Turkish Online Journal of Qualitative Inquiry (TOJQI)
Volume 12, Issue 6, July, 2021: 10135 - 10147

Research Article

Android Malware Family Classification using Ensembling of Fpt and Fcm with Decision Tree

Raju Kumar Ranjan [a], Manoj Sethi [b]

[a] Department of Computer Science, Delhi Technological University (Formerly known as DCE)
Shahbad Daulatpur, Main Bawana Road, Delhi, 110042, India
Rajukumarranjan_2k18cse501@dtu.ac.in
[b] Department of Computer Science, Delhi Technological University (Formerly known as DCE)
Shahbad Daulatpur, Main Bawana Road, Delhi, 110042, India
manojsethi@dce.ac.in

Abstract

Android malware classification and assigning the appropriate android malware family is challenging. Traditional static analysis methods can easily be misguided by malware, and dynamic analysis consumes more space and time. This research proposed a fuzzy-based android malware family classification using multiple aspects of the DEX file. The considered aspects are Permissions of Android application, Image obtained from DEX file sectional features, Dalvik Opcode, and Bytecode of corresponding DEX file. The feature vectors acquired from these multiple aspects are fuzzified using a triangular fuzzifier. The obtained fuzzy sets are classified using an FPT classifier and clustered using Fuzzy C-means. FPT and FCM are combined according to the views, and a Decision Tree model is obtained for classifying the Android malware family. The final model produces an accuracy of up to 95.75%.

Keywords: Android malware; Fuzzy pattern tree; Fuzzy C-means; Family classification; Ensemble Learning; Dalvik opcode

1. Introduction

The android operating system evolved based on the Linux open-source kernel. Most of the smartphone and IoT devices are based on the android operating system. The extensive usage of the android operating system prepares a market for android application development and distribution. These android applications are distributed with the help of android application stores such as Google Play, Amazon App store, Xiaomi marketplace, etc. These are the official marketplace, but there are unofficial marketplaces for accessing and installing the android application. Some applications are also distributed directly. Official marketplace provides security tools for scanning and analyzing the maliciousness of published applications on their platforms like Google play store offers “Google Play Protect.” Applications installed directly from a third-party store, or direct source may act as malicious applications and harm the user. If the authenticity of the application is not confirmed, then the publisher of the fake application may also steal revenue that is entitled to the original developer. For the quick check of maliciousness of application, the user can apply to the platform like [VirusTotal, 2021], and [Koodous, 2021].

According to AV-TEST android malware statistics, as of March 2020, 10.5 million android malware get detected in 2019, and new Android malware samples are growing at the rate of 482,579 per month. This data shows that the android operating system is one of the most attractive targets for android malware developers. This malware consists of various variants of the same android family. Android malware is produced via the

piggybacking of the legitimate application with the repackaging of the malicious code. Hence, classifying the android malware based on their family is essential.

In the present scenario, the classification of android malware is based on static, dynamic, or hybrid methods. Android application package (APK) static features like permissions, registered receivers, execution code [Schmidt, et al., 2009], etc., are used to classify the android malware in traditional static methods. "DroidMoss" proposed in [Zhou, Zhou, Jiang, & Ning, 2012] calculates the fuzzy hash of bytecode after decomposing the DEX file of APK. Based on the calculated fuzzy hash, it is determined that the application is repackaged or not, android application maliciousness gets detected. In this API, calls are mainly considered for improving the efficacy of the system. [Jung, et al., 2018] taken the API as a feature set for classifying the samples using machine learning. [Ma, Ge, Liu, Zhao, & Ma, 2019] enhanced the android malware detection by adding frequency and characteristics based sequence of API calls. These methods have great accuracy for determining the maliciousness of android applications but are highly affected by confusion and reinforcement. Techniques in [Zarni Aung, 2013;Utku, Dogru, & Akcayol, 2018;Mahindru & Singh, 2017] consider android permission as a feature for detecting maliciousness. These are not affected by confusion and reinforcement, but accuracy is not achieved due to minor variations in permissions for the android application. [Martín, Hernández, Muñoz, & Guzmán, 2018] used multiple static features to improve accuracy. [Sahs & Khan, 2012] combined control flow diagram and [Zhu, et al., 2018] added sensitive API with permission feature. In general, static features may classify android malware but have limitations in order to achieve accuracy.

