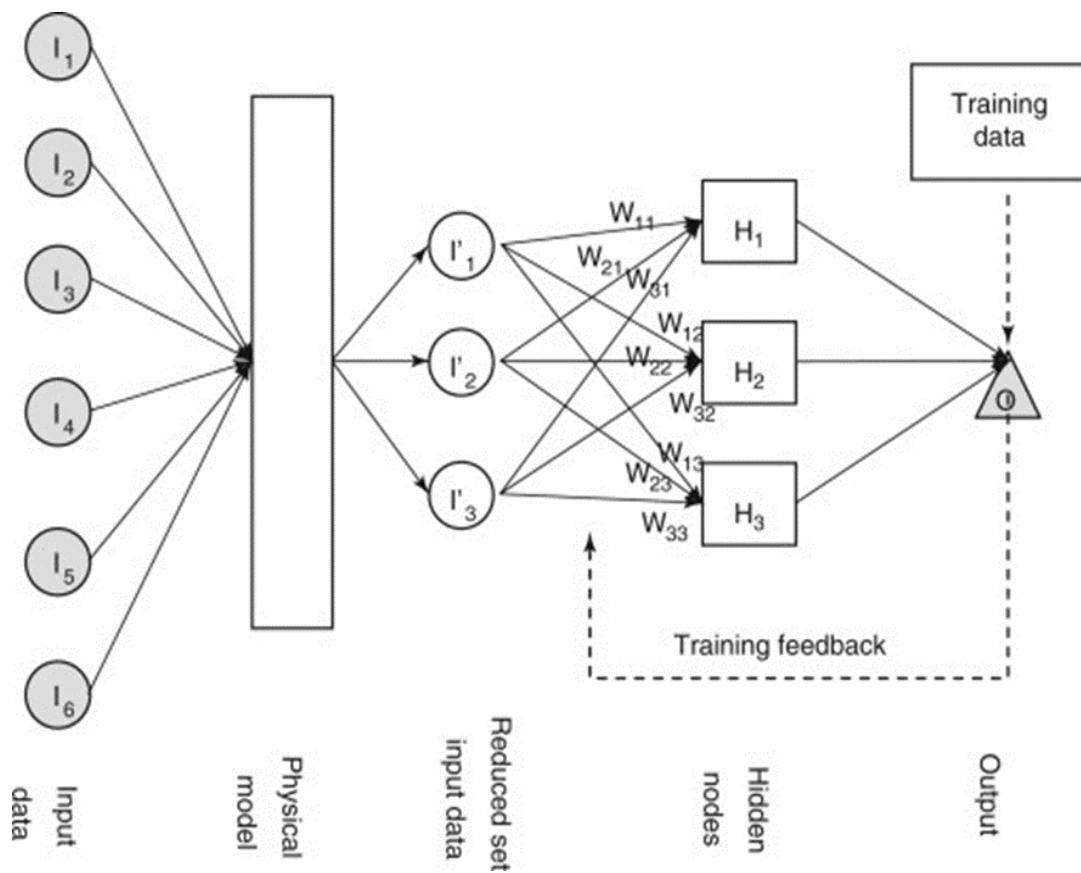


Figure 4.5 Represents working of ANN.

### **4.3.5 Training**

Once compiled, the sequential model can be trained on a dataset using the `fit()` method. During training, the model adjusts its weights based on the optimization algorithm and the provided training data. Training typically involves iterating over batches of data multiple times (epochs), where the model learns to minimize the loss function by updating its parameters (weights and biases)

### **4.3.6 Evaluation and Prediction**

After training, the model can be evaluated on a separate validation set using the `evaluate()` method, which computes the model's performance metrics. Once trained and evaluated, the model can be used to make predictions on new, unseen data using the `predict()` method.

## 5 System Design

The basic architecture of the proposed model develops a robust and scalable solution for permission-based detection of Android malware, optimized with Support Vector Classifier (SVC) and Artificial Neural Network (ANN) using Genetic Algorithm (GA).

