

2741 MALICIOUS APK'S DETECTION USING MACHINE
LEARNING TECHNIQUES

1221300933 ZEYAD OSAMA WAHID ELSHARKAWY

BACHELOR'S OF COMPUTER SCIENCE SOFTWARE
ENGINEERING

FACULTY OF COMPUTING AND INFORMATICS MULTIMEDIA
UNIVERSITY

JULY 2024

2741 MALICIOUS APK'S DETECTION USING MACHINE LEARNING TECHNIQUES

BY

1221300933 ZEYAD OSAMA WAHID ELSHARKAWY

PROJECT REPORT SUBMITTED IN PARTIAL FULFILMENT OF THE
REQUIREMENT FOR THE DEGREE OF

Bachelor's of Computer Science Software Engineering

in the

Faculty of Computing and Informatics

MULTIMEDIA UNIVERSITY MALAYSIA

JULY 2024

Copyright of this report belongs to Universiti Telekom Sdn. Bhd. as qualified by Regulation 7.2 (c) of the Multimedia University Intellectual Property and Commercialisation Policy. No part of this publication may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Universiti Telekom Sdn. Bhd. Due acknowledgement shall always be made of the use of any material contained in, or derived from, this report.

DECLARATION

I hereby declare that the work has been done by myself and no portion of the work contained in his report has been submitted in support of any application for any other degree or qualification of this or any other university or institution of learning.

ZEYAD

Name of candidate: ZEYAD OSAMA WAHID ELSHARKAWY

Faculty of Computing & Informatics

Multimedia University

Date: 1: 7: 2024

Acknowledgement

Foremost, I express my deepest gratitude to Allah for blessing me with the strength and ability to complete my Final Year Project Part 2 successfully. I extend my heartfelt thanks to my supervisors, Ms. Aziah Aly and Ms. Zarina Binti Che Embi, for their invaluable assistance and guidance throughout this journey, enabling me to perform at my best.

Finally, I wish to convey my profound appreciation to my family, especially my mother, for their unwavering support throughout this endeavor.

Abstract:

This study examines the problems of prejudice and unfair treatment in machine learning models used on Android Application Package (APK) datasets, with a main focus on identifying weaknesses within the Android application ecosystem. The research is divided into seven chapters, which are: an introduction that addresses the problem, a literature review, the establishment of a theoretical framework, a description of the research methodology, the implementation and simulation of the study, an evaluation of the findings of the One-Class Support Vector Machines (SVM) in comparison to other algorithms, and a concluding synthesis. This paper not only emphasizes the difficulties of bias in machine learning algorithms for threat identification but also provides significant perspectives and tactics for mitigating such biases. It establishes the foundation for future endeavors in creating fair systems for the ever-changing and growing digital realm.

DECLARATION	iv
ACKNOWLEDGMENT	v
ABSTRACT	vi
TABLE OF CONTENTS	vii
LIST OF FIGURES	viii
LIST OF TABLES	x
INTRODUCTION	1
LITERATURE REVIEW	7
THEORETICAL FRAMEWORK	29
RESEARCH METHODOLOGY	41
IMPLEMENTATION	50
EVALUATION OF FINDINGS	71
CONCLUSION	74
REFERENCES	75
APPENDIX	80

LIST OF FIGURES

Figure 2.1 - How CNN works with images

Figure 2.2 - Multilayer feedforward neural network implementation

Figure 2.3 - Machine Learning Categories

Figure 2.4 - Generic Supervised Learning Algorithm

Figure 2.5 - Generic Unsupervised Learning Algorithm

Figure 2.6 - How semi-supervised clustering works

Figure 2.7 - Generic Semi-Supervised Learning Algorithm

Figure 2.8 - Anomaly Detection Process

Figure 2.9 - Graph of K-Nearest Neighbor

Figure 3.1 - Demonstration of Supervised, Semi-Supervised and Unsupervised ML

Figure 3.2 - inner product of 2 vectors in one-class SVM

Figure 5.1 - Imports

Figure 5.2. - Installation of Kaggle

Figure 5.3 - Create kaggle folder

Figure 5.4 - Copying, permission and listing kaggle datasets

Figure 5.5 - Downloading kaggle dataset

Figure 5.6 - extracting features

Figure 5.7 - Set dataset paths and initialize variables

Figure 5.8 - Prints total benign, malware and total number of permissions

Figure 5.9 - Scaling Features

Figure 5.10 - Training the algorithms

Figure 5.11 - Results

Figure 5.12 - Computing Decision Scores Code

Figure 5.13 - Malware Scores

Figure 5.14 - Malware Scores

Figure 5.15 - Customizing Threshold

Figure 5.16 - Customizing Threshold Result

Figure 5.17 - Visualization Code

Figure 5.18 - Distribution of Decision Scores

Figure 5.19 - Precision Recall Curve

LIST OF TABLES

Table 4.1 - Comparison Between Algorithm Techniques

Table 6.1 - Comparison of Findings vs One-Class SVM

Chapter 1: Introduction

1.1.Problem Statement

The APK problem, also known as the Android application package problem, pertains to the difficulties and dangers related to installing and using Android programs (APKs) obtained from unauthorized sources or dubious app marketplaces. The issue has garnered considerable attention as a result of the broad adoption of Android devices and the growing prevalence of malicious APKs that jeopardize user privacy and device security.

The relevance of the APK issue is in its possible ramifications for people, companies, and even governments. Malicious APKs have the potential to jeopardize critical user data, including personal information, financial particulars, and login passwords. Moreover, they have the capability to facilitate illegal entry into devices, resulting in unlawful actions, breaches of data, and even remote manipulation by malicious individuals.

Currently, there is an increase in the number of Package Kit Applications (APKs), which are programs designed to function on the Android operating system. Additionally, there is a growing creation of APK malware. Due to the abundance of Android APKs, an increasing number of targeted entities are launching attacks with the intention of financially benefiting the producers of malware. Consequently, Android handsets that are infected with malware experience several detrimental consequences.

In order to tackle this issue, researchers and specialists have been diligently examining several facets of the APK problem, including as detection methods, vulnerability evaluations, user consciousness, and the efficacy of security solutions. The objective of these endeavors is to augment the security and dependability of APKs and provide consumers with dependable directives for more secure program installation.

1.2. Project Objective

- The objective is to examine and assess the utilization of machine learning and deep learning methods in the field of Android security, specifically for the purpose of detecting and analyzing malware.
- To develop a machine learning model that can accurately detect and differentiate between harmful and harmless applications.
- To assess the efficacy of the algorithms in identifying Android malware.

1.3 Project Scope

The project's scope encompasses the precise goals and parameters of the project, while the limitations pertain to the constraints or restrictions that may impact its implementation or results.

The project aims to bolster the security of Android by using machine learning and deep learning techniques on specific datasets, focusing on the examination and analysis of Android Package (APK) files. Chapter 4 will examine and contrast several deep learning methodologies, emphasizing the selected methodology based on essential characteristics.

Afterwards, the models that are created will be trained and evaluated using publicly available datasets that are especially built for scientific reasons. The project's constraint is mostly due to the datasets, particularly the lack of a dataset containing local products. Acquiring such a dataset is extremely difficult and no internet sources have yet discovered it.

1.4. Introduction to the APK Issue

The difficulties and risks related to the installation and use of Android application packages (APKs) are often referred to as the "APK problem." Android devices employ the APK file format for the installation and dissemination of applications. Users may face security and privacy issues when they come across APKs acquired from unreliable sources, unapproved app stores, or dangerous individuals.

Android users can install applications from places other than the official Google Play Store because of the platform's open-source nature. Although the flexibility offers clients several choices, it also involves some hazards. Malicious actors has the capability to generate and disseminate APKs that incorporate malevolent code, such as spyware, malware, or ransomware. This presents a substantial menace to the security and dependability of Android devices.

Malware is intentionally designed to inflict harm, pilfer data, or disrupt regular operations. Malware may penetrate any computing device that executes user applications, sometimes referred to as apps. A wealth of comprehensive data exists about the safeguarding of personal computers against malware and the mechanisms by which malware propagates. Nevertheless, our current techniques for identifying malware on mobile platforms are failing to keep up with the increasing prevalence of mobile applications on smartphones. Based on a novel approach, the current number of Android applications available in the market is 2,787,954. The system's widespread use has led to a substantial increase in the occurrence of Android malware.[9] The issue with APKs goes beyond only malware. This aims to resolve issues related to fraudulent or counterfeit programs that imitate genuine ones, which might result in financial fraud, data theft, or unlawful acquisition of personal information.[9] Furthermore, APKs acquired from unofficial sources may be deficient in crucial security features or have not undergone comprehensive testing, rendering them more vulnerable to attacks and vulnerabilities.

Individuals who unintentionally install harmful APKs subject themselves to a range of potential hazards. The hazards encompassed in this context involve the unlawful acquisition of sensitive data, impairment of device operation, financial detriment, and violations of privacy[9]. Moreover, the existence of malware or other malevolent elements might enable assailants to remotely manipulate the compromised device, resulting in additional malevolent actions.

In order to address the APK issue, several strategies have been devised, including as bolstering the security of app stores, implementing access limits based on permissions, regularly updating security protocols, and educating users on safe procedures for installing apps. Furthermore, scientists and security specialists consistently investigate methods for identifying and analyzing APKs in order to pinpoint any dangers and weaknesses, so guaranteeing a more secure application environment for Android users.

Comprehending and resolving the APK issue is essential for upholding a secure Android environment and safeguarding the privacy and integrity of users' data.

1.5. Significance and influence of the APK issue in key fields.

The issue with APKs is of great significance and has a broad influence on individuals, companies, and the whole Android ecosystem. Gaining a comprehensive understanding of its significance is vital for effectively dealing with the related hazards and ensuring the protection of users' security and privacy. The subsequent points emphasize the importance and influence of the APK issue:

User Security: The problem with APKs directly affects the security of users. Malicious APKs has the capability to jeopardize critical data, such as personal information, financial particulars, and login credentials. Unsuspecting individuals who install such APKs without realizing it are at risk of falling victim to identity theft, financial fraud, and abuses of their privacy. Ensuring a secure app environment is crucial to protect users' personal and private data.

Malicious APKs have the potential to undermine the integrity of devices, allowing attackers to gain unauthorized access and control over Android devices. This can lead to illicit behavior, malfunctions in equipment, loss of data, and perhaps even attacks by ransomware. Preserving the authenticity of devices is essential for maintaining user trust and minimizing potential damage.

The APK issue may lead to monetary losses for both people and companies. Malicious APKs have the potential to illicitly acquire sensitive financial data, partake in deceitful actions, or enroll users in premium services without their explicit permission. These behaviors have the potential to cause financial harm, impacting both people' personal money and companies' reputation and financial performance.

Organizations have substantial dangers associated with the APK issue. Installing malicious APKs on company devices or networks can result in data breaches, the compromising of important business information, and the exposure of internal systems to external threats.

Organizations should adopt strong security measures and provide comprehensive training to staff in order to successfully reduce these threats.

The issue with APKs has a negative effect on the general confidence and reliability of the Android ecosystem. Users may acquire a sense of doubt or uncertainty when it comes to installing applications from sources that are not officially recognized. This can impede the progress of app developers and restrict the options available to users. Ensuring a safe and reliable app environment is essential for fostering user trust and cultivating a robust Android ecosystem.

Legal and regulatory considerations are also relevant to the APK dilemma. Individuals that distribute harmful APKs may face legal consequences, while governing bodies and regulatory agencies may enforce more stringent norms and laws to improve application security. Adhering to these laws is crucial for developers and organizations who operate inside the Android ecosystem.

The resolution of the APK problem requires the collaborative effort of application developers, cybersecurity experts, platform providers, and users themselves. To mitigate the risks posed by the APK problem and ensure a secure app ecosystem for Android users, it is imperative to implement robust security measures, enforce stringent app vetting protocols, enhance user awareness and knowledge, and promote a culture of responsible app installation.

1.6. Existing obstacles and deficiencies in tackling the APK issue.

Despite continuous endeavors to tackle the APK issue, several obstacles and deficiencies persist. These obstacles impede the implementation of effective mitigation techniques and require further study and progress. Presently, there exist several obstacles and deficiencies in effectively dealing with the APK issue:

The exponential growth of malicious APKs being generated and disseminated is a substantial obstacle. Malicious individuals consistently discover novel methods to obscure and avoid being detected, hence posing challenges in staying abreast of evolving risks. The fast spread of harmful APKs poses a difficulty in promptly identifying and addressing them.

Malware programmers continuously modify their tactics to circumvent security mechanisms. The utilization of advanced techniques like as obfuscation, encryption, and polymorphic malware poses significant difficulties in efficiently detecting and analyzing malicious APKs. To stay ahead of constantly changing malware approaches, it is necessary to consistently engage in research and development of sophisticated technologies for detecting and analyzing malware.

The Android ecosystem exhibits significant fragmentation, characterized by a multitude of device makers, diverse Android versions, and disparate degrees of security updates. The fragmentation of devices and versions poses issues in ensuring uniform security standards and timely patching of vulnerabilities. As a result, older devices and versions are more susceptible to attacks connected to APKs.

User knowledge and Education: The largest gap is in the lack of user knowledge and education on safe app installation methods. A significant number of users lack awareness regarding the potential hazards that come with installing APKs from unauthorized sources or providing excessive access to applications. To fill this void, it is necessary to implement extensive and continuous user education initiatives aimed at promoting secure app installation methods.

Privacy problems are a broader issue related to the APK problem, which goes beyond only malware. Several APKs gather an excessive amount of user data, resulting in privacy infringements. Many users frequently fail to consider or are uninformed about the privacy ramifications associated with the permissions asked by applications. It is crucial to provide transparent data handling procedures and give consumers the ability to make well-informed privacy decisions.

Unofficial and third-party app markets provide a hurdle due to their absence of rigorous security safeguards and app vetting procedures seen in official app shops. Individuals who depend on these marketplaces may inadvertently acquire harmful APKs, so exposing themselves to substantial hazards. To resolve the APK issue, it is necessary to tackle the security vulnerabilities included in these other routes for software delivery.

Ensuring that security updates are delivered promptly to Android devices is a significant problem. Effective collaboration among device makers, carriers, and software providers is necessary to enable the timely deployment of security fixes. Nevertheless, the Android ecosystem's vulnerability is exacerbated by the tardiness of security upgrades and the absence of support for older devices.

The scalability of detection algorithms becomes an issue as the number of APKs grows. There is a need for efficient and scalable techniques to analyze APKs in large quantities without sacrificing the accuracy of detection. It is essential to provide strong and effective detection methods to manage the growing number of APKs.

To overcome these problems and bridge the current gaps, it is essential to foster collaboration among researchers, industry stakeholders, platform providers, and users. In order to properly address the APK problem and maintain a secure Android app environment, it is imperative to engage in ongoing research and development, provide better user education, implement stronger security measures, and enforce more rigorous app screening processes.

Chapter 2: Literature Review

2.1. Elucidation of the search methodology and criteria employed for article inclusion

The search methodology for this study focuses on finding scholarly articles from Google Scholar that particularly address various methods to overcoming the APK challenge. Google Scholar is a highly respected academic search engine that provides users with access to intellectual content, including research papers, conference proceedings, theses, and dissertations. Our goal is to gather a comprehensive selection of scientific resources on the APK issue and its remedies by utilizing the resources available on Google Scholar.

The search methodology involves doing a systematic search by using relevant keywords and filters to improve the precision of the results. The chosen keywords include different forms of phrases like "APK issue," "Android application package issue," "APK security," "APK detection methods," and "APK mitigation solutions." Furthermore, the selected keywords

include "Machine learning approach for addressing the APK problem," "Supervised and Unsupervised Machine learning methodologies," and "Anomaly Detection." These keywords are primarily intended to target research papers and publications that investigate different solutions to the APK problem.

To ensure the reliability and relevance of the sources, the search is limited to papers published during the recent years. Our goal is to include the most recent scientific findings and advancements in addressing the APK problem by giving priority to current studies.

The search results acquired from Google Scholar will be evaluated based on their relevance to the subject of research. Preference will be given to papers that primarily focus on various solutions, techniques, or methodologies for mitigating the APK problem. In addition, priority will be given to articles that have been published in reputable journals or conference proceedings to ensure the quality and dependability of the selected sources.

Our methodology entails use Google Scholar to do a search, with a special focus on articles that tackle the APK problem and provide solutions via machine learning approaches. The objective is to gather a collection of reliable and relevant academic sources. This technique will enable us to closely examine and assess many concepts, tactics, and advancements that researchers have proposed to effectively tackle the APK problem.