For overcoming the limitation of static analysis, dynamic analysis methods are proposed. [Bhatia & Kaushal, 2017] uses the system runtime API calls as a feature vector for classifying android malware. (Shabtai, Kanonov, Elovici, Glezer, & Weiss, 2012) proposed android malware classification framework "Andromaly." This framework monitor runtime features and events and uses machine learning algorithms for classification. For effectively detecting android malware, this framework needs sufficient time to collect events and runtime features. In [Touili & others, 2017;Fan, et al., 2017;Fan, et al., 2018] API call graph and frequent sub-graph are extracted and used as features for classification of android malware. [Arshad, et al., 2018] proposed a hybrid method by integrating the static and dynamic characteristics of android malware—the proposed method, ''SAMADroid'', consists of a three-layer detection model. The hybrid method effectively introduces the shortcoming of static and dynamic methods like a waste of space and time.

In recent years, image processing methodologies are being widely applied to detect and classify Android malware. Android application is converted in the form of images, and images are used to classify the android malware samples. File visualization technique for visualization of features is used because Android application is a packaged file, and all the logical data is stored in DEX file (classes.dex). This technique does not need to reverse engineer the DEX file for code analysis concerning other visualization techniques. The code analysis includes the knowledge about classes, variables, functions, API calls, etc. This method also can efficiently handle the large volume of Android malware samples.

The majority of the ML-based solutions are biased towards specific features (static, dynamic, and hybrid). The chances of failure increase when android malware mutates itself according to components involved in the target defense system. Therefore AI system based on multiple aspects is an optimal alternative for the android malware classification. The proposed method benefits the features generated on numerous aspects like permission, Dalvik opcode, Bytecode, and transformed images. These views are then used to create corresponding fuzzy (loosely defined) views. These fuzzy views are ensemble using the supervised Fuzzy Pattern tree (FPT) classifier and unsupervised Fuzzy C-means Clustering with Decision Tree Classifier. The proposed work achieves the accuracy of 95.7% for the classification of the Android malware family. The significant contribution in this research are:

- Multiple Views generations that are Permission View, Image View, Dalvik Opcode Frequency View, Dalvik Opcode TF-IDF View, Bytecode View, Bytecode TF-IDF View, from the Android APK file.
- Android DEX executable transformation into images based on its sectional structure.
- Transformation of crisp views to fuzzy views and train the FPT and FCM based models.
- Ensemble the FPT and FCM output dataset and training a Decision Tree for Android Malware Classification

The paper is organized as follows. First, this paper reviews the related work in Section 2. Then, in Section 3, the proposed methodology is discussed with all of its internals, including View Generation, Fuzzification of views, FPT classifier, FCM clustering of views, Ensemble both FPT and FCM, and Decision Tree Classifier. Next, in section 4, obtained results are analyzed to present a systematic comparison between single-view and

multi-view-based android malware classifiers with a discussion on the proposed approach. Finally, In Section 5, this paper concludes and illustrates the path for future research in android malware classification.

2. Related Work

[Liu, Du, Lei, & Liu, 2020] implemented the two-stage fuzzy strategy to classify the android malware family, which has polymorphic variants. They used regular expressions for identifying the callbacks to determine the behavior of Android malware. For classification 1-NN classifier and distance between the regular expression is considered. [Altaher & BaRukab, 2017] classified the android malware by an adaptive neuro-fuzzy inference system (ANFIS) with FCM. They generate the optimal number of clusters using FCM, which is used to develop an ANFIS classifier. The maximum achieved accuracy was 91% with considering their permission as a feature vector.

[Fang, Gao, Jing, & Zhang, 2020] transformed the DEX file into images based on their sectional structure and extract the texture, color moment, and string-based features for predicting the android malware family. The authors utilized the multiple kernel learning SVM algorithm for classification. [Arefkhani & Soryani, 2015] transformed the executables of various platforms to a grayscale image rather than extracting and analyzing the texture features. Different hash values Average Hash (AHash), Perception Hash (PHash), and Difference Hash (DHash) of the image were deliberated in the form of features. Neural networks(NN) are applied for classification malware. The proposed method deals with a high volume of malware samples but suffered from poor accuracy due to the loss of information in a grayscale image. [Fu, Xue, Wang, Liu, & Shan, 2018] transformed PE files into RGB images according to PE files structural architecture. These RGB images are fed to extract texture features via the GLCM algorithm and color features. Among various classification algorithms, Random forest (RF) exhibits 97.4% classification accuracy. Although the PE file structure and the APK file structure are quite different, this method cannot be implemented directly for classifying the Android malware family.