### 5.1 Use Case Diagram

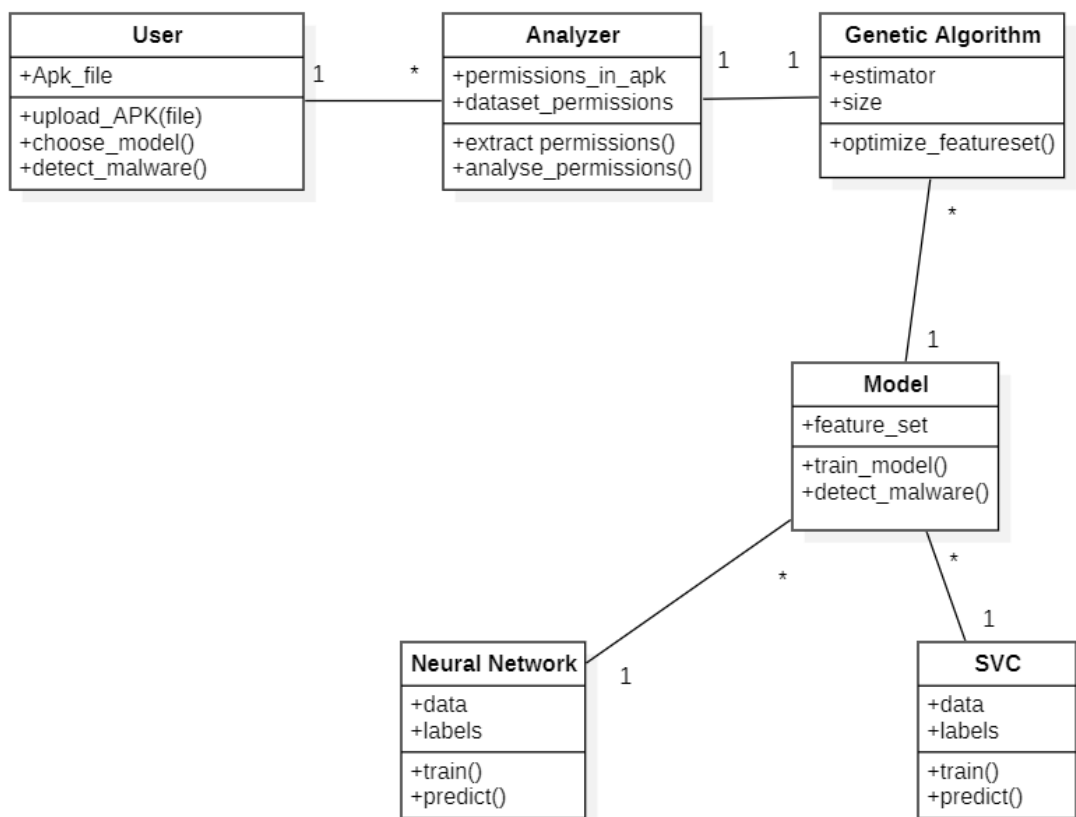
Use-case diagrams describe the high-level functions and scope of a system. These diagrams also identify the interactions between the system and its actors.



Figure 5.1 Use case Diagram.

## 5.2 Class Diagram

Class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application. A class diagram is a type of static structure diagram in the Unified Modelling Language (UML) that represents the structure and behaviour of a system or application. It depicts the classes in the system, their attributes, methods, relationships with other classes, and constraints.



**Figure 5.2 Class Diagram**



### 5.3 Activity Diagram

Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent. In UML, an activity diagram provides a view of the behaviour of a system by describing the sequence of actions in a process.