2.2. Analysis of Recent research

2.2.1 Deep learning techniques and architectures

Neural networks are a machine learning technique that emulates the structure and operation of the human brain and nervous system. The system is composed of many layers of input, hidden, and output processing units. The nodes or units inside each layer are interconnected with the nodes or units in the next levels above and below. Every hyperlink is accompanied by a relevant illustration. The inputs are multiplied by the weights of each unit, and the resulting products are then added together. The total value is then adjusted using the activation function, typically a sigmoid, hyperbolic tangent, or rectified linear unit (ReLU). These functions are utilized due

to their advantageous derivative properties, which facilitate the computation of partial derivatives of the error delta with respect to given weights[19].

Moreover, the input is limited to the specified output ranges or options provided by the sigmoid and tanh functions. The sigmoid function produces values within the range of 0 to 1, whereas the tanh function produces values within the range of -1 to +1. Saturated nonlinearity is employed when the outputs achieve a maximum level or saturation either before or at the corresponding thresholds. In contrast, ReLu is defined as the function $f(x)=\max(0,x)$ and demonstrates both non-saturating and saturating properties [19]. The function's output is passed on to the subsequent unit in the following layer. Analyzing the output of the last layer helps determine the answer to the problem. Figure 2.1 illustrates the ability of CNN, a form of deep neural network, to properly identify complex objects by obtaining hierarchical representations from a vector of low-level input[19]. The red squares in the illustration are a simplified depiction of the pixel values inside the highlighted region. Convolutional neural networks (CNNs) iteratively improve the visual characteristics of a picture at each layer, ultimately allowing them to recognize and categorize the image.

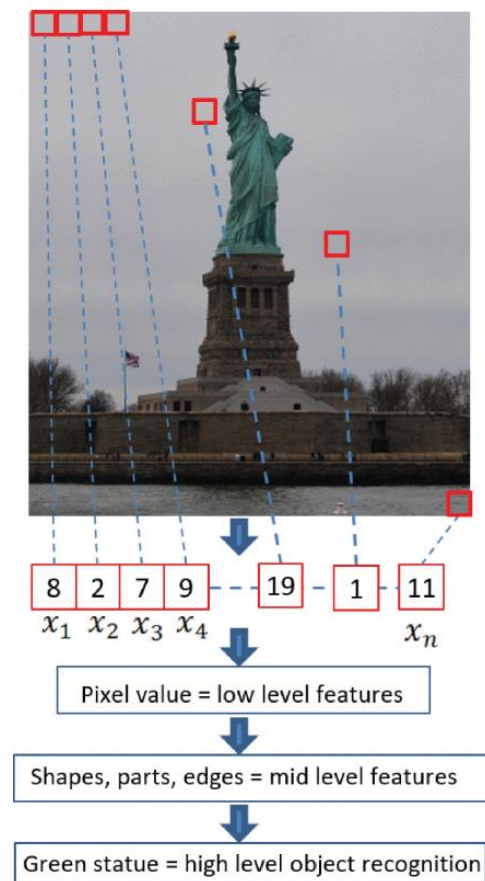


Fig 2.1 - How CNN works with images

Neural networks may be categorized into several distinct categories:

In a feedforward neural network, data moves in a single route from the input layer to the output layer. It may also pass via hidden nodes along the way. They do not form any closed loops or circular shapes. Figure 2.2 depicts a specific implementation of a multilayer feedforward neural network, where calculations of values and functions take place along the forward pass path. Z denotes the cumulative value of the weighted inputs, whereas y indicates the non-linear activation function (f) applied to Z for each layer. The bias value of the unit is represented by the symbol b , whereas the weights linking the units in adjacent layers are denoted as W with subscript letters[19].

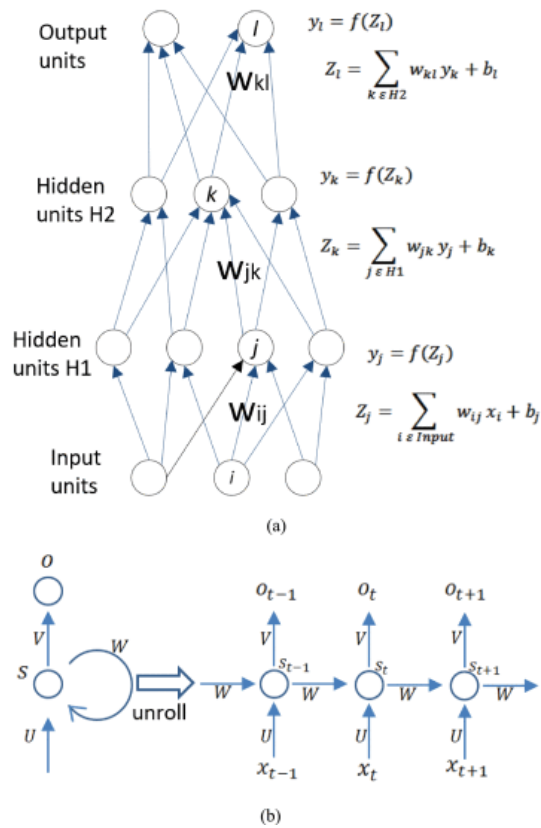


Fig 2.2 - Multilayer feedforward neural network implementation

RNNs distinguish themselves from feedforward neural networks by incorporating a recurrent cycle of processing units. A layer's output is recursively fed into itself, resulting in a feedback

loop. The output of a layer is then used as input for the following layer, which is frequently the sole layer in the network. As a result, the network may store and utilize past states to impact the current output. Recurrent neural networks (RNNs) are characterized by their capacity to handle sequential inputs and produce a corresponding sequence of output values, distinguishing them from feedforward neural networks. This divergence has been emphasized as a significant result [19]. RNN is especially useful for applications such as speech recognition and frame-by-frame video classification, where the analysis of a sequence of time-based input data is required.

Figure 2.2 illustrates the procedure of unfolding a recurrent neural network (RNN) throughout a time sequence. For example, if the input consists of a series of three words, each word would represent a layer, and the network would be extended or unfolded three times to create a three-layer RNN[19].

The figure can be mathematically interpreted in the following manner: The input is represented as x_t at time t . The common components that have been acquired and are present in every phase are U , V , and W . The value generated at a certain point in time, indicated as t , is denoted as o_t . The state at time t is represented as S_t and may be calculated using an activation function f , such as ReLU.

$$S_t = f(Ux_t + Ws_{t-1})$$

Eq 2.1

Radial basis neural networks are utilized for addressing issues pertaining to time series prediction, function approximation, and classification. The system consists of three distinct layers: input, concealment, and output. Each node in the hidden layer corresponds to a cluster center, and the radial basis function (represented as a Gaussian function) is included [19]. To do classification or inference, the output layer combines the outputs of the radial basis function and weight parameters, and the network learns to allocate the input to a center [19]

Kohonen self-organizing neural networks employ unsupervised learning to autonomously structure the network model based on the input data. The neural network consists of two completely linked layers, namely the input layer and the output layer. The output layer is configured as a two-dimensional grid. The weights represent the attributes or location of the output layer node, and there is no activation function used. The Euclidean distance between each output layer node and the input data is calculated, considering the weights. The technique described below updates the weights of the nearest node and its neighboring nodes based on the input data, in order to increase their similarity to the input data [20].

$$w_i(t+1) = w_i(t) + \alpha(t)\eta_{j*i}(x(t) - w_i(t))$$

Eq 2.2

The variable η_{j*i} represents the neighborhood function between the i th and j th nodes. The variable $w_i(t)$ represents the i th weight at time t , while $x(t)$ represents the input data at time t .

Modular neural networks are employed to partition extensive networks into smaller, distinct neural network modules for enhanced manageability. The smaller networks perform specific tasks that are then included into the overall output of the network [21].

2.2.2 Supervised, Unsupervised and Semi-Supervised machine learning.

Machine learning approaches may be classified into three categories: Supervised, Unsupervised, and Semi-Supervised learning, as seen in fig 2.3.

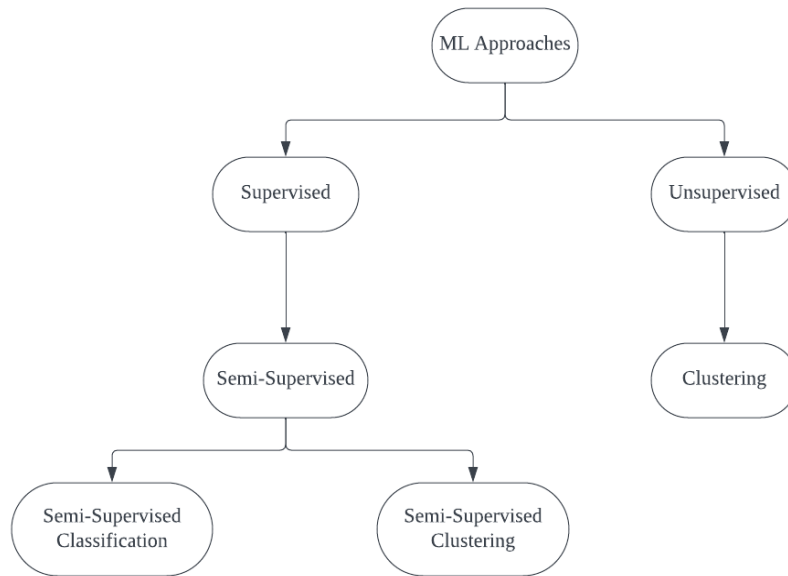


Fig 2.3 - Machine Learning Categories

Supervised learning:-

Supervised learning refers to a type of machine learning where a model is trained with labeled data, meaning the input data is paired with corresponding output labels. The model learns from this labeled data to make

Supervised learning utilizes predefined patterns to construct a database that aids in the classification of new patterns. The primary purpose of this learning process is to develop a correlation between the input attributes and an output, which is often known as a class. A model is created as a result of this learning process [11].

By analyzing input patterns. The model can accurately categorize cases that are not perceptible. Typically, it may be expressed as a function $f(x)$ where the input comprises of patterns and the output is a class y . The training set (TS) comprises pre-classified patterns containing both input and output pairs, whereas the test set comprises unseen patterns containing just input patterns. The dataset DS is composed of pairs, with each pair containing a feature vector X and its accompanying label y . The dataset has p classifications and n patterns or observations. Technique 1 is a broad form of supervised learning. The set of algorithms includes Decision

Trees, bagging, Boosting, Random Forest, k-NN, Logistic Regression, Neural Networks, Support Vector Machines, Naive Base, and Bayesian Networks[11].

Algorithm 1: Generic Supervised Learning

```

Input: N training examples with labels
dataset : {  $X \rightarrow Y$  }
{ <  $x_1, y_1$  >, <  $x_2, y_2$  > ..... <  $x_n, y_p$  > }

 $k \leftarrow 10$ ;
//cross validation Output: M - training model based
on probabilistic approach

 $i \leftarrow 0$ ;

for each  $i$  in  $k$  do
     $dataset\_samples \leftarrow dataset / k$ 
     $training\_dataset \leftarrow dataset\_samples[i]$ 
     $M_i \leftarrow Classifier(training\_dataset)$ 
     $M \leftarrow M_i$ 

return  $M$ 

```

Fig 2.4 - Generic Supervised Learning Algorithm

Unsupervised Clustering:-

Unsupervised clustering is the act of categorizing data points into groups based on their similarities, without any prior knowledge or direction.

Unsupervised learning investigates how systems may produce specific input patterns that effectively represent the statistical structure of the entire set of input patterns. Unsupervised learning relies on pre-existing biases to determine which elements of the input's structure should be included into the output. On the other hand, supervised learning and reinforcement learning involve the inclusion of specific desired outcomes or evaluations of the environment that are linked to each input.

Unsupervised learning approaches solely rely on the observed input patterns (x_i) as their primary resource[11].

Typically, it is assumed that the given data represents specific examples from a hidden probability distribution $PI[x]$, coupled with some explicit or implicit details about what is considered important. Below is the algorithm for Unsupervised Learning, specifically Algorithm 2. Density estimation algorithms are used to explicitly design statistical models, such as Bayesian networks, that explain how the input may have been produced by underlying causes. Feature extraction techniques are used to directly generate statistical regularities or anomalies from the inputs. In addition, unsupervised learning incorporates many extra strategies to identify and elucidate relevant characteristics of the data. Preprocessing approaches in data mining serve as the foundation for several unsupervised learning algorithms[11]. Unsupervised learning explores how systems might learn to represent individual input patterns in a manner that captures the statistical structure of the complete set of input patterns, without relying on pre-assigned clusters from training examples. Unsupervised learning depends on pre-existing biases regarding the inclusion of specific aspects of the input's structure in the output. In contrast, supervised learning and reinforcement learning incorporate explicit goal outputs or environmental evaluations associated with each input. Unsupervised learning approaches just need the observed input patterns (x_i), which are often assumed to be independent samples from an unknown probability distribution ($PI[x]$). The International Journal of Engineering & Technology 83 requires a certain level of prior knowledge, whether it is explicitly stated or implied, about relevant topics. Presented here is the algorithm for Unsupervised Learning, specifically Algorithm 2. Return M and c as long as clusters are allocated to all training samples.

Algorithm 2: Generic Unsupervised Learning

```
Input: N training examples without labels
dataset: {X→?}
{< x1, y? >, < x2, y? >, ..... , < xn-1, y? >, < xn, y? >}
k←5; // # of clusters
cv←10; //cross validation
Output: M c - returns model with k # of clusters and center of each cluster
i←0;
do
  for each i in cv do
    //iterates cv times
    dataset_samples ← dataset / k
    trainingi ← dataset - dataset_samples [i]
    Mi ← Cluster (trainingi)
    c ← ci M ← Mi
```

Fig 2.5 - Generic Unsupervised Learning Algorithm

Semi-Supervised learning:-

Semi-supervised learning is a machine learning approach that involves training a model using a combination of labeled and unlabeled data.

Semi-supervised learning (SSL) is a specific methodology within the field of machine learning (ML). It is positioned between supervised and unsupervised learning, as seen in Figure 2.3, with the dataset containing some labeled data. The primary goal of SSL[11] is to overcome the limitations of both supervised and unsupervised learning. Utilizing supervised learning to categorize test data necessitates a big commitment of time and financial resources, since it relies on a considerable quantity of training data. Unsupervised learning categorizes data by similarity in data points using either the maximum likelihood technique or clustering, without relying on any labeled data. An inherent limitation of this method is its lack of capacity to reliably categorize unknown data into clusters. SSL has been suggested by the scientific community as a resolution to these problems. It possesses the capacity to identify new (or test) material after gaining knowledge from a limited amount of training data. SSL treats the other patterns as test data and uses a small number of labeled patterns as training data to build a model. Here is the standardized version of Algorithm 3 specifically tailored for semi-

supervised learning. Semi-supervised learning may be categorized into two main types. The next section delves into the principles of semi-supervised classification and semi-supervised clustering.

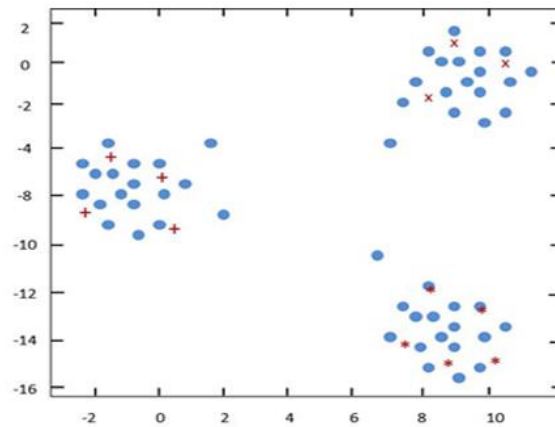


Fig 2.6 - How semi-supervised clustering works

Semi-Supervised Classification:-

Semi-supervised classification (SSC) employs additional training data to categorize the test data, similar to the supervised technique. In order to categorize the extensive amount of test data in SSC, a smaller amount of training data is utilized. Utilizing this semi-supervised classification method allows for a reduction in the amount of training data required. Currently, the research community has an ample supply of unlabeled data patterns, but there is a lack of labeled data. Because the process of designing training data is both time-consuming and expensive[11].

Algorithm 3: Generic Semi-Supervised Learning

Input: N training examples with partial labels Input.
dataset : $\{X \rightarrow Y\}$
 $\{ \langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle, \dots, \langle x_n, y_p \rangle \}$
 $cv \leftarrow 10$; // cross validation
Output: M - training model based on probabilistic approach.
 $i \leftarrow 0$;
do
 for each i in cv **do**
 //iterates cv times
 $dataset_samples \leftarrow dataset / k$
 $training_i \leftarrow dataset - dataset_samples[i]$
 $M_i \leftarrow \text{Classifier}(training_i, k)$
 $M \leftarrow M_i$
while till assign labels to all training examples **return** M

Fig 2.7 - Generic Semi-Supervised Learning Algorithm

A suggestion was presented in [11] for a selective incremental transductive nearest neighbor classifier (SI-TNNC) as an approximation technique for recognizing the test patterns. The researchers analyzed their findings by analyzing five distinct datasets and implementing five different methods. In three out of the five cases, the SI-TNNC algorithm exhibited higher accuracy than widely used algorithms like ID3 and 3NN, among others. A technique was introduced in [11] to classify partially labeled data with the objective of enhancing classification accuracy. The author's primary contention was that classification should exclusively depend on the sub-main fold and disregard the ambient space. The suggested methodology use the adjacency graph to approximate labels. The Laplace-Beltrami operator is used in the primary framework to generate a Hilbert space on the submanifold. The framework only requires unannotated examples to fulfill this objective. A approach utilizing semi-supervised learning has been presented in reference [11] for the purpose of real-time traffic classification.