[Haddadpajouh, Azmoodeh, Dehghantanha, & Parizi, 2020] proposed the fuzzy consensus clustering-based model for attributing the Advanced Persistent Threat (APT). The authors generate multiple views based on the static and dynamic features, which are further merged and used to train a decision tree model. The achieved accuracy for attributing the APT group is 95.2%. [Dovom, et al., 2019] utilized the fuzzy pattern tree classifier for the classification of IoT malware. They used the opcodes as the feature vector to classify the android malware.

[Fan, et al., 2018] methodology Faldroid is based on constructing a frequent subgraph based on the Dalvik opcode sequence, which represents the typical behavior of android malware. The implemented method has experimented with 8407 malware with 36 families, and 94.2% is the maximum accuracy.

[Mercaldo & Saracino, 2018] presents an android malware classification based on fuzzy classification algorithms. Authors classified the 5000 android malware samples taken from the Drebin dataset in the classes as Botnet, Rootkit, SMS Trojan, Spyware, Installer, and Ransomware. The considered fuzzy classification algorithms are NN, OWANN, VQNN, FURIA, FuzzyRoughNN, FuzzyOwnershipNN, DiscernibilityClassifier, MultiObjectiveEvolutionaryClassifier.

3. Proposed Methodology

In the previous work, the authors did their research to find malicious behavior of android applications based on permissions, Dalvik opcodes, transformed images, DEX file's hashes, their Dalvik code flow graph and even used Fuzzy logic for classification.

The proposed methodology considers the multiple aspects of Android malware samples to generate feature vectors. These aspects are termed as views and are used to train FPT and FCM models individually. The developed fuzzy classifier model and clustering models are ensembles, and a novel feature vector is generated, which is further used to train the Decision Tree classifier to classify the android malware family. Figure 1 depicts the overall methodology in which firstly DEX file is extracted from the provided android APK, which is fed into the view generation module. The view generation module generates the six views based on multiple aspects: permission, transformed image, Dalvik opcode (frequency and tf-idf), and bytecode (frequency and tf-idf). Each view is used to train an FPT classifier and Clustered using the FCM model, and their results are combined and used as a feature vector to train the Decision Tree Classifier to classify the android malware family.