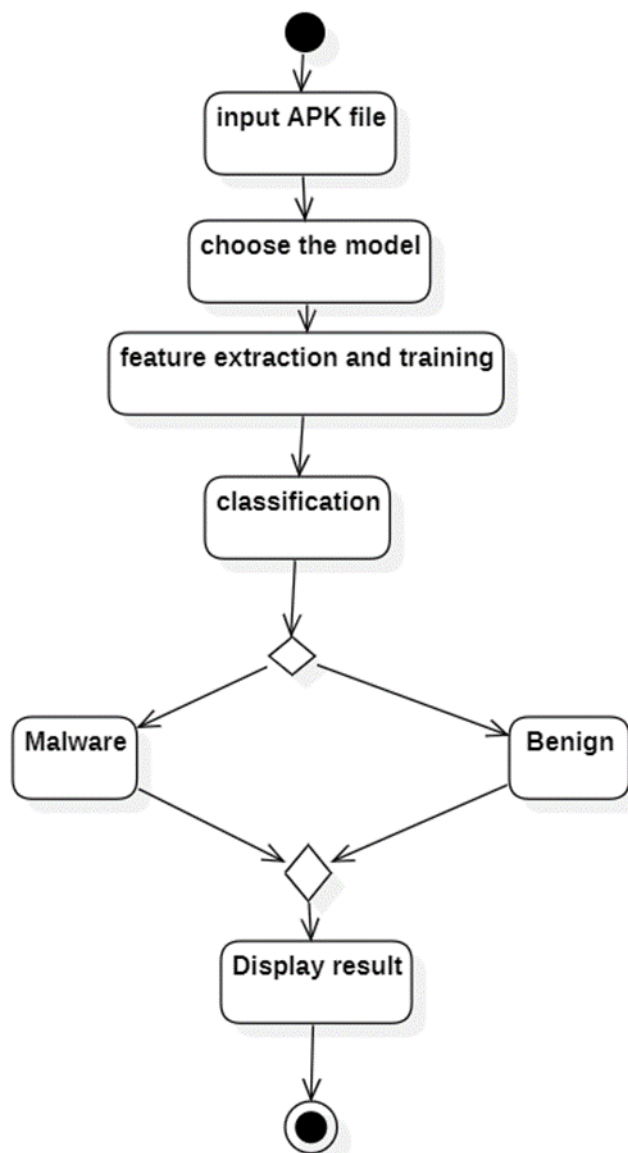
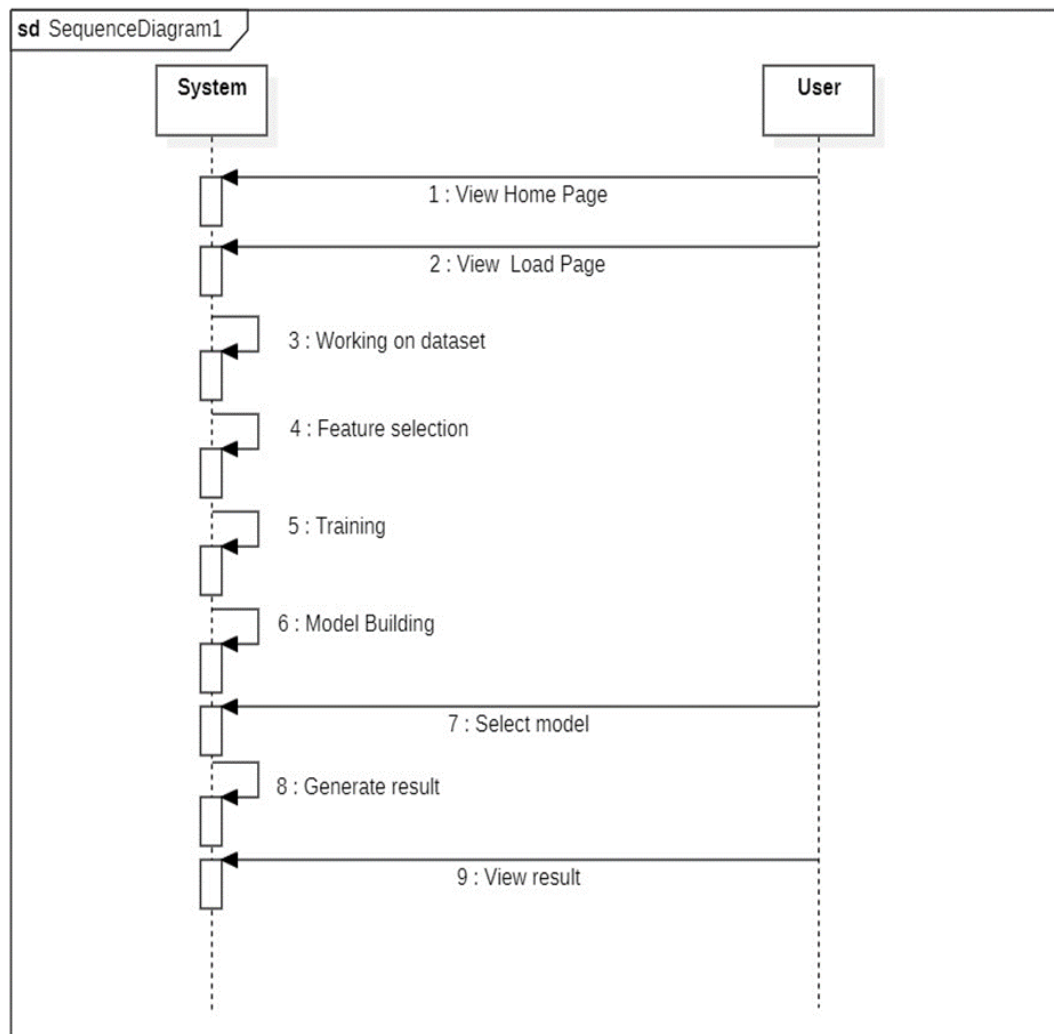


Figure 5.3 Activity Diagram

## 5.4 Sequence Diagram

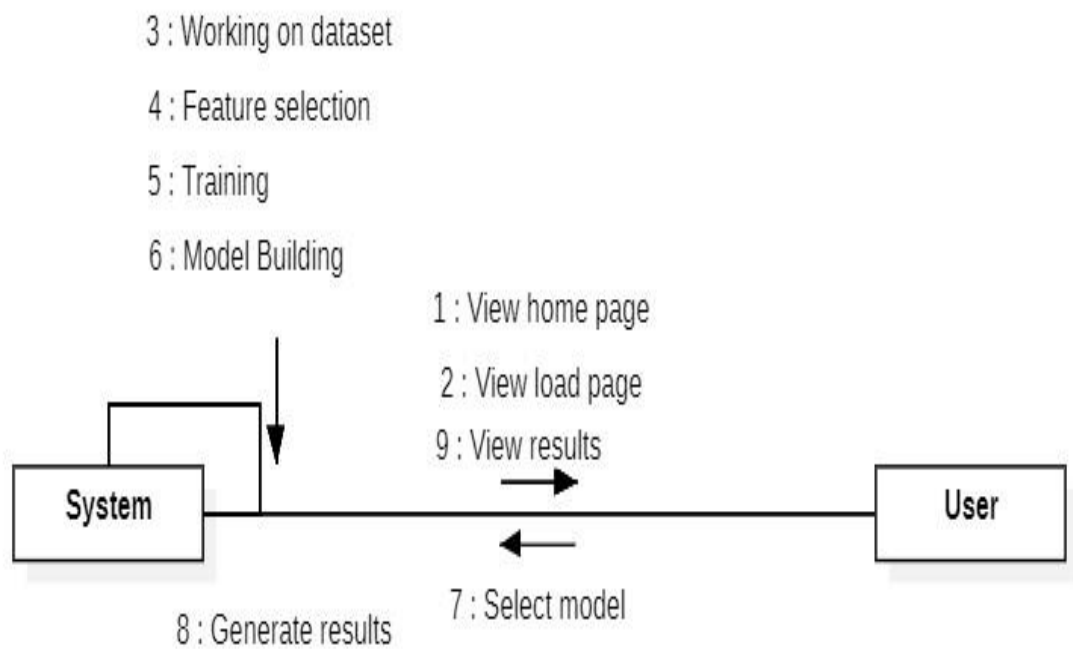
A sequence diagram is a type of interaction diagram because it describes how—and in what order—a group of objects works together. The Figure 4.4 shows the sequence diagram representation of the system. These diagrams are used by software developers and business professionals to understand requirements for a new system or to document an existing process. Much like the class diagram, developers typically think sequence diagrams were meant exclusively for them.