Semi-supervised clustering is a clustering method that combines labeled and unlabeled data to group similar occurrences together.

Semi-supervised clustering is a unique clustering approach that differs from typical approaches by including both labeled and unlabeled data, along with additional side information in the form of paired constraints (must-link and cannot-link). These constraints enable the effective categorization of data patterns. The Semi-supervised Single Link (SSL) clustering algorithm addresses the issue of grouping clusters that have arbitrary geometries. SSL solves the problem of the noisy bridge by using a pre-determined distance matrix with minimal constraints to control the distances between clusters. The researchers validated the precision of their findings by conducting tests on both synthetic and real datasets.

The self-training approach is a widely employed technique in semi-supervised clustering, specifically in the domain of SSL. This method involves classifying a small collection of labeled training data, then classifying unlabeled data, and incorporating these expected patterns into the training set. This iterative process continues until the test set is all used up. Some algorithms utilize a "unlearning" strategy to handle repeating processes by discarding unlabeled points, as long as the predicted patterns do not exceed a certain threshold. Self-training has been utilized in several activities, such as natural language processing (NLP)[11].

The Support Vector Machine (SVM) is a conventional classifier that exclusively depends on labeled data. Transductive Support Vector Machines (TSVMs) are an enhanced version of SVMs that integrate both labeled and unlabeled input data. TSVMs aim to categorize unlabeled data patterns that are linearly separable by maximizing the margin between the separating hyperplane and both the labeled and unlabeled data points. TSVMs have gained significant popularity and are being widely used in several domains such as image retrieval, bioinformatics, and named entity recognition.

In [11], a probabilistic framework was introduced for the task of semi-supervised clustering. This technique proposes the reduction of the objective function derived from the posterior energy of Hidden Markov Random Fields (HMRF) in order to address the task of semi-supervised clustering. The authors demonstrated the effectiveness of their system on several text datasets, highlighting the advantages of semi-supervised learning.

2.2.3 Anomaly Detection techniques:-

All anomaly-based network intrusion detection systems (A-NIDS) approaches have a common underlying structure. As described in reference [12], these techniques typically consist of the following fundamental modules or phases (see to Figure 2.8). The processes encompass training, detection, and parameterization. During the parameterization phase, it is necessary to collect raw data, such as packet data from a network, from a monitored environment. This data must accurately reflect the system that is being represented. During the training phase, the system is modeled by either human or automatic methods. In a client-server configuration, the server assumes the role of a host and actively listens for connections. Once the client and server establish a connection, the server generates a socket which is then used to execute a handler object on a separate thread. The handlers within a collection object are organized[12]. The behaviors depicted in the model differ based on the utilized methodology. The detection step entails comparing the system developed during the training stage with the specifically parameterized data chunk that has been chosen. Anomalous data instances are identified by selecting threshold criteria.

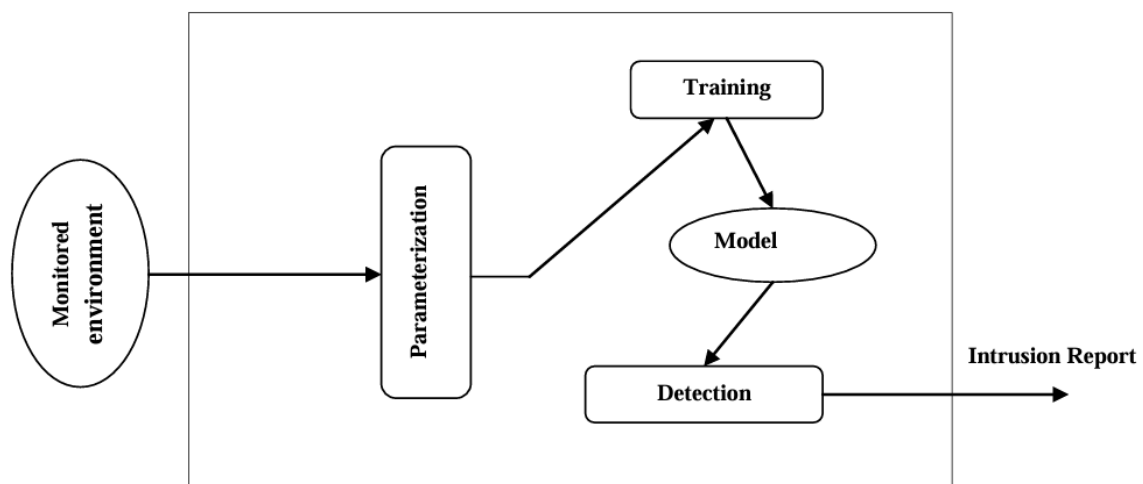


Fig 2.8 - Anomaly Detection Process

Machine learning can automatically generate the required model using available training data[12]. The justification for this strategy is based on the ease of obtaining the necessary training data, which is more easily available compared to the work needed for manually defining the model. Given the rising complexity and variety of assaults, the use of machine learning techniques is crucial in developing and managing anomaly detection systems (ADS) that require little human involvement. This strategy is a pragmatic method for attaining the next iteration of intrusion detection systems.

When utilizing machine learning methods for intrusion detection, the model is constructed automatically by employing a training dataset. This dataset consists of data objects characterized by a collection of qualities (features) and their corresponding labels. The attributes can manifest in several ways, including categorical or continuous, and the characteristics of these attributes dictate the appropriateness of anomaly detection algorithms. Data instances are usually labeled with binary values that indicate whether the behavior is normal or aberrant. Nevertheless, several researchers have chosen to employ different forms of attacks such as Denial of Service (DoS), User to Root (U2R), Remote to Local (R2L), and Probe, instead of a generic anomalous designation. This learning approach is capable of delivering more comprehensive information about the discovered abnormalities.

Methods:

1. The K-Nearest Neighbor algorithm.
2. A Bayesian Network (BN)
3. A Supervised Neural Network (SNN)
4. A support vector machine (SVM)
5. Clustering techniques
6. The One-Class Support Vector Machine (One-Class SVM)
7. The Fuzzy C-Means algorithm
8. Unsupervised niche clustering refers to the process of grouping similar items or entities together based on their specific characteristics or attributes without the need for any prior knowledge or guidance.

9. An unsupervised neural network
10. K-Means
11. The Expectation Maximization Meta Algorithm (EM) is a computational method used to estimate parameters in statistical models when there are missing or incomplete data.

K-Nearest Neighbor (KNN) is a machine learning algorithm that is used for classification and regression tasks. It is a non-parametric method, meaning it does not make any assumptions about the

The k-nearest neighbor technique is a semi-supervised learning method that utilizes training data and a specified number of k to determine the k closest data points by calculating their distances. When the k closest data points are assigned to distinct classes, the method predicts that the unknown data belongs to the majority class. For example, let's examine the use of the k-nearest neighbor method using a Euclidean distance metric. This is shown in Figure 2.9, which displays the categorization of iris data[13].

The point (5, 1.45) marked as "X" in figure 2.9 needs to be categorized. The k-nearest neighbor technique is applied with a value of k equal to 8, and the Euclidean distance is used for computation. The outcome is shown as a dotted line with a radius of 13. Within this range, there are two potential categories: the virginica class, which has two occurrences, and the versicolor class, which has six examples. The method categorizes point "X" as a member of the versicolor class due to it being the predominant class within the designated radius.

Distance measures can be employed to ascertain the distance between a new data point and an existing training dataset. In this work, we utilize eleven distance measurements, each of which is accompanied by its respective explanations[13].

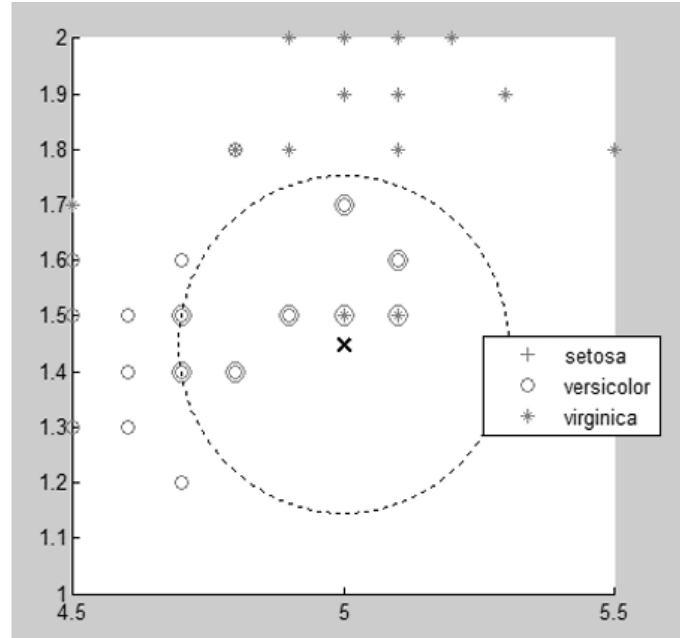


Fig 2.9 - Graph of K-Nearest Neighbor

The distances between the row vectors x_s and y_t , where x_s represents the $m \times n$ data matrix X handled as m (1-by- n) row vectors x_1, x_2, \dots, x_m , and y_t represents the $n \times m$ data matrix Y treated as n (1-by- n) row vectors y_1, y_2, \dots, y_n , are defined as follows:

1. Euclidean distance is a mathematical measure that calculates the straight-line distance between two points in a Euclidean space.

The Euclidean distance is a metric used to calculate the distance between two points and is defined by Equation 2.3[13].

$$d_{st}^2 = (x_s - y_t)(x_s - y_t)'$$

Eq 2.3

The Euclidean distance is a specific instance of the Minkowski metric, with the value of p equal to 2.

2. The Standardised Euclidean distance:

The issue of finding the distance is optimized using the standardized Euclidean distance, which is specified by Equation 2.4[13].

$$d_{st}^2 = (x_s - y_t)V^{-1}(x_s - y_t)'$$

Eq 2.4

In this case, V is a square matrix of dimensions n-by-n, where each element on the diagonal is the square of the jth element in the vector S. The vector S contains the reciprocals of the weights.

The number is 3. Mahalanobis distance:

The Mahalanobis distance is a statistical measure that quantifies the distance between a point and a distribution of data. It is defined by Equation 2.5.

$$d_{st}^2 = (x_s - y_t)C^{-1}(x_s - y_t)'$$

Eq 2.5

C is the covariance matrix.

4. City Block Distance:

The city block distance between two points is defined as the sum of the absolute differences of Cartesian coordinates, expressed by Equation 2.6:

$$d_{st} = \sum_{j=1}^n |x_{sj} - y_{tj}|$$

Eq 2.6

The city block distance is a specific case of the Minkowski metric, with a value of $p = 1$.

The number 5. The Minkowski distance is a mathematical measure that quantifies the dissimilarity between two points in a multi-dimensional space.

The Minkowski distance is a mathematical technique used to measure the distance between two points in Euclidean space. It is defined by Equation 2.7.

$$d_{st} = \sqrt[p]{\sum_{j=1}^n |x_{sj} - y_{tj}|^p}$$

Eq 2.7

When p is equal to 1, the Minkowski metric produces the City Block distance. When p is equal to 2, it gives the Euclidean distance. And when p is equal to ∞ , it gives the Chebychev distance.

The number 6. Chebyshev distance:

The Chebychev distance is a metric used to calculate the distance between two vectors or points using standard coordinates, as specified by Equation 2.8.

$$d_{st} = \max_j \{|x_{sj} - y_{tj}|\}$$

Eq 2.8

The Chebychev distance is a specific instance of the Minkowski metric, in which the value of p is equal to infinity.

The number 7. Cosine distance:

The Cosine distance is calculated by subtracting the cosine of the angle between two locations from one, as given by Equation 2.9.

$$d_{st} = \left(1 - \frac{x_s y'_t}{\sqrt{(x_s x'_s)(y_t y'_t)}} \right)$$

Eq 2.9

8. Correlation Distance:

The number 8. Correlation distance:

The Correlation distance, which quantifies the statistical relationship between two vectors, is mathematically described by Equation 2.10.

$$d_{st} = \left(1 - \frac{(x_s - \bar{x}_s)(y_t - \bar{y}_t)'}{\sqrt{(x_s - \bar{x}_s)(x_s - \bar{x}_s)'} \sqrt{(y_t - \bar{y}_t)(y_t - \bar{y}_t)'}} \right)$$

where

$$\bar{x}_s = \frac{1}{n} \sum_j x_{sj}$$

$$\bar{y}_t = \frac{1}{n} \sum_j y_{tj}$$

Eq 2.10

The number 9. Hamming Distance refers to the number of positions at which two strings of equal length differ.

Equation 2.11 defines the Hamming distance, which measures the fraction of coordinates that vary.

$$d_{st} = \left(\frac{\#(x_{sj} \neq y_{tj})}{n} \right)$$

Eq 2.11

10. Jaccard Distance:

The Jaccard distance is calculated using equation 2.12, which measures the difference between one minus the Jaccard coefficient. The Jaccard coefficient is defined as the proportion of non-zero coordinates that differ, as stated in equation 2.12.

$$d_{st} = \left(\frac{\#(x_{sj} \neq y_{tj})}{n} \right)$$

Eq 2.12

11. Spearman Distance:

The Spearman distance is computed and is derived from one minus the sample Spearman's ranked correlation between observations, as defined by Equation 2.13.

$$d_{st} = 1 - \frac{(r_s - \bar{r}_s)(r_t - \bar{r}_t)'}{\sqrt{(r_s - \bar{r}_s)(r_s - \bar{r}_s)'} \sqrt{(r_t - \bar{r}_t)(r_t - \bar{r}_t)'}}$$

Where

r_{sj} is the rank of x_{sj} taken over $x_{1j}, x_{2j}, \dots, x_{mj}$.

r_{tj} is the rank of y_{tj} taken over $y_{1j}, y_{2j}, \dots, y_{mj}$.

r_s and r_t are the coordinate-wise rank vectors of x_s and y_t ,

i.e., $r_s = (r_{s1}, r_{s2}, \dots, r_{sn})$ and $r_t = (r_{t1}, r_{t2}, \dots, r_{tm})$.

$$\bar{r}_s = \frac{1}{n} \sum_j r_{sj} = \frac{(n+1)}{2}$$

$$\bar{r}_t = \frac{1}{n} \sum_j r_{tj} = \frac{(n+1)}{2}$$

Eq 2.13

All the equations above are referenced from [13].

Chapter 3: Theoretical Framework

3.1 Mitigating Algorithmic Bias

The paper classifies bias mitigation methods into three categories based on their employment within the AI process: pre-processing, in-processing, and post-processing.

3.1.1 Pre-process stage:

Data preparation involves the utilization of strategies to alter APK data and machine learning methods to prepare the data in a suitable manner for entry into the learning model. The goal is to eradicate any biases that could exist within the training data itself. This domain includes techniques such as relabeling, data production, and fair representation.

1. Relabeling: - This technique entails altering the data labels in order to mitigate bias. When manipulating APK data, it can be essential to modify the labels of some application categories or developers to provide a more equitable portrayal.

2. Data creation: - Data generation methods offer artificial data to equalize class distributions. Regarding APK data, this may entail creating more examples for app categories that have insufficient representation, so creating a dataset that is fairer and less biased.

3. Fair Representation: Fair representation methods strive to provide a data representation that maintains the original qualities while eliminating factors that contribute to prejudice. APK data refers to the acquisition of information about a representation that eliminates biases associated with protected attributes, such as the author or category of the program.

3.1.2 Initial processing:

In-processing techniques are applied directly to the machine learning model training process. These strategies incorporate limits or adjustments into the learning process to reduce bias.

1. **Constraint Optimization:** - Algorithms are adjusted to incorporate constraints that ensure fairness throughout the learning process. Regarding APK data, it may need imposing restrictions on the model to guarantee that specific app categories are not given preferential treatment or handled unfairly.

2. **Regularization:** Regularization methods impose a penalty on the model for gaining characteristics that might potentially lead to bias. Regularization techniques possess the capacity to prevent the model from excessively depending on certain properties associated with app categories or developers when processing APK data.

3. **Calibration:** Calibration processes are employed to accurately fine-tune the model's outputs in order to accord with fairness objectives. APK data calibration refers to the process of adjusting the model's predictions to provide fair treatment across various app categories.

3.1.3 Influence on APK Data and Machine Learning Algorithms:

1. **Bias Reduction:** - The implementation of diverse bias mitigation approaches, whether in pre-processing, in-processing, or post-processing, aids in diminishing biases in the APK data. It is of utmost importance to guarantee equitable treatment of all app categories and developers.