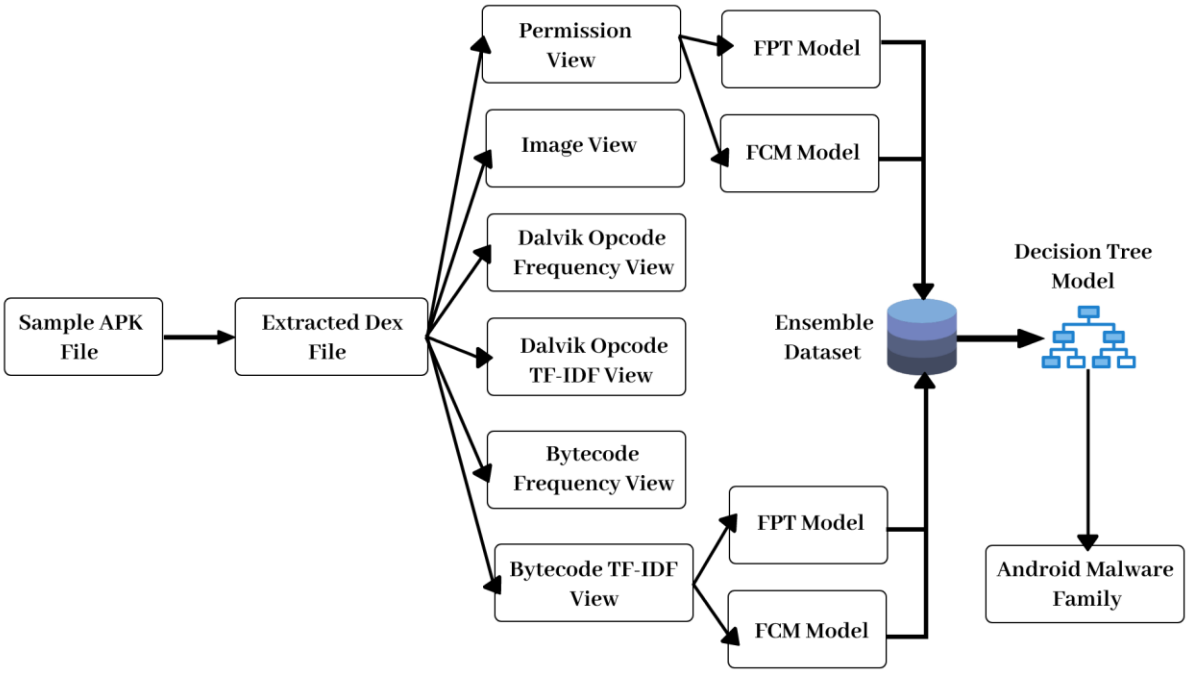

Figure 1 Overall Methodology for Classifying Android Malware Family

3.1 View Generation

3.1.1 Permission View

Android permission-based framework is developed to restrict unwanted applications to access critical information like SMS, call logs, and other vital and sensitive data stored on the device. Android malware gets these permissions by tricking a user into incorporating attack vectors. A dictionary of permissions is created for generating permission view, which contains the unique permission set within all android malware datasets used in this research. Each android malware is transformed to a vector-based on its permission list present in its manifest file. Dimension of the permission view vector is equal to the length of the dictionary and generated using equation 1.

$$\mathbf{Perm_{apk}} = \{\mathbf{x_i} = \mathbf{1 if D_{per}}[i] \in SampleAPK_{per}\} \qquad (1)$$

3.1.2 Image View

Android applications are compressed into APK files which are used for distribution. APK files are ZIP files such as JAR files which use Java libraries. APK file contains app code in the form of DEX file format, native libraries, resource files, configuration files, digital signatures, etc. Header, Index, and Data portion are the three portions of the DEX file. Figure 2 depicts snap of 010 Editor (hex editor) for representing the sections of DEX file. Basic information of the DEX file is present in the header section with the Index and offset values of other sections. The index portion of the DEX file consists size and offset of string index, proto Index, type index, method index, and field Index sections. The data portion consists size and offset of class definition sections and data sections. Therefore, in combining all these three portions, the DEX file is divided into eight sections.

Name	Value	Start	Size	Color	Comment
struct header_item dex_header		0h	70h	Fg: Bg:	Dex file header
struct dex_magic magic	dex 035	0h	8h	Fg: Bg:	Magic value
uint checksum	F7D48F72h	8h	4h	Fg: Bg:	Alder32 checksum of rest of file
SHA1 signature[20]	86E82336885A7...	Ch	14h	Fg: Bg:	SHA-1 signature of rest of file
uint file_size	3031556	20h	4h	Fg: Bg:	File size in bytes
uint header_size	112	24h	4h	Fg: Bg:	Header size in bytes
uint endian_tag	12345678h	28h	4h	Fg: Bg:	Endianness tag
uint link_size	0	2Ch	4h	Fg: Bg:	Size of link section
uint link_off	0	30h	4h	Fg: Bg:	File offset of link section
uint map_off	3031336	34h	4h	Fg: Bg:	File offset of map list
uint string_ids_size	25517	38h	4h	Fg: Bg:	Count of strings in the string ID list
uint string_ids_off	112	3Ch	4h	Fg: Bg:	File offset of string ID list
uint type_ids_size	3259	40h	4h	Fg: Bg:	Count of types in the type ID list
uint type_ids_off	102180	44h	4h	Fg: Bg:	File offset of type ID list
uint proto_ids_size	4851	48h	4h	Fg: Bg:	Count of items in the method prototype ID list
uint proto_ids_off	115216	4Ch	4h	Fg: Bg:	File offset of method prototype ID list
uint field_ids_size	19193	50h	4h	Fg: Bg:	Count of items in the field ID list
uint field_ids_off	173428	54h	4h	Fg: Bg:	File offset of field ID list
uint method_ids_size	24629	58h	4h	Fg: Bg:	Count of items in the method ID list
uint method_ids_off	326972	5Ch	4h	Fg: Bg:	File offset of method ID list
uint class_defs_size	2367	60h	4h	Fg: Bg:	Count of items in the class definitions list
uint class_defs_off	524004	64h	4h	Fg: Bg:	File offset of class definitions list
uint data_size	2431808	68h	4h	Fg: Bg:	Size of data section in bytes
uint data_off	599748	6Ch	4h	Fg: Bg:	File offset of data section

Figure 2 DEX file sections

For the transformation of the DEX file to RGB image size, entropy, bytecode, and proportions of sections are mapped to the size, red color channel, green color channel, and blue color channel of RGB image. Transformation and visualization are divided into five steps: Parsing of DEX file, Matrix Creation, Computation, Merging and Conversion.

DEX file Parsing: All eight sections of DEX file is parsed according to the DEX header as shown in Figure 2.

Matrix Creation: Each section of the DEX file is read byte and transformed into a byte matrix corresponding to each section. For deciding the width of matrix [Fang, Gao, Jing, & Zhang, 2020] proposed determining criteria according to DEX file size.