**Figure 5.4 Sequence Diagram**

## 5.5 Collaboration Diagram

The collaboration diagram is used to show the relationship between the objects in a system. Both the sequence and the collaboration diagrams represent the same information but differently. Instead of showing the flow of messages, it depicts the architecture of the object residing in the system as it is based on object-oriented programming.



**Figure 5.5 Collaboration Diagram**

## 5.6 State Chart Diagram

A state diagram, also known as a state machine diagram or state chart diagram, is an illustration of the states an object can attain as well as the transitions between those states in the Unified Modelling Language (UML). The Figure 4.5 shows the state chart diagram representation of system.

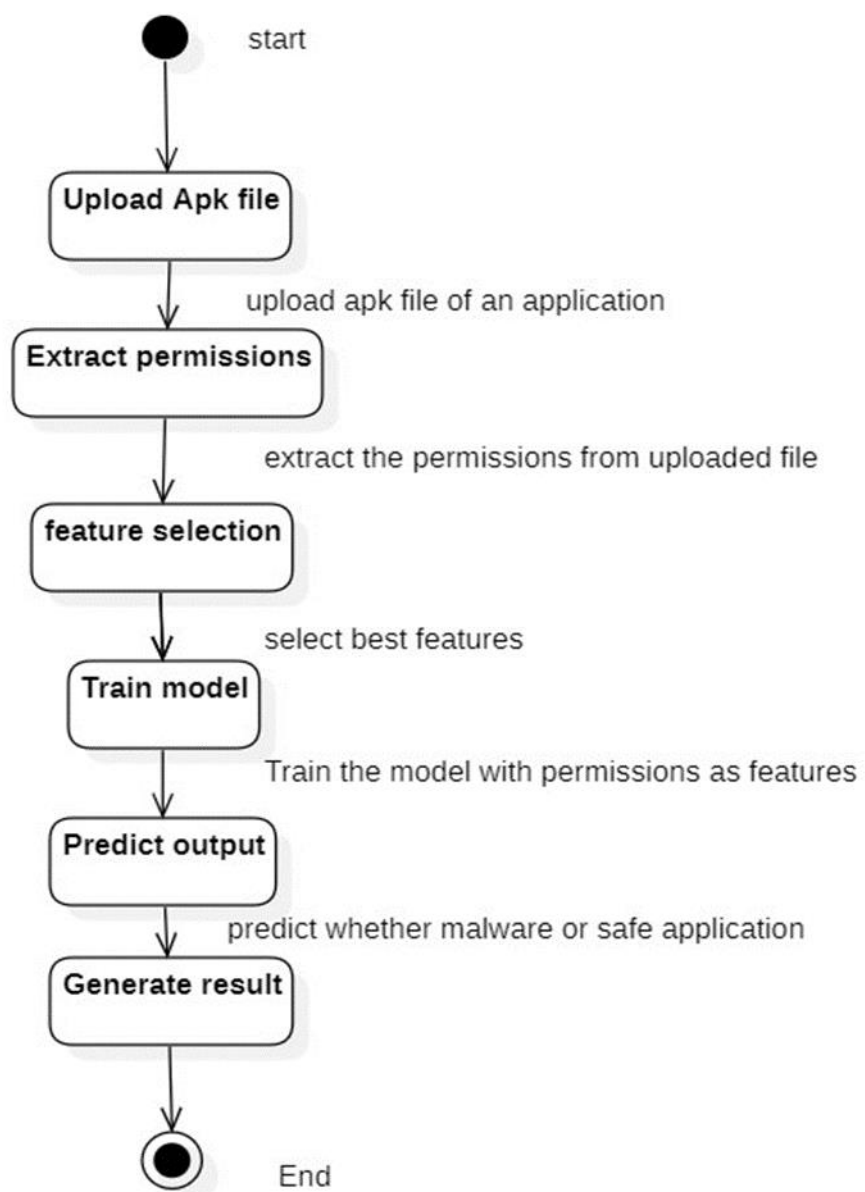


Figure 5.6 Start Chart Diagram

## **6 Implementation**

We develop a robust system for detecting Android malware based on app permissions. By analysing the permissions requested by apps, the system will be able to identify suspicious or potentially harmful behaviour, thereby enhancing the security of Android devices.