2. **Fairness Constraints:** - Methods such as constraint optimization and regularization incorporate fairness constraints into the learning process. The machine learning technique is limited by these limits to prevent the gathering of biased patterns in the APK data.

3. **Performance trade-offs:** While it is essential to reduce bias, it is necessary to acknowledge that there may be compromises in terms of performance. Particular methodologies might influence the overall anticipated precision of the model, and researchers must meticulously deliberate the trade-off between impartiality and usefulness.

4. **The adaptability of pre-processing techniques:** Pre-processing techniques like as relabeling, data augmentation, and equitable representation provide versatility and may be readily applied to many jobs without substantial limitations. This is useful for organizing various APK data.

The work recognizes the difficulties that arise when attempting to optimize two aspects simultaneously in pre-processing procedures, known as dual optimization issues. Implementing fair representation is a methodological difficulty as it requires balancing the preservation of original traits with the removal of biased aspects.

To ensure equity and minimize prejudice in machine learning algorithms, it is essential to employ bias mitigation measures inside the APK data context. The classified techniques offer a systematic framework for researchers to tackle bias at several phases of the AI process, guaranteeing that the model's choices are fair across multiple protected aspects in the APK data. The training procedure is conducted with a value of 3.2.

3.2 The Training Process

The training process is a crucial stage when the machine learning algorithm adjusts its parameters using the training data, resulting in the creation of a model that can make predictions on new and unfamiliar data.

3.2.1 Explanation of the Training Process

1. Objective Function: - The training process starts by formulating an objective function that encompasses the desired outcomes of the model. The study employed an objective function grounded in information theory, especially tailored to attain an equitable depiction of the APK data. It achieves a harmonious equilibrium by maintaining the original data while eliminating any prejudiced characteristics associated with safeguarded traits.

2. Variational Inference: - Variational inference is used to estimate the objective function. This entails establishing the minimum value (LB) for the first term, which pertains to the preservation of the original information, and the maximum value (UB) for the second term, which is linked to the elimination of biased qualities. Variational inference enables the use of practical approximations in cases where direct optimization may not be practical.

3. Dual Optimization: The training strategy tackles the problem of dual optimization caused by the competing objectives. The strategy achieves a balance between utility and justice by increasing the lowest value and limiting the highest value. In order to effectively reduce bias while preserving model performance, it is imperative to fulfill both optimization objectives.

The Information Bottleneck Theory is a theoretical framework that highlights the need of limiting the amount of information in order to extract the most relevant and important information. The training process is based on the principles of information bottleneck theory, leading to the creation of a distinct latent space. Data segregation is crucial for maintaining equity by eliminating any data related to protected attributes, while preserving the inherent qualities of the data.

The process of encoder-decoder learning entails the employment of an encoder-decoder architecture to provide guidance. The encoder acquires the ability to convert the APK data into a significant depiction, while the decoder guarantees that this depiction is adequate for reconstructing the initial data. The mitigation function and discriminator play a role in organizing the latent space throughout the learning process.

During training, the Mitigation Function and Discriminator have a reciprocal influence on each other as they interact. The mitigation function alters the latent space to eliminate distinctive characteristics, while the discriminator identifies and eliminates protected properties. This contact follows an iterative process, meaning that it occurs in a series of phases or cycles. The purpose of this approach is to methodically enhance the equity of the results produced by the model.

The primary goal of the training process is to attain a harmonious equilibrium between justice and usefulness, by effectively balancing fairness and utility. Although the main objective is to reduce bias, it is equally important for the model to maintain utility by ensuring that the obtained representations are meaningful and successfully enhance the model's prediction capabilities.

3.2.2 Effect on Model Performance:

1. Impartial Representations: - The training approach is customized to guarantee that the model acquires unbiased representations of the APK data. This entails acquiring information about a hidden domain that captures significant data while simultaneously reducing biases associated with protected attributes.

2. Diminished Bias: - By engaging in an iterative learning process, the model progressively reduces prejudice towards particular protected attributes that have been detected in the APK data. The decrease is achieved by the synergistic optimization and interaction between the mitigation function and discriminator.

3. Adaptable bias Reduction: The suggested training paradigm, based on information theory, offers a flexible method for minimizing bias. It is not restricted to certain protected characteristics and may be utilized in a range of operations, suiting the different nature of APK data.

4. Methodological Consistency: The training strategy effectively addresses any inconsistencies in pre-processing operations by splitting the hidden space. It offers a reliable paradigm for achieving equitable representations by effectively addressing the issues of dual optimization.

Essentially, the training procedure is a crucial step in the development of a machine learning algorithm for APK data, aiming to ensure fairness and reduce bias. The training approach utilizes principles from information theory, variational inference, and an encoder-decoder architecture. The model includes a mitigation function and discriminator to aid in acquiring representations that achieve a harmonious equilibrium between usefulness and impartiality. Consequently, this finally results in enhanced model performance.

3.3 Assessment:

1. Fairness metrics: In the assessment step, the trained machine learning model is assessed to measure its degree of fairness. Studies suggest that fairness is assessed using measures such as

Parity Difference (PD), Equalized Opportunity (OPP), and Equalized Odds (ODD). These measures assess the extent to which the model ensures equitable treatment of particular groups or protected attributes.

2. Performance evaluation metrics: Additionally, conventional performance benchmarks are assessed with fairness. Metrics that may be assessed include accuracy, precision, recall, and F1-score. Assessing the model's capacity to identify dangerous risks in APK files and guaranteeing impartiality is of paramount significance.

3. Utility Evaluation: - The effectiveness of the model is evaluated to ensure that fairness concerns have not significantly affected its ability to produce accurate predictions. This entails attaining a balanced state of agreement between reducing bias and ensuring the model's effectiveness in accurately predicting outcomes for new data.

4. Comprehensive Evaluation: A comprehensive assessment takes into account both fairness and utility measurements. The evaluation of the model's performance takes into account a comprehensive perspective that incorporates both the goals of lowering bias and preserving prediction accuracy.

5. The study emphasizes the need of utilizing benchmark datasets, such as Census and COMPAS, for the purpose of evaluating performance. These datasets are widely accepted as standard standards for evaluating the fairness and effectiveness of the model in various situations.

3.4 Procedures:

The previous section of chapter 2 offered a concise explanation of the K-nearest neighbor approach. K-nearest neighbor (k-NN) is a conventional and conservative nonparametric technique used for classifying data. Subsequently, the unmarked point is categorized based on its K-nearest neighbors by computing the distances between various points on the input vectors. The parameter (k) plays a vital role in constructing the k-NN classifier, and varying values of (k) can lead to distinct results. When k is large, it takes a significant amount of time to identify

the surrounding data points used for prediction, resulting in a drop in the accuracy of the forecast[12]

A Bayesian network (BN) is a statistical model that represents the probabilistic relationships between variables of interest, as stated in reference [12]. This strategy is commonly employed in conjunction with statistical methods to detect intrusions. An advantage of this is its ability to absorb input and existing knowledge, as well as to encode the relationships between variables and predict events.

According to reference [12], decision trees are very efficient and often used methods for categorization and forecasting. The fundamental components of a decision tree consist of nodes, edges, and terminal nodes. The feature property assigned to each node is the most informative attribute that has not been considered throughout the traversal from the root. Each leaf in the tree is assigned a certain class or category, and each branch that comes from a node is assigned the feature value linked to that node. When a decision tree reaches a leaf node, it can classify a data point by traversing the entire tree, beginning at the root. The classification of the data point is determined by the leaf node.

K-Means is a traditional clustering technique. The dataset is partitioned into k clusters, ensuring that the data inside each cluster is similar. Nevertheless, the data in the 12 clusters exhibit little similarities. The K-means method starts by randomly picking K data points as the first cluster centroids. Subsequently, the residual data is appended to the cluster exhibiting the greatest resemblance, as determined by its proximity to the cluster centroid. Subsequently, the cluster centroids for each cluster are recalibrated. Continue iterating this procedure until there are no more updates to any of the cluster centers.

Fuzzy C-Means is a clustering algorithm that enables a single data point to be assigned to many groups simultaneously. It is utilized in circumstances, such as pattern recognition, when it is difficult or impractical to classify tangible data. The C-means approach is essentially the same as the K-means algorithm, except that it incorporates a fuzzy function to identify the membership of each point and the impact of each point on the movement of a cluster centroid, based on its fuzzy membership to that cluster[12].

3.4.1 Anomaly Detection Algorithm:-

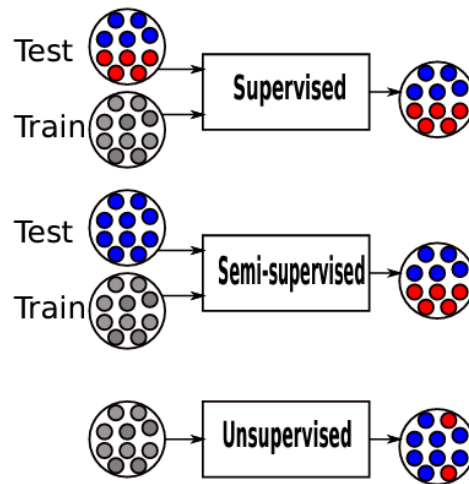


Fig 3.1 - Demonstration of Supervised, Semi-Supervised and Unsupervised ML

An illustrative example showcasing the training and test sets for each learning method. The colors gray, black, and white represent typical, exceptional, and unclassified data, respectively. Prior knowledge is necessary for unsupervised learning to establish a training dataset. Inexperienced algorithms. Obtaining a training dataset that accurately represents anomalies in specific applications is extremely challenging due to the constantly changing attributes of the abnormal records. Anomaly detection systems can utilize the assumptions made about the unusual records to categorize them. The dominant and proficient group in an unsupervised situation consists of algorithms that depend on nearest-neighbor ideas. The algorithms' effectiveness lies on their inherent capacity to function autonomously and their intuitive standards for detecting anomalies. The system is limited by its quadratic temporal complexity and its unpredictable behavior while processing high-dimensional input. The user's material includes a specific reference or citation indicated by the numeral [14].

The training set, denoted as $S = \{x_1, x_2, \dots, x_n\}$, is a sample that is randomly and uniformly selected from an unknown multivariate nominal density $f_0(\cdot)$ with d dimensions. The formula $f(\eta) = (1 - \pi)f_0(\eta) + \pi f_1(\eta)$ represents the merging of the standard density $f_0(\cdot)$ and a known abnormal density $f_1(\cdot)$ from which a test sample η is extracted. To simplify, let's assume that an anomaly is evenly distributed and has an equal likelihood of occurring

wherever. Anomaly detection may be described as an issue of testing composite hypotheses. Hypothesis H_1 suggests the presence of an abnormality, where the parameter π is greater than 0. In contrast, hypothesis H_0 states that there is no anomaly, and the parameter π is equal to 0 for nominal data. The goal is to improve the capacity to detect signals while ensuring that the frequency of false alarms remains below a specific threshold, as shown by the inequality $PF \leq P(\text{decision} = H_1 | H_0) \leq \alpha$ [15]. The most efficient diagnostic test for identifying the previously described issue is:

$$D(\eta) = \begin{cases} H_1 & p(\eta) \leq \alpha \\ H_0 & \text{otherwise} \end{cases}$$

where $p(\cdot)$ is the p -value function defined as:

$$p(\eta) = \mathcal{P}_0 \left(x : \frac{f_1(x)}{f_0(x)} \geq \frac{f_1(\eta)}{f_0(\eta)} \right) = \int_{\{x: f_0(x) \leq f_0(\eta)\}} f_0(x) dx$$

Eq 3.1

One-class classification methods are employed to elucidate the expected (or "normal") behavior in a traditional approach to anomaly detection. This approach involves creating a model of a standard or usual situation by using several instances, such as determining the exact spatial location of training patterns within a feature space. When a novel test pattern differs from the standard class, we classify it as anomalous [15].

Here, we present a brief summary of two traditional methods for classifying a single category. We are provided with a sample pattern $(x_1, \dots, x_l) \in X \subset \mathbb{R}^n$ that exhibits the i.i.d. (independent and identically distributed) characteristic. The primary goal of these algorithms is to distinguish between sample patterns that are deemed "typical" and those that are, in some manner, "atypical." This content is sourced from [15].

3.4.1.1 Support Vector Machine (SVM)

SVM classification in binary classification categorizes data samples into negative and positive classes by choosing an optimal separating hyperplane as the decision boundary from a range of potential hyperplanes in the input space or feature space. The decision boundary is established by choosing the greatest distance between the linear separators and the nearest positive and negative samples, either in the input space or feature space. Every decision boundary is linked to a set of interconnected hyperplanes. Understanding a decision boundary with a wide safety margin can help reduce mistakes in generalization and improve performance on test cases. The number 18 is bracketed by square brackets.

A linear Support Vector Machine (SVM) is a binary classifier that uses a decision boundary in the input space to categorize training data points. Hard margin classifiers are utilized to tackle linear SVM problems where the data points cannot be separated by a linear boundary. Conversely, soft margin classifiers are employed to address linear SVM difficulties when the data points cannot be separated by a linear boundary. The source of the passage is referenced as "[18]".

Nonlinear SVM employs kernel methods to map datasets with a decision boundary that changes online in the input space to a higher-dimensional space (feature space) in order to get a maximum margin in the hyperplane. The following sections provide a brief explanation of the mathematical principles that provide the basis for the creation of binary nonlinear Support Vector Machines (SVMs).

Kernel techniques in Support Vector Machines (SVM) are a technique that enhances generalization during training by utilizing a non-linear decision boundary in the input space. The method, as outlined in reference [18], transforms each vector x (datapoint) from the input space, which has n dimensions, into a new space named $\Phi(x)$ with l dimensions. Subsequently, it builds a linear classifier or hyperplane to partition the data points in the feature space.

$$f(x) = w \cdot \Phi(x) + b.$$

Eq 3.2

Represented below in figure 3.2 is the inner product of two vectors:

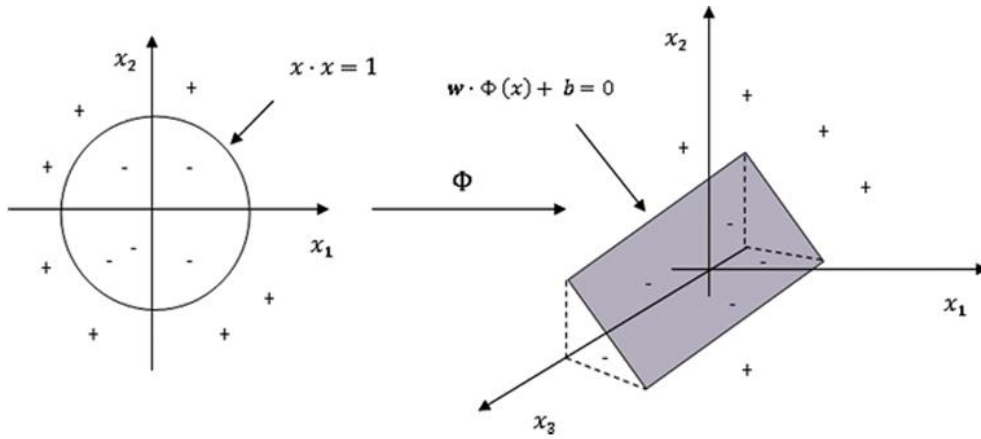


Fig 3.2 - inner product of 2 vectors in one-class SVM

The following Hilbert-Schmidt theorem for every input dataset $x_i \in X$ (input space) is satisfied if an asymmetric function $k(x_i, x_j)$.

$$\sum_{i,j=1}^N h_i h_j k(x_i, x_j) \geq 0, \forall h_i \in \mathbb{R}_+, N \in \mathbb{N},$$

Eq 3.3

then the following condition is satisfied by the mapping function $\Phi(x)$, which transforms any input data x_i in input space into feature space :

$$k(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j).$$

Eq 3.4

If Equation 3.4 is satisfied then

$$\sum_{i,j=1}^N h_i h_j k(x_i, x_j) = \left(\sum_{i=1}^N h_i \Phi(x_i) \right) \cdot \left(\sum_{j=1}^N h_j \Phi(x_j) \right) \geq 0.$$

Eq 3.5

The function $k(x_i, x_j)$ is widely recognized as the positive definite kernel or Mercer kernel, and the requirement mentioned in Equation 3.3 or Equation 3.5 is frequently referred to as Mercer's condition.