Computation: Calculation of the entropy matrix and the proportion matrix is done with the help of the bytecode matrix. Every section has a single value of entropy and proportion; hence these values remain the same for every section.

Entropy computation: The entropy represents the stability of the byte sequence. Therefore, the entropy of each section is calculated by using the equation 2.

$$Entropy = \sum_{i=0}^{255}[p(c_i)\log_2\ p(c_i)] \qquad (2)$$

Where c_i = frequency of byte i,

$p(c_i)$ = probability of frequency of byte i,

The value of $p(c_i)\backslash\log_2 p(c_i)$ is defined as 0 if the number of bytes is 0. Since range of entropy lie between [0, 8] and pixel value lie between [0, 255] hence entropy value was extended non-linearly. Value of R channel is calculated as shown in equation 3.

$$R = (Entropy^2 \bmod 8) \times 255/8 \qquad (3)$$

Proportion Computation: Every section of the DEX file has a different size because every section has many methods, variables, and classes. Therefore, the proportion of each section may vary. The Blue channel of RGB images is mapped with the proportion of each section. Equation 4 depicts the calculation formula for the proportion.

$$Proportion = (SectionSize)/(FileSize) \qquad (4)$$

The range of proportion is [0, 1]; hence it gets mapped with a range of pixel [0, 255] according to the equation 5.

$$B = Proportion \times 255 \qquad (5)$$

Merging and Conversion: After computation of R channel as entropy, G channel as byte code matrix, and B channel as proportion matrix is merged and RGB tuple matrix is obtained. This RGB tuple matrix is finally converted into the RGB image. Figure 3 illustrates the overall steps of transformation, and the GIST algorithm for feature extraction is applied as:

Step 1: Gabor filter bank shown in expression 6 is utilized to filter the grayscale Image. A Gaussian envelope modulates a sinusoidal plane with fixed direction and frequency in Gabor function with 2-D mode. Gabor filters are selective in terms of direction and frequency. Multiple groups of Gabor filters can be constructed by modifying the direction and frequency parameter.

$$g_{pq}(x,y) = a^{(-p)}g(x',y')(a>1)$$
$$x' = a^{(-p)}(x\cos\ \theta + y\sin\ \theta)$$
$$y' = a^{(-p)}(-x\cos\ \theta + y\sin\ \theta) \quad (6)$$
$$\theta = q\pi/(q+1)$$

where θ = direction of filter

$a^{(-p)}$ = scaling factor of wavelet expansion, $a=(U_h/U_l)^{(1/(p-1))}$ U_h and U_l are the lower and upper value of interest of frequencies.

Gabor filter bank shown in expression 6 is used for generating the f number of Gabor filters by modifying the value of p and q. So the number of the filter is f = p×q. During this research, p=4 and q=6 are considered, and a total of 32 Gabor kernels are generated for filtering the obtained grayscale image.

Step 2: A Gabor filter bank with f filter channels is used to convolute the gray image f(x, y) to obtain the f number of the filtered image. A grid of size $n_b \times n_b$ is obtained from the filtered image. The mean value of grids is considered as a feature vector. Hence a vector of length $f \times n_b \times n_b$ is obtained as a feature vector. During this research, the considered size of the grid is 4 × 4, and a total of the 512-dimensional feature vector as image view is generated.