### **6.1 Requirements**

requirements are critical to the success of a project. By meeting these requirements, you can ensure that the project's software runs smoothly, reduces the risk of compatibility issues, and ensures that all stakeholders can access and use the project's software. By ignoring these requirements, you risk encountering compatibility issues, errors, and crashes, which can negatively impact the project's success.

#### **6.1.1 Hardware Requirements**

Hardware requirements are essential to ensure that the project's software can run efficiently and effectively. These requirements include the minimum specifications for the CPU, RAM, storage, and other hardware components. By meeting these requirements, the project's software can run smoothly, reducing the risk of crashes, errors, and other issues.

1. RAM (min 16GB)
2. Hard Disk (min 128GB)
3. CPU.
4. X64 based Processor.
5. 64-bit operating system

### 6.1.2 Software Requirements

Software requirements are also critical to the success of a project. These requirements include the specific versions of software and operating systems that are compatible with the project's software. By ensuring that the project's software is compatible with the required software and operating systems, you can reduce the risk of compatibility issues and ensure that the project's software functions as intended.

1. Software: Python 3.10 or high version
2. IDE: Visual Studio Code

### 6.1.3 Libraries

Libraries play a crucial role in software projects by providing pre-written code and functionality that developers can leverage to expedite development, improve code quality, and enhance the capabilities of their applications.

1. **Keras==2.4.3:** Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, Theano, or Microsoft Cognitive Toolkit (CNTK). It allows for easy and fast prototyping, supports both convolutional networks and recurrent networks, and is user-friendly, modular, and extensible.
2. **tensorflow==2.3.1:** TensorFlow is an open-source machine learning framework developed by Google. It provides a comprehensive ecosystem of tools, libraries, and community resources for building and deploying machine learning models. TensorFlow supports deep learning, reinforcement learning, and other machine learning algorithms.

3. **androguard==3.3.5:** Androguard is a powerful tool for reverse engineering Android applications. It allows users to analyze APK files, decompile bytecode, extract resources, and inspect the inner workings of Android apps. Androguard is commonly used for malware analysis, vulnerability research, and understanding the behavior of Android apps.
4. **scikit-learn==0.23.2:** Scikit-learn is a popular machine learning library in Python, providing simple and efficient tools for data mining and data analysis. It features various algorithms for classification, regression, clustering, dimensionality reduction, and model selection. Scikit-learn is known for its easy-to-use interface and extensive documentation.
5. **matplotlib==3.3.2:** Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It provides a MATLAB-like interface for plotting 2D and 3D data, supporting a wide range of plot types, customization options, and output formats. Matplotlib is widely used in scientific computing, data analysis, and data visualization.
6. **pickleshare==0.7.5:** Pickleshare is a small library that provides a persistent dictionary-like interface for storing Python objects on disk. It is particularly useful for caching objects between Python sessions or sharing data between multiple processes. Pickleshare simplifies the process of serializing and deserializing Python objects using the pickle module.
7. **numpy==1.18.5:** NumPy is a fundamental package for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently. NumPy is essential for numerical computations, data manipulation, and linear algebra operations in Python.

## 6.2 Code

Code for implementation of Android malware detection.

### 6.2.1 Importing all necessary packages

It imports various libraries including NumPy, Pandas, scikit-learn, Keras, and Androguard. The script defines functions for loading data, selecting features using a genetic algorithm, training, evaluating SVM and ANN models, and saving the trained models. The script also uses a Flask app to provide a web interface for uploading APK files and classifying them using the trained models.

### 6.2.2 Extracting permissions from input file

This code defines a function that takes an APK file as input, extracts its list of permissions, and returns a vector of binary values representing the presence or absence of certain features in the APK.

The function **predict** takes an APK file as input and returns a vector of binary values representing the presence or absence of certain features in the APK. The function first imports the **APK** class from the **androguard.core.bytecodes.apk** module. It then defines an empty dictionary vector to store the feature values. The function creates an instance of the APK class using the input APK file and extracts the list of permissions using the **get\_permissions** method.

It then iterates over a list of features and checks if each feature is present in the list of permissions. If the feature is present, the corresponding value in the vector dictionary is set to 1, otherwise, it is set to 0. The function then converts the vector dictionary into a list of binary values and converts it into a NumPy array. The resulting array is returned as the output of the function.



### 6.2.3 Loading Dataset

This code snippet performs the following steps:

Sets the random seed for reproducibility. Reads in the Android malware dataset from a CSV file and stores it in a Pandas DataFrame. Extracts the target variable (class) from the DataFrame and stores it in a separate variable. Extracts the features (all columns except for the target variable) from the DataFrame and stores them in a separate variable. Encodes the target variable using the Label Encoder from scikit-learn to convert the class labels into numerical values.

An SVC model is instantiated as `est`. Cross-validation is performed on the model using the X and Y data frames, with 5-fold cross-validation and the negative mean squared error as the scoring metric. The mean negative mean squared error is calculated and printed as the cross-validation mean squared error before feature selection

### 6.2.4 Optimising best feature set using Genetic Algorithm

This code defines a class **GeneticSelector** for feature selection using a genetic algorithm. The genetic algorithm is a search heuristic that is inspired by the process of natural selection. It is commonly used to find approximate solutions to optimization and search problems.