The kernel approach is a technique that employs the kernel trick, which substitutes the evaluation of the transformation $\Phi(x_i)$ with the use of $k(x_i, x_j)$ for training and classification. Support Vector Machines (SVM) have the benefit of being able to reduce overfitting and improve generalization by properly choosing suitable kernels. The selection of kernels for a certain application has greater importance in this context. The kernels often used in SVM are given [18] are as follows:

1-Linear Kernel

$$k(x_i, x_j) = x_i \cdot x_j.$$

Eq 3.6

2-Polynomial Kernels

$$k(x_i, x_j) = (\gamma(x_i \cdot x_j) + r)^d, r \geq 0, \gamma > 0.$$

Eq 3.7

3- Radio Basis Functions(rbf).(Used in chapter 5)

$$k(x_i, x_j) = \exp(-\|x_i - x_j\|^2 / 2\sigma^2) \text{ where } \sigma > 0.$$

Eq 3.8

Chapter 4: Research Methodology

The increasing ubiquity of mobile applications in recent years has significantly revolutionized the manner in which individuals engage with digital technology. The Android platform, being a very popular mobile operating system, has experienced a significant increase in the number of apps available inside its ecosystem. Nevertheless, this sudden increase in popularity has also drawn the interest of malevolent individuals aiming to exploit weaknesses for different detrimental intentions. Therefore, it is becoming more crucial to develop strong and effective techniques for identifying harmful threats in Android Application Package (APK) files.

Within chapter 3, many techniques were mentioned. The One-Class Support Vector Machine (One-Class SVM) is a specialized machine learning technique designed specifically for detecting hazardous threats in APK files, among other accessible approaches. The One-Class Support Vector Machine (SVM) is highly advantageous for applications that need anomaly detection, whereby the primary objective is to identify instances that deviate significantly from the standard pattern. Despite the fact that most of the training data comprises of innocuous samples, the One-Class SVM demonstrates remarkable precision in distinguishing between potentially hazardous programs and genuine, non-malicious ones in the context of APK file analysis.

Given the ongoing development of threats, it is crucial to create security solutions that are both sophisticated and adaptable. The aim of this approach is to improve mobile security by leveraging the capabilities of One-Class Support Vector Machines (SVM) to detect abnormalities in Android APK files. It provides a proactive protection mechanism to combat new threats in the dynamic environment of mobile apps.

One-Class Support Vector Machines (OCSVM) are unsupervised machine learning methods used to detect anomalies. The main goal of OCSVM is to identify observations that exhibit substantial deviations from the majority of data points in a given dataset. Anomalies, also known as outliers, are data points that deviate from the general patterns detected in the rest of the data, making them uncommon or extraordinary.

Unlike the conventional Support Vector Machine (SVM) approach, which is commonly used for binary classification tasks, the One-Class Support Vector Machine (OCSVM) is specifically designed for situations when only one class of data (known as the normal class) is present. Anomaly detection involves the task of defining a border that includes the majority class while eliminating probable abnormalities.

The core principle of OCSVM is to transform the input data into a feature space with higher dimensions using a kernel function, similar to traditional SVMs. Afterwards, the algorithm aims to determine a hyperplane that includes as many regular data points as feasible, while also retaining the largest possible distance between this hyperplane and the data points.

4.1 One-Class Support Vector Machine (SVM):

In the first formulation of One-Class SVM, patterns that are located around the origin of coordinates in a feature space are regarded as anomalous. The process of discerning patterns is achieved by introducing a hyperplane in a feature space defined by a particular feature map $\phi(\cdot)$ and a normal vector to the hyperplane, denoted as w . According to the classification criterion, a pattern x is classified as "normal" if the dot product of w and $\phi(x)$ is higher than ρ . The optimization issue aims to determine the hyperplane, namely the normal vector w and the value of ρ [16].

$$\begin{aligned} & \frac{1}{2} \|w\|_{\ell_2}^2 + \frac{1}{\nu l} \sum_{i=1}^l \xi_i - \rho \rightarrow \min_{w, \xi, \rho} \\ s.t. \quad & (w \cdot \phi(x_i)) \geq \rho - \xi_i, \quad \xi_i \geq 0. \end{aligned}$$

Eq 4.1

Regarding the i -th pattern, there is a regularization coefficient and ξ_i represents a slack variable. The answer to the convex optimization problem in equation 3.2 coincides with its dual solution[16].

$$\begin{aligned} & - \sum_{i=1}^l \sum_{j=1}^l \alpha_i \alpha_j K(x_i, x_j) \rightarrow \max_{\alpha} \\ s.t. \quad & \sum_{i=1}^l \alpha_i = 1, \quad 0 \leq \alpha_i \leq \frac{1}{\nu l}. \end{aligned}$$

Eq 4.2

In this situation, the scalar product $(\phi(x_i) \cdot \phi(x_j))$ is substituted by the equivalent kernel function $K(x_i, x_j)$. Therefore, it is typical to solve the dual issue without the need for an explicit formulation of $\phi(x)$. Furthermore, the answer to the primal problem may be obtained by solving the dual problem, where w is equal to the sum of α_i multiplied by $\phi(x_i)$. Equation 3.3 establishes

that x_i is deemed to be on the boundary of the "normal" domain if there is a positive α_i such that $(w \cdot \phi(x_i))$ equals ρ . Therefore, the offset ρ may be found by utilizing the fact that the matching pattern x_i satisfies the equality for all positive α_i .

$$\rho = (w \cdot \phi(x_i)) = \sum_{j=1}^l \alpha_j K(x_j, x_i).$$

Eq 4.3

The relevant choice rule is structured in the following form:

$$f(x) = \sum_{i=1}^l \alpha_i K(x_i, x) - \rho.$$

Eq 4.4

A pattern x is considered to be in the "normal" class if $f(x)$ is greater than 0, and conversely. The user's text is "[16]".

4.2 Comparison:

In the table below you will be able to notice the differences between the algorithms:

Table 4.1 - Comparison Between Algorithm Techniques

Technique	Pros	Cons

K-Nearest Neighbor	<ul style="list-style-type: none"> • When there are few predictor variables, it is very straightforward to grasp. • Good for developing models using non-standard data sources, like text. 	<ul style="list-style-type: none"> • need a lot of storage • . Sensitive to the selection of the similarity function utilised for instance comparison. • Ignore a logical method for selecting k other than cross-validation or anything comparable. • A costly computational method.
Support Vector Machine	<ul style="list-style-type: none"> • Locate the hyperplane of ideal separation. • Able to handle data with a very high dimensionality. • Certain kernels possess an infinite Vapnik-Chervonenkis dimension, enabling them to acquire highly complex notions. • Usually function quite nicely. 	<ul style="list-style-type: none"> • Require both positive and negative examples. • Need to select a good kernel function. • Requires lots of memory and CPU time. • There are some numerical stability problems in solving the constraint QP.

Decision Tree	<ul style="list-style-type: none"> • Easy to comprehend and analyze. • Minimal data preparation is needed. • Capable of managing data that is both category and numerical. • Employs a white box paradigm. • It is possible to use statistical tests to validate a model. • Strong. • Accomplish tasks quickly while handling a lot of data. 	<ul style="list-style-type: none"> • It is recognized that learning an optimum decision tree is an NP-complete issue under many optimality conditions, even for basic notions. • Overly intricate trees with poor data generalisation are produced by decision-tree learners. • Some ideas are difficult to understand because decision trees make it difficult to describe them.
Expectation Maximization Meta	<ul style="list-style-type: none"> • Can easily change the model to adapt to a different distribution of data sets. • Parameters number does not increase with the training data increasing. 	<ul style="list-style-type: none"> • Slow convergence in some cases
Fuzzy C-means	<ul style="list-style-type: none"> • permits a data point to reside in more than one cluster. • A more accurate depiction of how genes behave. 	<ul style="list-style-type: none"> • The cluster number, c, needs to be defined. • Determining the membership cutoff value is necessary.

		<ul style="list-style-type: none"> • Clusters are susceptible to the centroids' initial assignment.
K-Means	<ul style="list-style-type: none"> • Low complexity 	<ul style="list-style-type: none"> • k must be specified • it will be sensitive to noise and anomalous data points. • Clusters are susceptible to the centroids' initial assignment.
Neural Network	<ul style="list-style-type: none"> • What a linear program cannot do, a neural network can. • The neural network's ability to function even in the event of an element failure makes it distinctive, particularly for dimensionality reduction. 	<ul style="list-style-type: none"> • To function, the neural network requires training. • It is necessary to imitate the architecture of a neural network, as it differs from that of microprocessors. • Large neural networks require a high processing time.

4.3 Data Collect:

To detect fraudulent APKs using the One-Class SVM approach, we utilized a publically available dataset from Kaggle. The dataset is widely regarded in the cybersecurity and machine learning areas for its comprehensive compilation of both benign and malicious APK files. The article offers an extensive overview of many characteristics derived from APK files,

encompassing permissions, API calls, and other static and dynamic attributes that reflect their activity. The collection has two primary subsets: benign APKs, which have been verified as secure and non-dangerous, and malicious APKs, detected based on various criteria such as recognized malware signatures, abnormal behaviors, or inclusion in threat intelligence databases.

In order to enhance the accuracy of detection, the dataset offers a range of essential features that are crucial for differentiating between harmless and harmful APKs. The features can be broadly classified into two categories: static features and dynamic features. Static features can be extracted without executing the APK and include information such as permissions requested, package name, and size of the APK. On the other hand, dynamic features describe behaviors exhibited during execution, such as API calls, network traffic patterns, and resource usage. Static features are highly useful for making rapid initial assessments, whereas dynamic features provide more in-depth insights into the behavior of the APK. However, analyzing dynamic features necessitates the use of advanced analysis tools and environments.

Prior to utilizing the One-Class SVM method, various preprocessing steps were implemented to guarantee the quality and relevance of the data. Significant features were chosen based on their ability to differentiate between benign and malicious APKs, while eliminating irrelevant or redundant features to enhance the efficiency and accuracy of the model. Normalization of the selected features was performed to ensure they are on a comparable scale. This step is crucial for the One-Class SVM algorithm, as it relies on distance metrics that can be influenced by features of varying scales. Ultimately, the dataset was split into two sets: the training set and the testing set. The training set mainly comprised benign APKs, which were used to train the One-Class SVM model. On the other hand, the testing set included both benign and malicious APKs, allowing for an evaluation of the model's performance.

Through meticulous curation and preprocessing of the dataset, we guarantee that our One-Class SVM model is trained on top-notch, representative data. By utilizing this approach, we were able to successfully detect anomalies that are suggestive of malicious APK behavior. As a result, our detection system has been significantly improved in terms of reliability and accuracy.

4.4 Advantages of One-Class SVM:

Choosing the most suitable machine learning approach to detect malware in APK files depends on several factors, including the characteristics of the dataset, the specific problem at hand, and the desired outcomes. One-Class Support Vector Machines (One-Class SVM) may offer various benefits over the other algorithms indicated in the table for identifying malware in APK files.

1. Addressing the issue of imbalanced datasets: The One-Class SVM approach is specifically designed to identify anomalies in situations where one class (benign APKs) is far more prevalent than the other (malicious APKs). When it comes to identifying malware, it is common to come into an imbalance between benign and hazardous samples. The One-Class SVM algorithm excels in such scenarios by assimilating information about the characteristics of the prevailing class throughout the training phase and identifying any departures as anomalies.

2. Absence of negative incidents Essential: The One-Class SVM technique only requires examples from the majority class for training, making it suitable for situations where obtaining representative samples of the minority class (malicious APKs) is challenging or expensive. This is a common occurrence in the world of malware detection, where there is a limited number of known malicious samples.

3. Dimensionality Reduction: One-Class SVM is effective in lowering the dimensionality of data with a large number of dimensions, such as the characteristics derived from APK files. It is crucial to extract significant information and patterns from the data while minimizing the issue of excessive variables, which might provide challenges for algorithms like K-Nearest Neighbors or Neural Networks.

4. Noise Robustness: The One-Class SVM method exhibits a propensity to withstand noise and outliers that may be present in the dataset. The One-Class Support Vector Machine (SVM) is effective in detecting malware, even when hazardous samples have varying features. This is because the One-Class SVM can prioritize the majority class while identifying anomalies, enabling it to handle noise and variety.

5. One-Class SVM is a highly effective method for effectively detecting anomalies, particularly in complex and high-dimensional domains. The effectiveness of this system is based on its ability to gather information about normal activity and identify abnormalities, allowing it to accurately detect unusual malicious patterns in APK files.

It is important to recognize that the choice of the best method depends on the specific characteristics of the dataset and the problem being solved. The efficacy of One-Class SVM can be affected by variables such as the data's properties, the extracted features, and the specific requirements of the malware detection task in APK files, resulting in variances in its performance. Conducting empirical assessments and comparisons is recommended to determine the most efficient strategy for a certain circumstance.

Chapter 5: Implementation:

This chapter explores the use of the One-Class Support Vector Machine (SVM) approach for detecting anomalies in a given dataset. The second component of the final project will employ this method, with a focus on adapting an APK dataset. This chat provides a summary of the previous chapter, where we will demonstrate the effectiveness of the One-Class SVM model in identifying anomalies in the provided APK dataset.

5.1 Imports

```
# Install the required packages
!pip install -U androguard
!pip install -U scikit-learn

# Importing necessary modules
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.svm import OneClassSVM
from sklearn.metrics import classification_report
import pandas as pd
import numpy as np
from collections import Counter
import logging
import os
import matplotlib.pyplot as plt

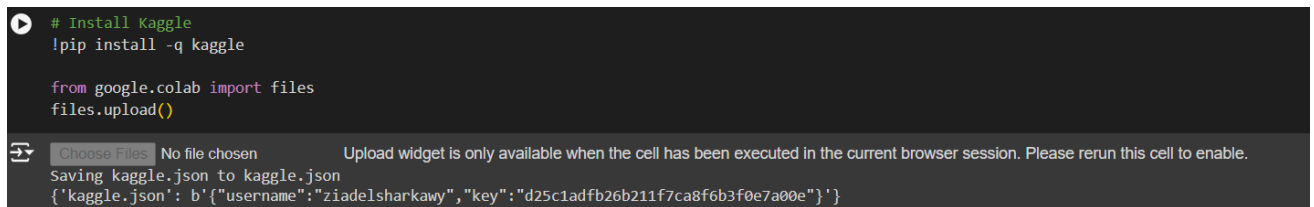
# Correct Androguard imports
from androguard.core.analysis.analysis import Analysis
from androguard.core.apk import APK
from androguard.core.dex import DEX # Adjusted import path
from androguard.misc import AnalyzeAPK
import pickle
import networkx as nx

logging.basicConfig(level=logging.INFO)
```

Fig 5.1 Imports

The installations verifies that the necessary packages, androguard and scikit-learn, are installed and upgraded to their most recent versions. The androguard library is utilized for the analysis of Android apps (APKs), while scikit-learn is employed for the implementation of machine learning techniques. The scikit library provides several useful functions for classification tasks, including `make_classification`, `train_test_split`, `OneClassSVM`, and `classification_report`. Learn the process of creating datasets, dividing them, applying the One-Class SVM method, and testing the model, in that order. The libraries `pandas` and `numpy` are used for manipulating data and performing numerical computations. The "Counter" class from the "collections" module is used to count the occurrences of items in a dataset. The purpose of this is to log information. `os` for interfacing with the operating system. The library `matplotlib.pyplot` is used for the purpose of graph charting. `Analysis`, `APK`, and `DEX` for analyzing APK files and DEX (Dalvik Executable) files. `AnalyzeAPK` for analyzing APK files in a simplified manner. `pickle` for serializing and deserializing Python objects. `networkx` for creating and manipulating complex networks/graphs.

5.2 Connect google colab to Kaggle dataset



```
# Install Kaggle
!pip install -q kaggle

from google.colab import files
files.upload()
```

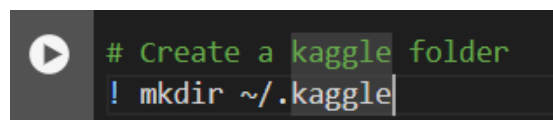
Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Saving kaggle.json to kaggle.json

```
{'kaggle.json': b'{"username": "ziadelsharkawy", "key": "d25c1adfb26b211f7ca8f6b3f0e7a00e"}'}
```

Fig 5.2 - installation of Kaggle

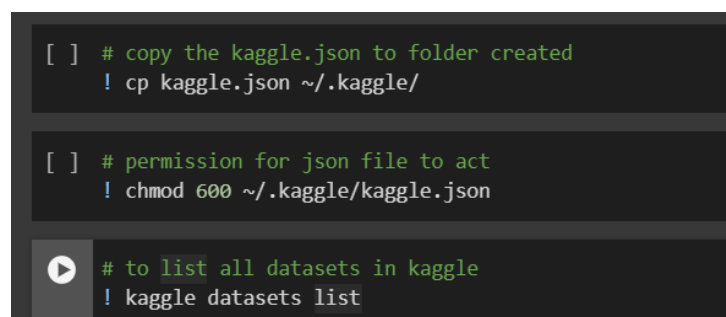
This command installs the kaggle package, which serves as a tool for communicating with the Kaggle API. This feature is advantageous for directly retrieving datasets from Kaggle.



```
# Create a kaggle folder
! mkdir ~/.kaggle
```

Fig 5.3 - Creating Folder

This command generates a concealed directory called .kaggle in the primary directory. The purpose of this directory is to keep the kaggle.json file, which has your API credentials for authenticating with Kaggle.



```
[ ] # copy the kaggle.json to folder created
! cp kaggle.json ~/.kaggle/

[ ] # permission for json file to act
! chmod 600 ~/.kaggle/kaggle.json

# to list all datasets in kaggle
! kaggle datasets list
```

Fig 5.4 - Copying, permission and listing kaggle datasets

The command `! cp kaggle.json ~/.kaggle/` copies the file named "kaggle.json" from the current directory to the directory named ".kaggle". The command `! chmod 600`

~/kaggle/kaggle.json" sets the file permissions of the file "kaggle.json" in the directory ".kaggle" so that only the owner may read and write to it. This measure is crucial to ensure security and safeguard your Kaggle API credentials.