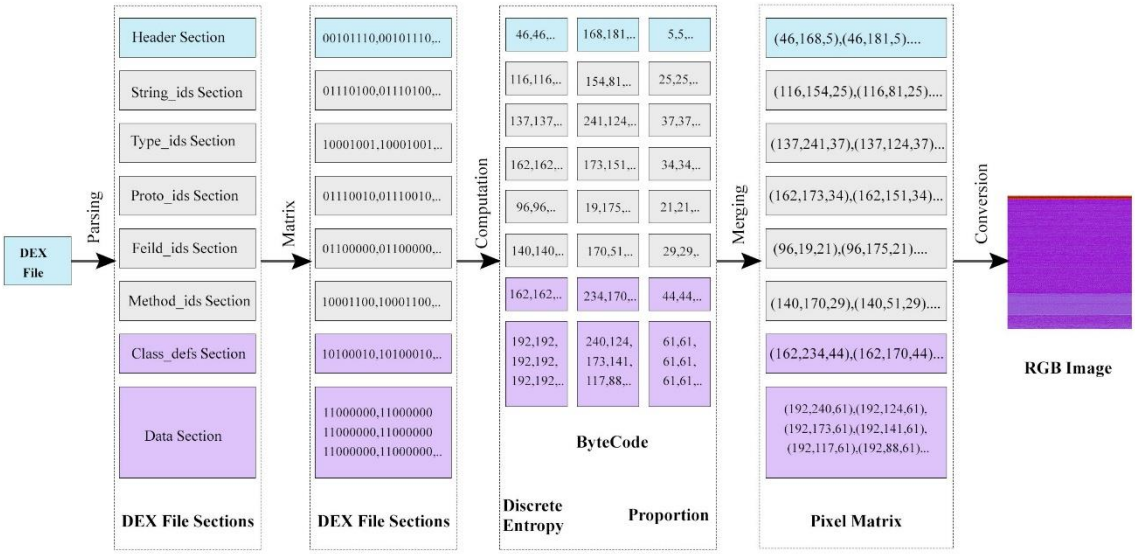

Figure 3 DEX to RGB transformation

3.1.3 Dalvik Opcode frequency View

Android APK file consists of a DEX file executable in Android Runtime Environment (ART). Dalvik opcodes are the intermediate opcodes that get executed in ART. These opcodes are responsible for achieving the objective of the Android Application. To consider it as a feature vector, firstly, a dictionary of Dalvik opcode is created. The dictionary is the unique Dalvik opcodes that are present in the complete dataset. The resulted dictionary is used as feature columns in the Dalvik opcode view.

Frequency view is the vector representation of the count of each opcode that is present in the dictionary in the respective Android Malware Sample. The vectorization of this view is done according to equation 7.

$$Opcodefrequency_{apk} = \{x_i = (Opcodecount_{apk}[i]/|SampleAPK_{op}|)\} \quad (7)$$

3.1.4 Dalvik Opcode TF-IDF View

TF-IDF value of a Dalvik opcode represents its relevance among all the datasets. Every DEX file is vectorized according to equation 8 to generate this view. Again the length of the resulted feature vector is equal to the length of the Dalvik opcode dictionary.

$$\begin{aligned} OpcodeTFIDF &= tf \times idf \\ tf &= Opcodefrequency_{apk} \\ idf &= \log\left[|TotalSampleAPK|/|SampleAPK_{op} \in SampleAPK|\right] \end{aligned} \quad (8)$$

3.1.5 Bytecode frequency and TF-IDF View

Since all the DEX is the sequence of bytecode hence the frequency and tf-idf views are generated similar to Dalvik opcode views. For this view, all possible Bytecode values are considered a dictionary of bytecode (Dict_ByteCode = 0,1,2 ……, 255). Hence the length of each vector is 256 in Bytecode view.

3.2 Fuzzification

Fuzzy logic represent the partial logic of truth or vagueness of reasoning. It takes the knowledge or data in a very similar way that our brain takes in. The transformation crisp values for the fuzzy inference engine (fuzzy set) is called fuzzification. Fuzzification is the assignment of membership function, which can represent the linguistic notion of a crisp set.

All the generated views represent the crisp value which must be fuzzified before implementing any fuzzy logic algorithm for classification and clustering. [Abbasbandy & Hajjari, 2009] proposed triangular and trapezoidal membership function μ(x) for fuzzifying the dataset. Some other membership functions are Gaussian, Generalized bell, and Sigmoid membership function.

In this research triangular fuzzifier is used to generate the fuzzy membership value μ(x).The definition of μ(x) is shown in equation 9 and depicted in Figure 4. In equation 9, a is the lower limit, b is the upper limit and c is the actual value. The considered value of a is $min(F) - [max(F) - min(F)]^2$, b is min(F) and c is max(F) where F refers the corresponding Fuzzy Set.

$$\mu(x) = \begin{cases} 0 & \text{if } x \le a \\ [(x-a)/(b-a)] & \text{if } a \le x \le b \\ [(c-x)/(c-b)] & \text{if } b \le x \le c \\ 0 & \text{if } x \ge c \end{cases} \quad (9)$$

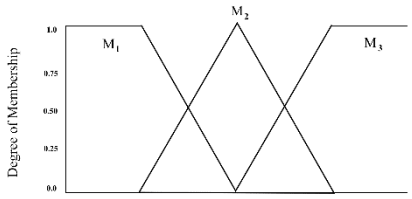

Figure 4 Degree of Membership

3.3 Fuzzy Pattern Tree Classifier

FPT [Senge & Hüllermeier, 2010] is a fuzzy-based classification algorithm introduced recently. This research considers the bottom-up approach for learning. It is a hierarchal tree-like structure whose inner nodes are associated with fuzzy-based logical and arithmetic operators. The most commonly used fuzzy operators in constructions of FPT are t-norms, t-conorms, weighted average (WA), and ordered weighted average (OWA) [Klement, Mesiar, & Pap, 2002;Schweizer & Sklar, 2011;Yager, 1988]. Input to the tree is provided at the leaf node. An instance of an FPT model is depicted in Figure 5. An FPT model is the collection of pattern trees, and each pattern tree is associated with a specific class. For classification, test data is given to the FPT model, and the highest score resulting pattern tree represents the predicted class.