The GeneticSelector class takes in several parameters:

1. **estimator:** The machine learning model used for training and evaluation.
2. **n\_gen:** The number of generations to run the genetic algorithm for.
3. **size:** The size of the population of feature subsets.
4. **n\_best:** The number of best feature subsets to select from the population.

5. **n\_rand**: The number of random feature subsets to add to the population.
6. **n\_children**: The number of children created during crossover.
7. **mutation\_rate**: The probability of mutation for each chromosome.

The **initilize** method initializes the population of feature subsets by creating an array of Boolean values representing whether each feature is included or not. It then randomly sets some of the values to False with a probability of 0.3.

The **fitness** method calculates the fitness score for each feature subset by training and evaluating the estimator on the selected features using 5-fold cross-validation. The negative mean squared error is used as the scoring metric. The scores and corresponding feature subsets are then sorted in ascending order of score.

The **select** method selects the best and random feature subsets from the sorted population. The number of best feature subsets selected is **n\_best**, and the number of random feature subsets selected is **n\_rand**.

The **crossover** method creates new feature subsets by randomly selecting a crossover point and combining the parents on either side of the crossover point. The number of children created is **n\_children**.

The **mutate** method randomly flips some of the Boolean values in the feature subset with a probability of **mutation\_rate**. The **generate** method performs selection, crossover, and mutation on the population and updates the history of the best and average scores.

The **fit** method trains the genetic selector for a specified number of generations and returns the final best feature subset.

The **support\_ property** returns the final best feature subset. Finally **plot\_scores** method plots the best and average scores over generations.

The genetic algorithm works by iteratively improving the population of feature subsets over generations. At each generation, the algorithm selects the best feature subsets, creates new feature subsets through crossover and mutation, and evaluates their fitness. The algorithm then replaces the old population with the new population and repeats the process until a stopping criterion is met (in this case, `n_gen` generations). The goal of the algorithm is to find the feature subset that results in the highest fitness score.

### 6.2.5 SVC as estimator

The code performs genetic feature selection on a dataset with features `X` and labels `Y` using a Support Vector Classifier (SVC) as the estimator. Here's a breakdown of the code:

1. Converts the input feature matrix `X` to a NumPy array.
2. Creates an instance of the **GeneticSelector** class and initializes it with the following parameters: `sel = GeneticSelector(estimator=SVC(), n_gen=7, size=200, n_best=40, n_rand=40, n_children=5, mutation_rate=0.05)`.
3. **sel.fit(X, Y)**: fits the GeneticSelector object to the input data `X` and labels `Y`. During the fitting process, the genetic algorithm is run for `n_gen` generations, and the fitness of each individual is calculated as the accuracy score of the SVC model trained on the selected features.
4. Generates a plot of the fitness scores of the best individual in each generation.

5. Saves the GeneticSelector object to a file named **ga.pkl** in the specified directory using the pickle module. This allows the object to be loaded and used later without having to re-run the genetic algorithm.

### 6.2.6 splitting data and training ANN

This code defines a neural network model using the Keras library and trains it on the input data `X_train` and labels `y_train` using the genetic feature selection results from the `sel` object. Here's a breakdown of the code:

1. Creates an instance of the Sequential model from the Keras library.
2. Adds a dense (fully connected) layer with 256 neurons to the neural network and specifies the number of input features, which is set to 302 based on the number of selected features from the genetic feature selection step.
3. Adds dropout regularization to the neural network with a dropout rate of 0.2.
4. Adds another 2 dense layer with 128 neurons and 1 dense layer with 32 neurons to the neural network and adds dropout regularization to the neural network with a dropout rate of 0.2 between each layer.
5. Adds the output layer with 1 neuron and a sigmoid activation function, which is suitable for binary classification problems and compiles the neural network with the stochastic gradient descent (SGD) optimizer, binary cross-entropy loss function, and accuracy metric.
6. Trains the neural network on the input data `X_train` and labels `y_train` using the genetic feature selection results from the `sel` object. The `fit` method trains the neural network for 175 epochs with a batch size of 32.

7. Evaluates the performance of the trained neural network on the test data `X_test` and labels `y_test` using the genetic feature selection results from the `sel` object.
8. Finally, the trained neural network model `AN` is being serialized and saved to a file named **ANN\_GA.pkl** in the specified directory.

### 6.2.7 Splitting data and training SVC

This performs a train-test split on the input data `X` and labels `Y`. It then defines a parameter grid for a Support Vector Classifier (SVC) using the **GridSearchCV** function from the scikit-learn library. The `GridSearchCV` function is used to perform a grid search over the parameter space and find the best hyperparameters for the SVC model. The trained `GridSearchCV` object is then saved to a file named **svc\_ga.pkl** in the specified directory using the **`pickle.dump()`** function.

The code also prints the best hyperparameters and the classification report, which includes metrics such as accuracy, precision, recall, and F1-score for each class in the dataset.

### 6.2.8 Loading Pickle file

This code defines a custom unpickler class **CustomUnpickler** that inherits from the **`pickle.Unpickler`** class. The **`find_class`** method is overridden to fix an issue with loading pickled objects from multiple modules. The code then unpickles the genetic selector object `sel` from the file **`./static/models/ga.pkl`** using the custom unpickler.