! kaggle datasets list : Enumerates all datasets accessible on Kaggle. We use this as a valuable tool for confirming the proper functioning of the Kaggle API and for investigating the datasets that are currently accessible.

```
[ ] # Downloading dataset from kaggle
!kaggle datasets download -d tuanasleevawn/apk-files

Dataset URL: https://www.kaggle.com/datasets/tuanasleevawn/apk-files
License(s): Apache 2.0
Downloading apk-files.zip to /content
 98% 233M/238M [00:02<00:00, 85.1MB/s]
100% 238M/238M [00:02<00:00, 95.1MB/s]

[ ] !unzip apk-files.zip
```

Fig 5.5 -Downloading kaggle dataset

Download the Kaggle dataset with the command "!kaggle datasets download -d".
tuanasleevawn/apk-files repository: Retrieves the dataset from Kaggle. The dataset identifier is tuanasleevawn/apk-files. To extract the contents of the zip file, use the command "unzip apk-files.zip". Extracts the apk-files.zip dataset file.

5.3 Extracting Features

```
def extract_permissions(apk_path):
    try:
        a, d, dx = AnalyzeAPK(apk_path)
        logging.info(f"Extracted permissions from: {apk_path}")
        return a.get_permissions()
    except Exception as e:
        logging.error(f"Error analyzing APK: {apk_path}, error: {e}")
        return []

def apk_to_feature_vector(apk_path, permission_to_index, all_permissions):
    permissions = extract_permissions(apk_path)
    feature_vector = np.zeros(len(all_permissions))
    for perm in permissions:
        if perm in permission_to_index:
            feature_vector[permission_to_index[perm]] = 1
    return feature_vector
```

Fig 5.6 - extracting permission

1- The **extract_permissions** method retrieves the permissions from an APK file.

Specifications:

apk_path: The location of the APK file. The output is a list of permissions that are necessary for the APK. In the event of an error occurring during the analysis process, the outcome will be an empty list.

Specifics:

The APK file is analyzed using the AnalyzeAPK function from androguard. Records a message to indicate a successful extraction or reports an error if one occurs.

2- **apk_to_feature_vector** is a function that turns an APK file into a feature vector by analyzing its permissions.

Specifications:

apk_path: The location of the APK file. The variable "permission_to_index" is a dictionary that associates permission names with their corresponding indices in the feature vector.

all_permissions: A comprehensive compilation of all potential permissions. The output is a feature vector, represented as a NumPy array, that indicates the existence or lack of permissions.

Specifics:

The APK invokes the extract_permissions method to obtain the permissions. Creates a feature vector of the same length as all_permissions, where all elements are set to zero. Assigns a value of 1 to the respective index in the feature vector for each permission that is found in the APK.

5.4 Set dataset paths and initialize variables:

See figure in next page:

```
benign_dir = '/content/Benign'
malware_dir = '/content/Malware'

all_permissions = set()
benign_files = []
malware_files = []

# Recursively find APK files and extract permissions
for root, dirs, files in os.walk(benign_dir):
    for file in files:
        if file.endswith(".apk"):
            apk_path = os.path.join(root, file)
            permissions = extract_permissions(apk_path)
            if permissions: # Ensure permissions were extracted
                all_permissions.update(permissions)
                benign_files.append(apk_path)
                logging.info(f"Added benign file: {apk_path}")

for root, dirs, files in os.walk(malware_dir):
    for file in files:
        if file.endswith(".apk"):
            apk_path = os.path.join(root, file)
            permissions = extract_permissions(apk_path)
            if permissions: # Ensure permissions were extracted
                all_permissions.update(permissions)
                malware_files.append(apk_path)
                logging.info(f"Added malware file: {apk_path}")
```

Fig 5.7 - Set dataset paths and initialize variables

1-Initialize Variables:

- `benign_dir` and `malware_dir`: Directories containing benign and malware APK files, respectively.
- `all_permissions`: A set to store all unique permissions across both benign and malware APK files.
- `benign_files`, `malware_files` : Lists to store the file paths of benign and malware APK files.

2-Recursively Find APK Files, Extract Permissions, log information and permission to index mapping:

- **Benign APK Files:**
 - Traverse the benign_dir directory to find APK files.
 - For each APK file, extract its permissions using extract_permissions.
 - Add the extracted permissions to the all_permissions set and the file path to the benign_files list.
- **Malware APK Files:**
 - Traverse the malware_dir directory to find APK files.
 - For each APK file, extract its permissions using extract_permissions.
 - Add the extracted permissions to the all_permissions set and the file path to the malware_files list.
- **Log Information:**
 - Log the total number of benign and malware files and the total number of unique permissions extracted.
- **Create a Permission to index mapping:**
 - Create a dictionary permission_to_index that maps each unique permission to an index. This will be used to create feature vectors.

```
logging.info(f"Total benign files: {len(benign_files)}")
logging.info(f"Total malware files: {len(malware_files)}")
logging.info(f"Total unique permissions: {len(all_permissions)}")

permission_to_index = {perm: idx for idx, perm in enumerate(all_permissions)}
```

Fig 5.8 -Prints total benign, malware and total number of permissions

5.5 Scaling the Feature Vectors

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

# Initialize feature vectors
X_benign = []
X_malware = []

# Convert lists to numpy arrays
X_benign = np.array(X_benign)
X_malware = np.array(X_malware)

# Scale features if non-empty
if X_benign.size > 0:
    X_benign_scaled = scaler.fit_transform(X_benign)

if X_malware.size > 0:
    X_malware_scaled = scaler.transform(X_malware)
```

Fig 5.9 - Scaling Features

This code snippet uses the `StandardScaler` from the `sklearn.preprocessing` package to standardize feature vectors that represent benign and malicious data. At first, two empty lists, `X_benign` and `X_malware`, are created to hold these feature vectors. The lists are subsequently transformed into NumPy arrays, specifically referred to as `X_benign` and `X_malware`.

The `StandardScaler` is created as `scaler` and it will normalize the data by removing the average and scaling it to have a variance of one. The code verifies if the arrays `X_benign` and `X_malware` are not empty by checking whether their sizes are more than zero (`X.size > 0`). whether this condition is met, it proceeds to perform the scaling transformation on `X_benign` using the `scaler.fit_transform()` method, and on `X_malware` using the `scaler.transform()` method.

5.6 Training the algorithm:

```
# Train the One-Class SVM model using only benign data
if X_benign_scaled.size > 0:
    oc_svm = OneClassSVM(kernel='rbf', gamma='auto').fit(X_benign_scaled)

# Predict using the model
if X_benign_scaled.size > 0 and X_malware_scaled.size > 0:
    y_pred_benign = oc_svm.predict(X_benign_scaled)
    y_pred_malware = oc_svm.predict(X_malware_scaled)

    # Convert to 0 (normal) and 1 (anomalous)
    y_pred_benign = np.where(y_pred_benign == 1, 0, 1)
    y_pred_malware = np.where(y_pred_malware == 1, 0, 1)

    # True labels
    y_true_benign = np.zeros(len(y_pred_benign))
    y_true_malware = np.ones(len(y_pred_malware))

    # Combine benign and malware predictions
    y_true = np.concatenate((y_true_benign, y_true_malware))
    y_pred = np.concatenate((y_pred_benign, y_pred_malware))

    # Evaluate the model
    print(classification_report(y_true, y_pred))
else:
    logging.error("Insufficient data to train or test the model.")
```

Fig 5.10 - Training the algorithm

The code segment supplied illustrates the procedure of training and assessing a One-Class SVM model to differentiate between benign and malicious APK files using their extracted permissions. At first, the One-Class SVM model is only trained using the benign APK data, which has undergone preprocessing and scaling using the `StandardScaler`. The model utilizes the Radial Basis Function (RBF) kernel with the gamma parameter set to 'auto'. This stage guarantees that the model acquires knowledge about the typical behavior shown by harmless files.

Once the model has been trained, it is employed to classify both benign and malicious APK files. The predictions are subsequently transformed, so that a prediction of 1 (representing normal) is turned to 0, indicating benign, while -1 (representing anomalous) is converted to 1, signifying malware. This conversion is essential as it ensures that the prediction labels are in line with the genuine labels for the purpose of assessment.

The benign and malware datasets are assigned true labels, with benign files classified as 0 and malicious files labeled as 1. The labels and their accompanying predictions are consolidated

into two arrays, `y_true` and `y_pred`. The concatenated arrays are utilized to assess the model's performance by employing the `classification_report` function from `sklearn.metrics`. This function generates a comprehensive report that includes precision, recall, f1-score, and support for each class. The last conditional check guarantees that the evaluation will only continue if there is a enough amount of data available for both training and testing. If the dataset is insufficient, an error message will be logged. This systematic method successfully emphasizes the model's capacity to differentiate benign and malicious APK files.

Result:

	precision	recall	f1-score	support
0.0	0.98	0.51	0.67	107
1.0	0.60	0.99	0.74	78
accuracy			0.71	185
macro avg	0.79	0.75	0.71	185
weighted avg	0.82	0.71	0.70	185

Fig 5.11 - Results Scores

The provided table is a classification report that evaluates the effectiveness of a machine learning model in categorizing APK files into two groups: benign (class 0) and malware (class 1). Every row in the table represents a class, while the columns provide many important metrics:

Precision quantifies the degree of correctness of positive predictions provided by the model. The accuracy for class 0 (benign) is 0.98, meaning that when the model classifies a file as benign, it is accurate 98% of the time. However, when it comes to class 1 (malware), the accuracy is 0.60, indicating that the model accurately detects 60% of the files that are anticipated to be malware.

Recall measures the model's capacity to accurately detect genuine positives among all the real positives. The recall rate for class 0 is 0.51, showing that the model accurately recognizes 51% of benign files. Conversely, for class 1, the recall rate is 0.99, signifying that it properly identifies 99% of malware files.

The F1-score is a mathematical average that combines accuracy and recall, offering a well-balanced evaluation of the model's effectiveness. The F1-score for class 0 is 0.67, and for class 1, it is 0.74, suggesting a favorable equilibrium between accuracy and recall. This is particularly significant in the context of malware detection.

Support refers to the frequency or count of each class in the dataset. There are a total of 107 benign files (class 0) and 78 malicious files (class 1).

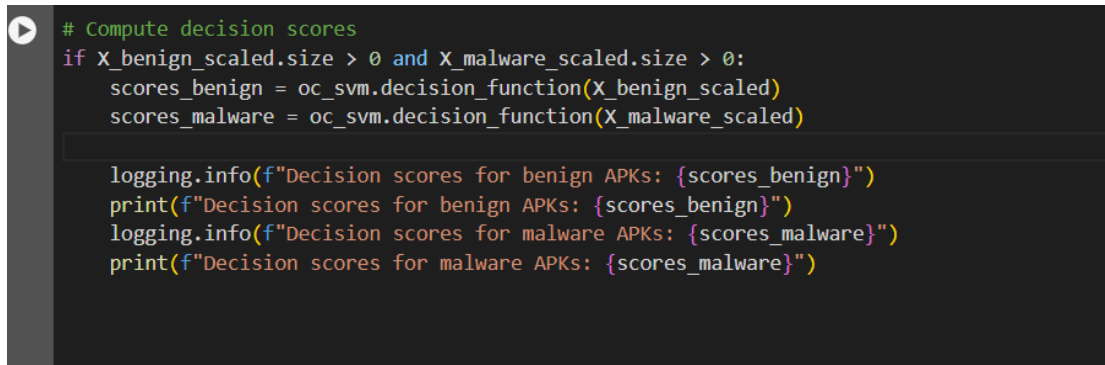
Accuracy, although not expressly mentioned in the table, denotes the overall accuracy of the model's predictions across all categories.

The macro average calculates the average of accuracy, recall, and F1-score without taking into account the class distribution. The accuracy is 0.79, the recall is 0.75, and the F1-score is 0.71.

The weighted average calculates the statistic by taking into account the number of examples for each class. The accuracy, recall, and F1-score are 0.82, 0.71, and 0.70, respectively.

To conclude, the classification report provides a concise overview of the model's ability to differentiate between benign and malicious APK files. It has exceptional accuracy in recognizing harmless files and impressive ability to recognize malicious software. The F1-scores demonstrate an equitable performance across both classes, with support values reflecting the distribution of the dataset. These metrics together offer a thorough assessment of the model's efficacy and identify possible areas for enhancement in categorizing APK files.

5.7 Computing Decision Scores



```
# Compute decision scores
if X_benign_scaled.size > 0 and X_malware_scaled.size > 0:
    scores_benign = oc_svm.decision_function(X_benign_scaled)
    scores_malware = oc_svm.decision_function(X_malware_scaled)

logging.info(f"Decision scores for benign APKs: {scores_benign}")
print(f"Decision scores for benign APKs: {scores_benign}")
logging.info(f"Decision scores for malware APKs: {scores_malware}")
print(f"Decision scores for malware APKs: {scores_malware}")
```

Fig 5.12 - Computing Decision Scores code

The offered code snippet calculates decision scores by utilizing the `decision_function` method of the One-Class SVM (`oc_svm`) model. Decision scores quantify the proximity of each sample (APK file) to the separation hyperplane acquired using the SVM model. Below is a comprehensive elucidation presented in the style of a paragraph:

The code starts by verifying if there are enough data points (`X_benign_scaled` and `X_malware_scaled`) to calculate decision scores for both benign and malware APK files. Given the presence of data, the `decision_function` method is called on the One-Class SVM (`oc_svm`) model. This approach computes a numerical value for each instance, with a greater value indicating a higher probability of the instance belonging to the usual category (benign), and a lower value suggesting it is more likely to be an anomaly (malware).

The decision scores calculated are saved in the variables `scores_benign` and `scores_malware` for benign and malware APK files, respectively. The scores serve as a numerical indicator of the model's certainty in its predictions. Higher scores for benign APKs indicate a greater level of conformity to regular behavior, while lower scores for malware APKs imply a departure from typical patterns, perhaps signaling malevolent conduct.

Logging statements, specifically the `logging.info` function, are utilized to document the decision scores of both benign and malicious APKs. This process aids in the monitoring and

analysis of the model's activity. In addition, `print` statements are used to immediately display the decision scores in the output for instant examination and analysis.

To summarize, the generated decision scores provide valuable information about how the model classifies APK files. They give a numerical evaluation of the probability that each file is either benign or malicious, based on its proximity to the decision border learnt by the One-Class SVM model. The scores are a helpful tool for future study and refinement of the model's ability to differentiate between benign and malicious APK files.

Results:

```
Decision scores for benign APKs: [ 2.06581190e+00 2.06581190e+00 1.64257863e-04 -2.03041032e+00
1.91917412e+00 1.91917412e+00 1.91917412e+00 -3.67931808e-01
-2.65873838e+01 -2.84941192e+00 -2.54458825e-01 -5.71286184e+00
5.16024527e-01 1.64257863e-04 2.95890403e-01 -3.50041455e-02
1.91917412e+00 -1.48234434e+01 6.20224343e-01 9.74822844e-01
-2.05095460e+00 -1.27461190e+01 -4.56183942e+00 4.40188185e-01
2.95890403e-01 -4.50632273e-01 -5.07016774e+00 1.91917412e+00
-2.64924021e+01 -2.05053010e+01 -4.92663654e-04 2.66508619e-01
1.91917412e+00 -3.29449050e+00 -1.49300997e+01 1.64209891e-04
-4.69426409e+00 -1.80590960e+01 -5.32941400e-01 1.55121010e+00
-8.30375172e+00 1.91917412e+00 -2.37215764e+00 -1.07113476e+00
1.13911189e+00 -1.69442783e+01 -2.53264551e+01 1.56754864e+00
-4.40542073e+00 6.20224343e-01 1.84476937e+00 -2.41911150e+01
1.58524228e+00 1.91917412e+00 -1.36782439e+01 -2.47456050e+01
1.01895002e+00 1.64257863e-04 2.06581190e+00 1.93823378e-01
2.06581190e+00 7.12136919e-01 2.06581190e+00 -4.70454867e-01
1.91917412e+00 -2.65610379e+01 1.64257863e-04 -2.43431357e+00
-1.32513941e+01 -1.38412532e+01 2.86479186e-01 -1.07019060e+01
-2.63505689e+01 -1.29344303e+01 1.51026327e+00 1.91917412e+00
-4.15009144e-01 1.91917412e+00 1.91917412e+00 1.55121010e+00
1.64209891e-04 1.56217292e+00 -2.05989726e+00 -1.20424327e+00
-9.05202205e-01 1.51026327e+00 1.56442014e-01 -4.01419158e+00
2.06581190e+00 1.91917412e+00 -3.63741596e-01 -4.34517962e-01
9.63041575e-01 -1.03208759e+01 5.16024527e-01 2.06581190e+00
-4.01479237e+00 1.84476937e+00 1.64257863e-04 -4.01419158e+00
1.02318315e+00 -1.14335882e+01 -1.74522093e+01 1.91917412e+00
-1.29277440e-01 -2.57694646e+00 6.20224343e-01]
```

Fig 5.13 - Malware Scores