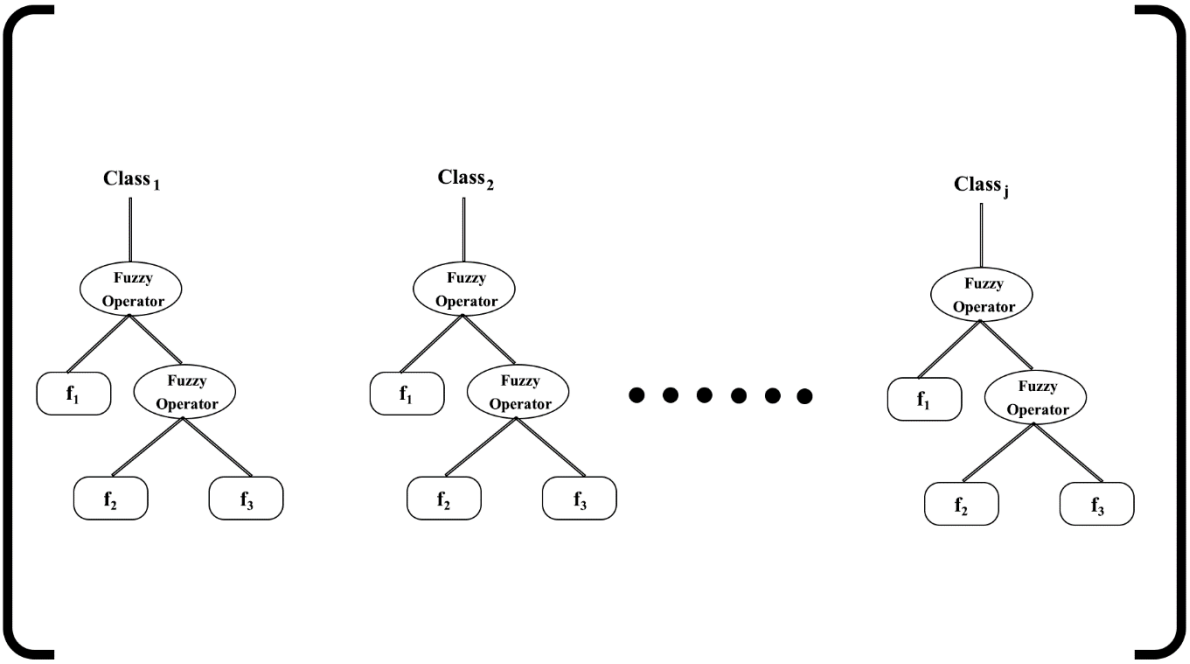

Figure 5 An instance of FPT model

Dataset $X(x_1, x_2, x_3 \ldots, x_i)$ corresponding to each view is the input for algorithm 1. Initially, three sets of basic partition tree PT, C^0 candidate tree, and M^* FPT model are initialized. PT is the collection of primitive fuzzy pattern trees F_{ij} where i represents the feature vector, and j refers to the corresponding class. A candidate pattern tree C^0 is initialized based on the root mean square error(RMSE) as a loss function. C^0 is the collection of N best trees among the PT, i.e., C^0 is a subset of PT. M^* contains the N best trees of C^0 as the initial FPT model, which get tuned and improved as the final FPT model. Now iteratively, each pattern tree from M* is taken and optimized with fuzzy operator and trees from PT as mentioned in algorithm 2. This research considers the maximum iteration as five, and in every iteration loss function, RMSE is used to optimize M^*. Finally, the M^* model is used to predict the confidence value corresponding to each class.