The code also defines a list **permission** to store app permissions read from a text file. The permissions are extracted by reading the text file line by line, removing

the newline character from each line, and appending the resulting string to the permissions list.

The **classify** function takes an APK file path, a channel identifier **ch**, and returns a classification result, app name, target SDK version, and file size. The classify function first extracts metadata from the APK file using the **meta\_fetch** function. It then creates a vector of permission values for the app.

If **ch is 0**, the function uses an ANN model to classify the app as benign or malware. The ANN model is loaded from the file **static/models/ANN.h5**. The predicted probability is obtained using the predict method of the ANN model. If the predicted probability is less than 0.02, the function returns 'Benign(safe)'. Otherwise, it returns 'Malware'.

If **ch is 1**, the function uses an SVC model to classify the app as benign or malware. The SVC model is loaded from the file **static/models/svc\_ga.pkl**. The predicted label is obtained using the predict method of the SVC model. If the predicted label is 'benign', the function returns 'Benign(safe)'. Otherwise, it returns 'Malware'.

The **meta\_fetch** function takes an APK file path and returns the app name, target SDK version, and file size. It uses the APK class to extract metadata from the APK file. The **get\_app\_name** method is used to obtain the app name, the **get\_target\_sdk\_version** method is used to obtain the target SDK version, and the file size is obtained by dividing the file size in bytes by  $1024^2$  and rounding to 2 decimal places.

### **6.2.9 Interface using Flask**

This is a Flask application that allows users to upload APK files and classify them as benign or malware using either a Neural Network or Support Vector Classifier model. The application displays the classification result, along with the app name, target SDK version, and file size.

The models are loaded from pickled files and the classification is performed using the `classify` function from the classifier module. The application also provides the accuracy of each model.

# 7 Result

## 7.1 Interface

Interface is designed using python flask frame work and html. This is the starting page of the application when the application is executed on Editor Terminal, the application is hosted on a web and the below page is opened on the browser.

It is designed in such a way that user can upload his APK file and choose either neural network or SVC to predict the result.

← → ↻ 127.0.0.1:5000

### ANDROID MALWARE DETECTION

#### APK Classification

**Algorithm**

Neural Network ▼

**Upload App**

Choose File No file chosen

Predict

#### Output

Predicted Class:

Model Accuracy:

---

**Metadata**

App Name:

Target SDK Version:

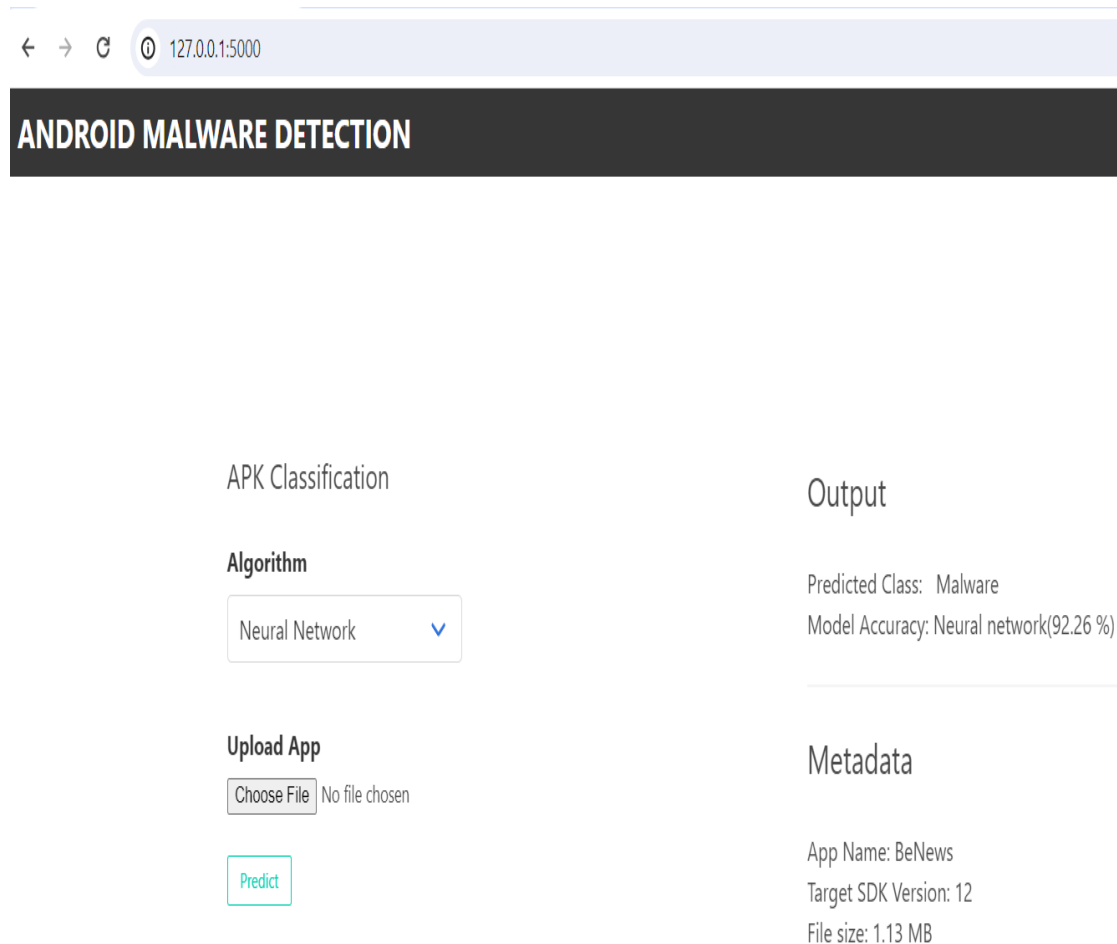
File size:

**Figure 7.1 User Interface**



## 7.2 Output when Malware file is input

When user uploaded APK file and choose Neural Network to predict, if the input APK file is malicious, then result is displayed as below.



The screenshot shows a web browser window with the address bar displaying "127.0.0.1:5000". The page title is "ANDROID MALWARE DETECTION". The interface is divided into two main sections: "APK Classification" on the left and "Output" on the right.

**APK Classification**

- Algorithm:** A dropdown menu showing "Neural Network" with a blue downward arrow.
- Upload App:** A "Choose File" button next to the text "No file chosen". Below it is a green "Predict" button.

**Output**

Predicted Class: Malware  
Model Accuracy: Neural network(92.26 %)

---

**Metadata**

App Name: BeNews  
Target SDK Version: 12  
File size: 1.13 MB

**Figure 7.2 Result when malware APK is input.**

## 7.3 Output when Benign file is input

When user uploaded APK file and choose Neural Network to predict, if the input APK file is benign, then result is displayed as below.

File size: 1.13 MB

## 8 Conclusion

In conclusion, the project successfully demonstrates the efficacy of employing permission-based detection of Android malware, leveraging advanced algorithms such as Neural Networks and Support Vector Classification (SVC). By optimizing the feature set through Genetic Algorithms, the system achieves enhanced accuracy and robustness in identifying malicious apps.

Through the utilization of Neural Networks, the project harnesses the power of machine learning to discern intricate patterns within permission structures, enabling accurate classification of malware. Additionally, SVC adds another layer of classification, bolstering the system's ability to discriminate between benign and malicious applications.

The integration of Genetic Algorithms for feature selection further refines the model, ensuring that only the most relevant and discriminative features are utilized, thereby enhancing efficiency and reducing computational overhead.

Overall, the project underscores the potential of combining cutting-edge algorithms and optimization techniques to create a robust and effective malware detection system for Android devices. This approach not only enhances security but also contributes to the ongoing efforts to combat evolving threats in the mobile ecosystem.

## 9 References

- [1] Jannath, N.O.S.; Bhanu, S.M.S. Detection of repackaged Android applications based on Apps Permissions. In Proceedings of the 2018 4th International Conference on Recent Advances in Information Technology (RAIT), Dhanbad, India, 15–17 March 2018; pp. 1–8
- [2] Sandeep, H.R. Static analysis of android malware detection using deep learning. In Proceedings of the 2019 International Conference on Intelligent Computing and Control Systems (ICCS), Secunderabad, India, 15–17 May 2019; pp. 841–845
- [3] Li, J.; Sun, L.; Yan, Q.; Li, Z.; Srisa-An, W.; Ye, H. Significant Permission Identification for Machine-Learning-Based Android Malware Detection. *IEEE Trans. Ind. Inform.* 2018, 14, 3216–3225.
- [4] Wang, Z.; Li, K.; Hu, Y.; Fukuda, A.; Kong, W. Multilevel permission extraction in android applications for malware detection. In Proceedings of the 2019 International Conference on Computer, Information and Telecommunication Systems (CITS), Beijing, China, 28–31 August 2019; pp. 1–5.
- [5] Sun, Y.; Xie, Y.; Qiu, Z.; Pan, Y.; Weng, J.; Guo, S. Detecting android malware based on extreme learning machine. In Proceedings of the 2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech), Orlando, FL, USA, 6–10 November 2017; pp. 47–53

- [6] Zhu, H.-J.; You, Z.-H.; Zhu, Z.-X.; Shi, W.-L.; Chen, X.; Cheng, L. DroidDet: Effective and robust detection of android malware using static analysis along with rotation forest model. *Neurocomputing* 2018, 272, 638–646.
- [7] Rawat, Nishant & Singh, Avjeet & Amrita. (2023). Permission-Based Malware Detection in Android Using Machine Learning. *Ymer*. 22(2023). 1505-1517. 10.37896/YMER22.03/C4.
- [8] Ehsan A, Catal C, Mishra A. Detecting Malware by Analyzing App Permissions on Android Platform: A Systematic Literature Review. *Sensors (Basel)*. 2022 Oct 18;22(20):7928. doi: 10.3390/s22207928. PMID: 36298282; PMCID: PMC9609682.
- [9] Kshirsagar, D., & Agrawal, P. (2022). A study of feature selection methods for android malware detection. *Journal of Information and Optimization Sciences*, 43(8), 2111–2120. <https://doi.org/10.1080/02522667.2022.2133218>.