```
Decision scores for malware APKs: [-27.41564655 -27.41449933 -16.94217823 -27.41564655 -27.41449933
-27.41449933 -27.41449933 -2.23195695 -27.41449933 -27.41449933
-2.75037501 -27.41564655 -27.41449933 -2.23195695 -27.41449933
-27.41449933 -24.32363687 -27.29533974 -27.63982842 -27.13930305
-27.51578555 -27.41449933 -25.31035405 -2.23195695 -25.18446843
-27.41449933 -27.13930305 -27.41564655 -27.41564655 -27.41449933
-21.11800621 -26.41751374 -27.41564655 -27.41449933 -27.29533974
-27.51578555 -24.17673234 -27.41449933 -27.41449933 -2.23195695
-27.41564655 -27.41449933 -27.41449933 -19.67928335 1.91917412
-27.29533974 -20.03526001 -25.44625011 -22.96006992 -27.29533974
-1.16594025 -27.41564655 -24.49632409 -27.41564655 -2.23195695
-9.47731517 -2.23195695 -27.41564655 -16.94217823 -27.41449933
-27.29533974 -27.41449933 -27.41449933 -27.41449933 -26.37224209
-27.41449933 -25.53157966 -2.23195695 -19.67928335 -7.53895976
-27.41449933 -27.41449933 -27.29533974 -27.41449933 -27.41564655
-27.41564655 -27.41449933 -27.41449933]
```


Fig 5.14 -Malware Scores

The given decision scores correspond to the numerical results of the decision function of the One-Class SVM model for benign and malicious APK files. Decision scores reflect the level of certainty the model has in its predictions. Higher scores generally indicate that an APK file closely follows the observed pattern of harmless behavior, while lower (more negative) values imply that an APK file diverges from this pattern and may be classified as malware.

The decision scores for benign APKs vary from positive values such as 2.07 to negative ratings close to -27.41. Positive ratings over 1.84, such as 2.07 and 1.92, show a strong adherence to the innocuous class. On the other hand, negative scores below -26.53, like -27.41, represent a severe divergence from typical behavior. These cases may be flagged as anomalies or outliers.

In contrast, the judgment scores for malware APKs are mainly quite unfavorable, with values ranging from around -27.42 to -1.17. The results suggest that the algorithm accurately detects these files as anomalies or outliers, which aligns with their categorization as malware. It is worth mentioning that certain ratings for malware APKs, such as -1.17 and -2.23, are less negative than others. This might suggest that certain APKs deviate from the benign behavior model to different extents.

The decision scores give a numerical foundation for comprehending how the One-Class SVM model differentiates between benign and malicious APK files. They provide insights into the model's classification judgments by considering the distance of each APK file from the decision boundary established during training. Evaluating the model's performance is crucial and can provide valuable insights for analyzing and improving the categorization process.

5.8 Customizing Threshold

```
def predict_with_threshold(scores, threshold):  
    return np.where(scores < threshold, 1, 0)  
  
# Customize the threshold  
custom_threshold = -0.5 # Example threshold, you can change this value  
  
y_pred_benign_custom = predict_with_threshold(scores_benign, custom_threshold)  
y_pred_malware_custom = predict_with_threshold(scores_malware, custom_threshold)  
  
# True labels  
y_true_benign = np.zeros(len(y_pred_benign_custom))  
y_true_malware = np.ones(len(y_pred_malware_custom))  
  
# Combine benign and malware predictions  
y_true_custom = np.concatenate((y_true_benign, y_true_malware))  
y_pred_custom = np.concatenate((y_pred_benign_custom, y_pred_malware_custom))  
  
# Evaluate the model with custom threshold  
print("Custom Threshold Classification Report")  
print(classification_report(y_true_custom, y_pred_custom))
```

Fig 5.15 - Customizing Threshold Code

The above code segment exemplifies the implementation of a customized threshold to decision scores produced by a One-Class SVM model. It then proceeds to assess the model's performance using a classification report based on this threshold. Below is a detailed explanation presented in paragraph format:

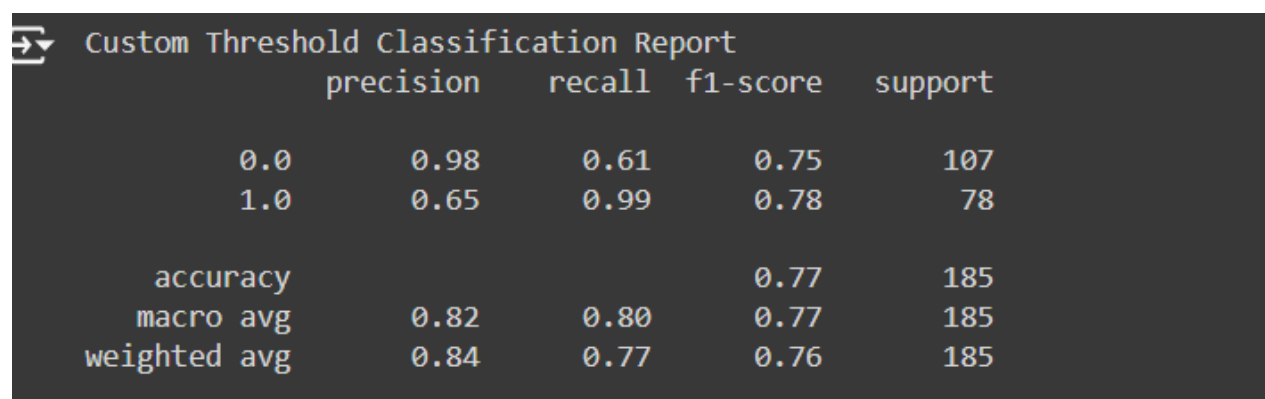
The `predict_with_threshold(scores, threshold)` function is implemented to categorize samples by comparing their decision scores to a provided threshold. The function assigns a label of 1 to scores that are below the threshold, indicating an anomaly or malware. Scores that are equal to or above the threshold are assigned a label of 0, signifying normal or benign. This function is independently applied to the decision scores (`scores_benign` and `scores_malware`) that were previously produced using the One-Class SVM model.

In this instance, a specific threshold value of -0.5 is selected to serve as a dividing line. Any scores below -0.5 are categorized as malware, while scores equal to or above -0.5 are labeled as benign. The threshold can be modified based on the desired balance between false positives and false negatives in the classification task.

The ``predict_with_threshold`` function calculates the predicted labels for benign and malware APKs, denoted as ``y_pred_benign_custom`` and ``y_pred_malware_custom`` correspondingly, using a custom threshold. Subsequently, the user generates true labels (``y_true_benign`` and ``y_true_malware``), assigning a value of 0 to benign APKs and a value of 1 to malware APKs.

Subsequently, ``y_true_custom`` and ``y_pred_custom`` are created by combining the actual and anticipated classifications for benign and malicious APKs. The purpose of these arrays is to construct a classification report using the ``classification_report`` function from the ``sklearn.metrics`` library. This function calculates and presents precision, recall, F1-score, and support for both classes (0 for benign and 1 for malware) based on a specific threshold.

The result of this procedure, known as the "Custom Threshold Classification Report," offers a comprehensive assessment of the model's effectiveness using the selected threshold. It provides a summary of the model's ability to differentiate between benign and malicious APK files based on a certain decision boundary. This helps evaluate the model's efficacy and allows for adjusting thresholds to suit different application circumstances. This technique enables precise adjustment of the model's behavior to satisfy certain performance criteria, such as optimizing for better accuracy or recall based on the requirements of the application.



	precision	recall	f1-score	support
0.0	0.98	0.61	0.75	107
1.0	0.65	0.99	0.78	78
accuracy			0.77	185
macro avg	0.82	0.80	0.77	185
weighted avg	0.84	0.77	0.76	185

Fig 5.16 - Customizing Threshold Result

The "Custom Threshold Classification Report" provides a concise overview of the effectiveness of a machine learning model, especially a One-Class SVM, in classifying APK files as either benign or malicious, using a customized decision threshold. Below is a detailed explanation of the report:

Precision, recall, and F1-score are metrics used to evaluate the performance of a classification model.

Accuracy: This statistic quantifies the accuracy of predictions by calculating the ratio of accurately predicted instances of each class to the total number of examples anticipated as that class. The accuracy for class 0 (benign) is 0.98, meaning that the model correctly forecasts an APK file as benign 98% of the time. The accuracy for class 1 (malware) is 0.65, indicating that the model accurately identifies an APK file as malware 65% of the time.

Recall is a metric that quantifies the accuracy of a model in properly identifying genuine positive cases among all the real positive instances. The recall for class 0 is 0.61, meaning that the model accurately detects 61% of all benign APK files. The recall for class 1 is 0.99, indicating that the model accurately detects 99% of all malicious APK files.

The F1-score is a mathematical measure that combines accuracy and recall in a balanced way, offering a single metric to assess the trade-off between precision and recall for each class. The F1-score for class 0 is 0.75, showing a favorable equilibrium between accuracy and recall for benign files. In class 1, the F1-score is 0.78, indicating a relatively equal performance in accurately recognizing malicious files.

Assistance:

Support: This statistic quantifies the frequency of each class in the dataset used for assessment, measured by the number of instances (APK files) per class. There are a total of 107 benign files (class 0) and 78 malicious files (class 1).

Precision:

Precision: The model's total accuracy, encompassing all classes, is 0.77, signifying that 77% of all APK files were accurately categorized by the model.

Explanation:

The high precision for class 0 (benign) suggests that when the model identifies an APK file as benign, it is very probable that the prediction is accurate. Nevertheless, the decreased accuracy in identifying class 1 (malware) implies a greater likelihood of misidentifying innocuous items as malware.

The high recall for class 1 signifies that the model accurately detects nearly all occurrences of malware in the dataset. The lower recall rate for class 0 indicates that there are instances where innocuous files are mistakenly identified as malware.

The F1-scores for both groups demonstrate an equitable performance in terms of accuracy and recall, which is essential in situations where minimizing both false positives and false negatives is important.

In general, the model demonstrates a strong overall performance in accurately differentiating between benign and malicious APK files, as indicated by the high weighted average accuracy, recall, and F1-score. This performance is achieved while utilizing the supplied custom threshold.

The "Custom Threshold Classification Report" offers a thorough assessment of the model's performance in categorizing APK files based on a specific decision threshold. It emphasizes the model's ability to accurately identify malware and suggests potential enhancements, such as minimizing false positives in benign file classifications.

5.9 Visualization:

```
import seaborn as sns

# Combine scores for visualization
scores = np.concatenate((scores_benign, scores_malware))
labels = np.concatenate((np.zeros(len(scores_benign)), np.ones(len(scores_malware))))

# DataFrame for visualization
df_scores = pd.DataFrame({'Scores': scores, 'Labels': labels})

# Plot score distributions
plt.figure(figsize=(12, 6))
sns.histplot(df_scores[df_scores['Labels'] == 0]['Scores'], color='blue', label='Benign', kde=True, stat="density", linewidth=0)
sns.histplot(df_scores[df_scores['Labels'] == 1]['Scores'], color='red', label='Malware', kde=True, stat="density", linewidth=0)
plt.axvline(x=custom_threshold, color='black', linestyle='--', label='Threshold')
plt.title('Distribution of Decision Scores')
plt.xlabel('Decision Scores')
plt.ylabel('Density')
plt.legend()
plt.show()

# Precision-Recall curve visualization
from sklearn.metrics import precision_recall_curve

y_scores = np.concatenate((scores_benign, scores_malware))
precision, recall, thresholds = precision_recall_curve(y_true_custom, -y_scores)

plt.figure(figsize=(12, 6))
plt.plot(thresholds, precision[:-1], 'b--', label='Precision')
plt.plot(thresholds, recall[:-1], 'g-', label='Recall')
plt.axvline(x=custom_threshold, color='black', linestyle='--', label='Custom Threshold')
plt.title('Precision-Recall Curve')
plt.xlabel('Threshold')
plt.ylabel('Score')
plt.legend()
plt.show()
```

Fig 5.17 - Visualization Code

The above code segment displays the decision scores and precision-recall curve for a One-Class SVM model used to identify benign and malicious APK files. It utilizes a user-defined threshold for classification. Below is a detailed explanation presented in the form of a paragraph:

Visualization of Decision Score Distribution: The initial graph illustrates the distribution of decision scores produced by the One-Class SVM model for benign (blue) and malicious (red) APK files. The decision scores, which represent the level of confidence with which the model categorizes each APK file as either benign or malicious, are displayed using a density plot that incorporates kernel density estimation (KDE). The x-axis displays the decision scores, which are numerical values used to evaluate APK files. The y-axis reflects the density or frequency of APK files that have a specific decision score.

- Blue Distribution (Benign): The density plot displays a clustering of higher decision scores towards the positive end, suggesting that the model exhibits greater certainty in categorizing these APK files as benign.

- The density map for malware APKs has lower judgment scores, suggesting that the model is more inclined to categorize these files as outliers or anomalies, namely malware.

The custom decision threshold set for categorization is indicated by the vertical dashed line at ``custom_threshold``. The threshold serves as a dividing point for the decision scores, categorizing scores below the threshold as malware and scores above or equal to it as benign.

The second figure displays the **precision-recall curve**, which represents the model's classification performance at various thresholds. The evaluation of the trade-off between precision and recall, which are measures that measure the model's capacity to accurately classify malware (recall) while limiting incorrect classifications (precision), relies heavily on this curve.

- Precision (Blue Line): The graph illustrates the relationship between precision, which is the ratio of genuine positives to all positive predictions, and different decision thresholds. Greater numbers imply a lower occurrence of false positives, which refers to innocuous files being mistakenly categorized as malware.

- The graph illustrates the relationship between recall (the ratio of successfully detected true positives) and thresholds. Greater values reflect the model's capacity to accurately detect a larger number of malware occurrences while minimizing the number of false negatives.

The vertical line, shown with black dashes, represents the selected ``custom_threshold`` for categorization. It demonstrates the position of this threshold on the precision-recall curve, emphasizing the influence of threshold alterations on the model's accuracy and recall performance.

Analysis: Distribution of Decision Scores: The graphic demonstrates a distinct distinction between benign and malware APKs, as determined by decision scores. In general, the model assigns higher scores to benign files and lower scores to malware. The density charts aid in seeing the model's level of certainty in its classifications.

The precision-recall curve allows for the evaluation of how the selected threshold impacts the model's capacity to accurately identify malware while decreasing the occurrence of false positives. The positioning of the `custom_threshold` on this curve establishes the equilibrium between accuracy and recall that is customized to meet unique application needs.

These visualizations offer valuable information about how the model behaves and can help in adjusting the decision threshold to improve the classification performance. This adjustment aims to strike a balance between accurately recognizing malware and minimizing the number of false positives for harmless files.

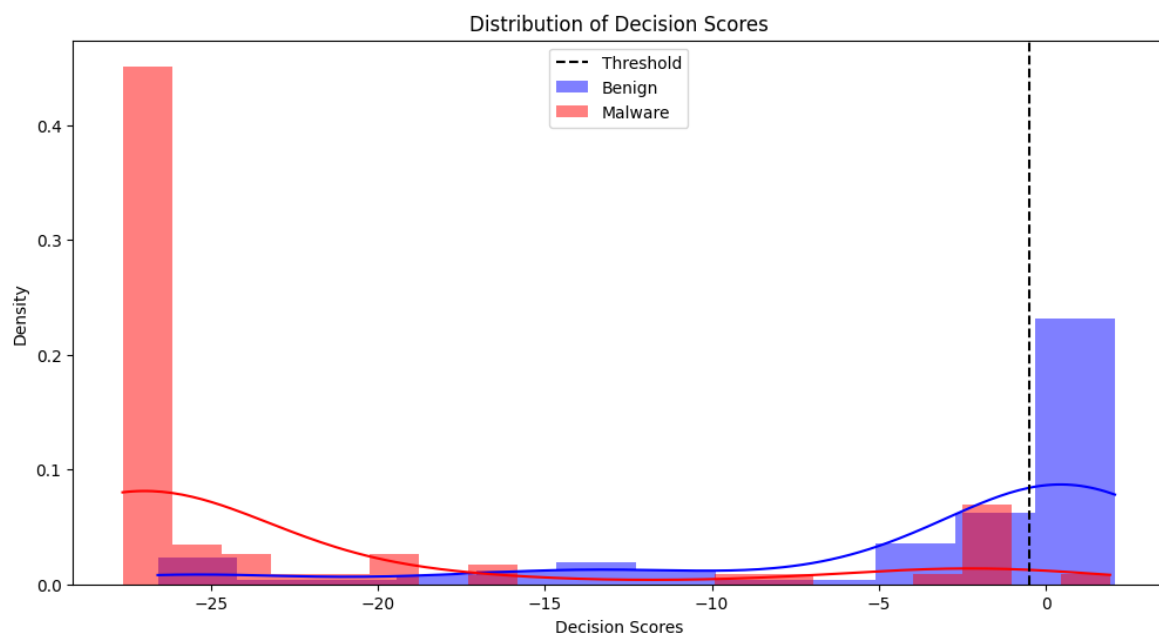


Fig 5.18 - Distribution of Decision Scores

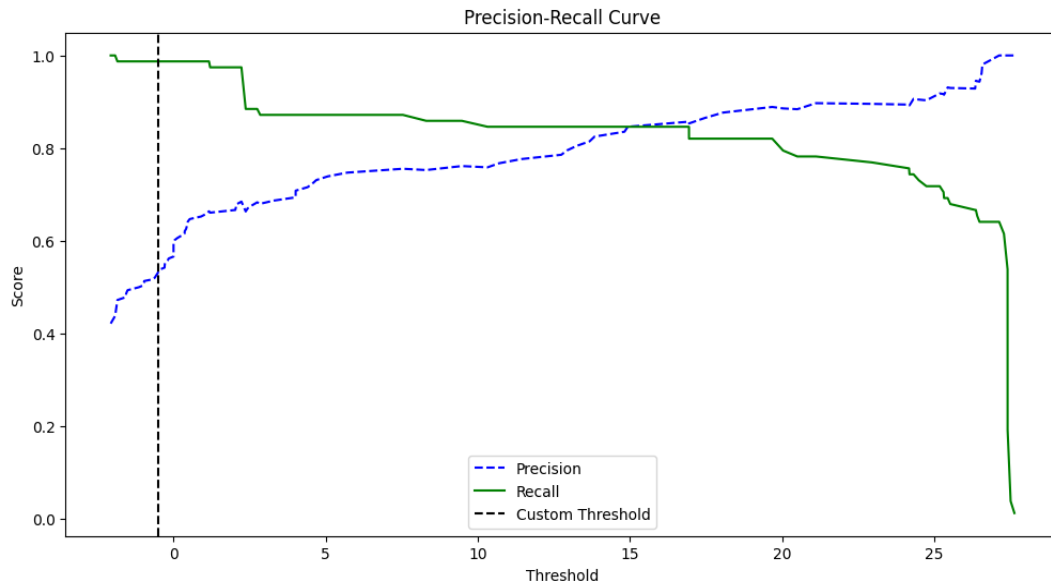


Fig 5.19 - Precision Recall Curve

Chapter 6: Evaluation of Findings

6.1 Performance Metrics Values:

F1 Score:

The F1 score is a metric that quantifies the performance of a model by taking the harmonic mean of its accuracy and recall. It provides a fair evaluation of the model's effectiveness. The scale spans from 0 to 1, with higher numbers indicating superior ability in both precision and recall.

Accuracy:

Accuracy is a metric that quantifies the degree of accuracy in a model's predictions. It is derived by dividing the number of properly predicted occurrences by the total number of examples in the dataset. Greater precision signifies superior overall performance.

Precision:

Precision is a metric that quantifies the ratio of accurate positive predictions to all positive predictions generated by the model. The main objective is to reduce the occurrence of false positives, which is particularly crucial in situations when incorrectly identifying harmless APKs as malware might result in significant consequences.

Recall:

Recall, also known as sensitivity, quantifies the ratio of successfully diagnosed malware instances to the total number of genuine positive instances. The main objective is to reduce the occurrence of false negatives, which is essential for detecting all cases of malware.

6.2 Comparison of Findings vs One-Class SVM

Table 6.1 - Comparison of Findings vs One-Class SVM

Algorithms	F1 Score	Accuracy	Precision	Recall	Mean
One-Class SVM	0.923	0.9333	1	0.857	0.857
K-Nearest Neighbor	0.92	0.93	0.95	0.95	-
CNN	0.9382	0.9332	0.9385	0.9379	-
K-means Clustering	-	0.9812	-	-	-
Decision Tree	0.92	0.92	0.94	0.89	0.0924
L-SVM	-	0.996	0.966	0.966	-

One-Class SVM model obtains an F1 score of 0.923, demonstrating great accuracy at 93.33%. It also achieves perfect precision at 1.0 and a recall of 0.857. This shows that it has a high level of precision in correctly recognizing instances of malware, while also keeping a decent balance of accurately detecting true malware. The great precision of the system guarantees that the majority of discovered malware instances are indeed harmful, resulting in a large reduction in false positives.

K-Nearest Neighbor (KNN) has a little lower F1 score (0.92) compared to One-Class SVM, but it has similar accuracy and precision, both at 0.95[23]. It performs exceptionally well in situations when the data exhibits distinct clusters and examples closely resemble their neighboring data points, enabling it to effectively identify both harmless and harmful APKs that are comparable to existing clusters.

The Convolutional Neural Network (CNN) obtains the highest F1 score (0.9382), demonstrating exceptional overall performance in terms of precision (0.9385) and recall (0.9379)[26]. It utilizes its capacity to extract hierarchical information from APK files, successfully collecting intricate patterns and subtle differences that differentiate malware from harmless apps.

K-means clustering algorithm. Clustering demonstrates its usefulness in grouping comparable APK files into clusters, achieving a high accuracy of 98.12%[25]. However, particular precision and recall metrics are not supplied. It enhances previous methods by detecting groups of possibly similar APK actions.

The Decision Tree model demonstrates an F1 score of 0.92, indicating a high level of accuracy. The precision and recall values are also balanced, with a precision of 0.94 and a recall of 0.89[24]. It is proficient at managing feature interactions and producing comprehensible rules for detecting malware, particularly in situations where the ability to explain the process is vital.

The L-SVM (Support Vector Machine) displays exceptional accuracy (99.6%) along with excellent precision (0.966) and recall (0.966)[22], showcasing its strong detection skills when enhanced with feature selection approaches.

In conclusion, each technique possesses distinct advantages that cater to the specific needs of detecting malware in APK files. The One-Class SVM algorithm is notable for its exceptional precision and well-balanced performance in identifying harmful occurrences while decreasing the occurrence of false positives. It is advantageous in situations when accuracy is crucial to prevent misidentifying harmless applications. CNN is highly proficient in identifying complex patterns, making it well-suited for thorough malware detection jobs. When selecting an algorithm for malware detection applications, it is important to take into account elements such as interpretability, computational efficiency, and the unique features of the dataset in order to obtain the best possible results.

Chapter 7: Conclusion

To summarize, this thesis has focused on the notable obstacles of bias and discrimination in machine learning models used for Android security, particularly in the areas of malware detection and analysis. The study process consisted of seven extensive chapters, starting with an introduction to the subject and followed by a thorough examination of the existing literature, with a specific focus on machine learning and anomaly detection approaches. The study subsequently chose and utilized One-Class Support Vector Machines (SVM) as the main approach.

The main goals of the study were to examine the utilization of machine learning and deep learning methods in Android security, create a model to differentiate between harmful and harmless applications, and assess the effectiveness of these algorithms in identifying Android malware. Extensive assessment has shown that One-Class SVM is highly effective in accurately detecting harmful threats in APK files.

This research provides significant insights and approaches for present applications and establishes a platform for future efforts in constructing fair and trustworthy machine learning models in the ever-changing digital environment. The results emphasize the significance of

mitigating biases in threat detection algorithms, which will facilitate the development of fair and resilient security mechanisms in Android applications, thereby improving user confidence and safety.

References:

1-European Commission, Joint Research Centre (JRC), Via E. Fermi 2749, I-21027 Ispra (Va), Italy.

<https://www.sciencedirect.com/science/article/pii/S2214212621000387>

2-Electronics 2021, 10(16), 1999; <https://doi.org/10.3390/electronics10161999>

3-Lilik Ariyanto, Supandi, Intan Indiati, Mina Tika Selviana, Widya Kusumaningsih.
<https://www.atlantis-pess.com/proceedings/icesre-19/125937254>

4-Ilkogretim Online - Elementary Education Online, 2021; 20 (1): pp. 1440-1450

<http://ilkogretim-online.org>-doi: 10.17051/ilkonline.2021.01.137

5-Djarot Hindarto, Pradita University & Handri Santoso Magister Technology Informatic, Pradita University. Dec 2021

<https://jcosine.if.unram.ac.id/index.php/jcosine/article/view/420>

6-An information theoretic approach to reducing algorithmic bias for machine learning Author links open overlay panel, Jin-Young Kim, Sung-Bae Cho

<https://www.sciencedirect.com/science/article/abs/pii/S0925231222005987>

7-Indiati, Intan & Supandi, Supandi & Ariyanto, Lilik & Kusumaningsih, Widya. (2021). The effectiveness of the problem-posing method based on android applications in mathematics learning. İlköğretim Online. 20. 1440-1450. 10.17051/ilkonline.2021.01.137.

https://www.researchgate.net/profile/Supandi-Supandi/publication/349003989_The_effectiveness_of_the_problem-posing_method_based_on_android_applications_in_mathematics_learning/links/601ab09592851c4ed5478d55/The-effectiveness-of-the-problem-posing-method-based-on-android-applications-in-mathematics-learning.pdf.

8-Aggarwal, K., Yadav, S.K. (2023). An Effectual Analytics and Approach for Avoidance of Malware in Android Using Deep Neural Networks. In: Shakya, S., Du, KL., Ntalianis, K. (eds) Sentiment Analysis and Deep Learning. Advances in Intelligent Systems and Computing, vol 1432. Springer, Singapore. https://doi.org/10.1007/978-981-19-5443-6_58

https://link.springer.com/chapter/10.1007/978-981-19-5443-6_58

9-Riasat, Rubata, et al. "Machine Learning Approach for Malware Detection by Using APKs." DEStech Transactions on Computer Science and Engineering cnsce (2017).

<https://www.academia.edu/download/67771804/8452.pdf>

10-Rana, Md Shohel, and Andrew H. Sung. "Malware analysis on Android using supervised machine learning techniques." International Journal of Computer and Communication Engineering 7.4 (2018): 178.

https://aquila.usm.edu/fac_pubs/15648/

11-Y C a, Padmanabha & Pulabaigari, Viswanath & B, Eswara. (2018). Semi-supervised learning: a brief review. International Journal of Engineering & Technology. 7. 81. 10.14419/ijet.v7i1.8.9977.

https://www.researchgate.net/profile/Eswara-B/publication/324050146_Semi-supervised_learning_a_brief_review/links/5b5c02f8458515c4b24e2b15/Semi-supervised-learning-a-brief-review.pdf

12-Omar, Salima & Ngadi, Md & Jebur, Hamid & Benqdara, Salima. (2013). Machine Learning Techniques for Anomaly Detection: An Overview. International Journal of Computer Applications. 79. 10.5120/13715-1478.

https://www.researchgate.net/profile/Salima-Benqdara/publication/325049804_Machine_Learning_Techniques_for_Anomaly_Detection_An_Overview/links/5af3569b4585157136c919d8/Machine-Learning-Techniques-for-Anomaly-Detection-An-Overview.pdf

13- Chomboon, Kittipong, et al. "An empirical study of distance metrics for k-nearest neighbor algorithm." Proceedings of the 3rd international conference on industrial application engineering. Vol. 2. 2015.

<https://pdfs.semanticscholar.org/815f/22363962afda432435cdbb478857547a17b5.pdf>

14-Mennatallah Amer, Markus Goldstein, and Slim Abdennadher. 2013. Enhancing one-class support vector machines for unsupervised anomaly detection. In Proceedings of the ACM SIGKDD Workshop on Outlier Detection and Description (ODD '13). Association for Computing Machinery, New York, NY, USA, 8–15.

<https://doi.org/10.1145/2500853.2500857>

<https://dl.acm.org/doi/abs/10.1145/2500853.2500857>

15-Y. Chen, J. Qian and V. Saligrama, "A new one-class SVM for anomaly detection," 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, BC, Canada, 2013, pp. 3567-3571, doi: 10.1109/ICASSP.2013.6638322.

<https://ieeexplore.ieee.org/abstract/document/6638322/>

16-E. Burnaev and D. Smolyakov, "One-Class SVM with Privileged Information and Its Application to Malware Detection," 2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW), Barcelona, Spain, 2016, pp. 273-280, doi: 10.1109/ICDMW.2016.0046.

<https://ieeexplore.ieee.org/abstract/document/7836677>

17-One-Class SVM for detecting Anomalies

<https://medium.com/grabngoinfo/one-class-svm-for-anomaly-detection-6c97fdd6d8af>

18-Maryamsadat Hejazi Hejazi_m2002@yahoo.com & Yashwant Prasad Singh (2013) ONE-CLASS SUPPORT VECTOR MACHINES APPROACH TO ANOMALY DETECTION, Applied Artificial Intelligence, 27:5, 351-366, DOI: [10.1080/08839514.2013.785791](https://doi.org/10.1080/08839514.2013.785791)

<https://www.tandfonline.com/doi/citedby/10.1080/08839514.2013.785791?scroll=top&needAccess=true>

19-Shrestha, Ajay, and Ausif Mahmood. "Review of deep learning algorithms and architectures." IEEE access 7 (2019): 53040-53065.

<https://ieeexplore.ieee.org/abstract/document/8694781/>

20-Akinduko, Ayodeji A., Evgeny M. Mirkes, and Alexander N. Gorban. "SOM: Stochastic initialization versus principal components." Information Sciences 364 (2016): 213-221

<https://www.sciencedirect.com/science/article/pii/S0020025515007318>

21-Chen, Ke. "Deep and modular neural networks." Springer Handbook of Computational Intelligence (2015): 473-494.

https://link.springer.com/chapter/10.1007/978-3-662-43505-2_28

22- Singh, A.K., Jaidhar, C.D. & Kumara, M.A.A. Experimental analysis of Android malware detection based on combinations of permissions and API-calls. J Comput Virol Hack Tech 15, 209–218 (2019). DOI:<https://doi.org/10.1007/s11416-019-00332-z>

23- Babbar, H.; Rani, S.; Sah, D.K.; AlQahtani, S.A.; Kashif Bashir, A. Detection of Android Malware in the Internet of Things through the K-Nearest Neighbor Algorithm. Sensors 2023, 23, 7256. <https://doi.org/10.3390/s23167256>

24-Utku, Anıl & Doğru, İbrahim & Akcayol, M.. (2018). Decision tree based android malware detection system. 1-4. 10.1109/SIU.2018.8404151.

25-Isredza Rahmi A Hamid et al 2017 IOP Conf. Ser.: Mater. Sci. Eng. 226 012105

DOI: 10.1088/1757-899X/226/1/012105

26- Andrii Nicheporuk[0000-0002-7230-9475], Oleg Savenko[0000-0002-4104-745X], Anastasiia Nicheporuk and Yuriy Nicheporuk Khmelnytsky National University, Khmelnytsky, Ukraine

<https://ceur-ws.org/Vol-2732/20200198.pdf>

27-Feizollah, Ali & Anuar, Nor & Salleh, Rosli. (2018). Evaluation of Network Traffic Analysis Using Fuzzy C-Means Clustering Algorithm in Mobile Malware Detection. Advanced Science Letters. 24. 929-932. 10.1166/asl.2018.10660.

Appendix:

2-TurnitIN Report:

Feedback Studio - Brave
ev.turnitin.com/app/canvas/en_us/?m=Manager_en_us&id=2297902947&student_user=18&u=1160342191

feedback studio Zeyad ElSharkawy Final year project

2741 Malicious APKs detection using machine learning

ZEYAD OSAMA WAHID ELSHARKAWY

1221300933

Match Overview

19%

Rank	Source	Similarity
1	www.coursera.com Internet Source	4%
2	docplayer.net Internet Source	2%
3	scholarworks.bridgepo... Internet Source	1%
4	sciencepubco.com Internet Source	1%
5	Evgeny Bumaev, Dmitry... Publication	1%
6	Jin-Young Kim, Sung-B... Publication	<1%
7	grabgoinfo.com Internet Source	<1%
8	Maryamsadat Hejazi, Y... Publication	<1%
9	link.springer.com Internet Source	<1%

Page: 1 of 125 Word Count: 16762 Text-Only Report High Resolution On

577 Sunny

Search

ENG 2/21/2024 1:41 PM