Algorithm 1 Fuzzy Pattern Tree Classifier

1. $\boldsymbol{PT = \{F_{i,j} | \forall i \in inputfeatures, and \forall j \in classes\}}$
2. $\boldsymbol{S = PT}$
3. $\boldsymbol{C^0 = argmin_{F \in PT}^{NB}[\sum_{(x,y)\in T} RMSE(y, F(x))]}$
4. $\boldsymbol{M^* = argmin[error(C^0)]}$
5. $\boldsymbol{t_{max} = 5}$
6. $\boldsymbol{t = 0}$
7. $\boldsymbol{MaximumDepth = 10}$
8. **while**$\boldsymbol{(t \leq t_{max})}$
9. $\boldsymbol{t = t + 1}$
10. $\boldsymbol{C^t = C^{t+1}}$
11. **forall** $\boldsymbol{L \in leafs(M^*)}$
12. **if**$\boldsymbol{(Depth(L) \leq MaximumDepth)}$
13. **forall** $\boldsymbol{\psi \in FuzzyOperators}$
14. **forall** $\boldsymbol{p \in PT}$
15. $\boldsymbol{C^t = C^{t-1} \cup replaceleaf(M^*, L, \psi, p)}$
16. **endfor**
17. **endfor**
18. **endif**
19. **endfor**

20. M^* $= argmin_{F \in C^t}^{n}[\sum_{(x,y)\in T} RMSE(y, F(x))]$
21. **end while**
22. **return** M^*

3.4 Fuzzy C-means Clustering

There are mainly two types of clustering: Hard Clustering and Soft Clustering. In hard clustering, each data point belongs to a particular fixed cluster number. In soft clustering, instead of defining a specific cluster number, a probability is defined for each data point regarding each cluster number.

To generate fuzzy clusters, this research implements the Fuzzy C-means Clustering [Pal & Bezdek, 1995]. The technique is mentioned in algorithm 2. In FCM, C refers to the maximum cluster number, which is taken 200, and fuzziness parameter m is taken 1.75. The training set of each view is clustered, and a trained partition matrix U is returned corresponding to each view. Since each row refers to the probability distribution of data points, the sum of each row of the partition matrix should always be one.

Algorithm 2 Fuzzy C Means Clustering

1. $C = 200$
2. $m = 1.75$
3. $U^* = [u_{ij}]matrix, U^0$
4. $k = 0$
5. **repeat**
6. $C^{(k)} = [\sum_{i=1}^{N} u_{ij}^m x_i / \sum_{i=1}^{N} u_{ij}^m]$
7. $U^{(k+1)} = [\sum_{k=1}^{c} (x_i - c_j / x_i - c_k)^{2/m-1}]^{-1}$
8. **until** $|U^{(k+1)}| - |U^{(k)}| \leq StoppingCriteria$
9. **return** U

3.5 Ensemble FPT and FCM vectors with Decision Tree

The trained models of FPT and FCM are used to generate a new dataset on each view. The dataset of each view is fitted on FPT and FCM models for producing the new dataset as mentioned in algorithm 3. Now the aggregated dataset X is combined with the corresponding class Y. Here Y represents the actual class of the android malware family.

This module of the proposed methodology is the final decision maker on the aggregated dataset of FPT and FCM models. [X, Y] the new aggregated dataset is used to train a decision tree model that decides the Android malware family class. The optimal depth of the decision tree is considered ten during the research, which is obtained by the GridCVSearch method.

Algorithm 3 Ensemble FPT and FCM Models

1 $X_{FPT} = FPT_{view}(X_{view})$

2 $X_{FCM} = FCM_{view}(X_{view})$

3 $U^* = [u_{ij}]matrix, U^0$

4 ***return X***

4. Results and Discussion

4.1 Dataset

The samples utilized in this research are based on the RmvDroid Android malware dataset. [Wang, Si, Li, & Guo, 2019] collects the android applications of Google play in 4 years and then uses Virus Total to label the application. They also observe the removal of the application from the play store to label the application.

This dataset contains 9,133 android malware samples that belong to 56 families. These samples are randomly chosen from the dataset and used to generate the respective views. After analyzing the dataset, it may confer that it contains the imbalance number of malware samples in their families. Hence for getting the effective results, 11 families and 150 samples per family are taken as shown in Table 1.

S No.	Family name	Sample Count
1.	AIRPUSH	2872
2.	MECOR	993
3.	PLANKTON	8022
4.	ADWO	690
5.	YOUMI	597
6.	GAPPUSIN	441
7.	MOBIDASH	344
8.	VISER	291
9.	DOWGIN	279
10.	LEADBOLT	179
11.	KUGUO	168

Table 1 Android Malware Family Dataset

4.2 Performance Evaluation

The discussed methodology is used to classify the Android malware families. The dataset of RmvDroid is fed into the framework, and the following six different feature vector is generated based the multiple views. These views are based on permission, image, Dalvik opcode, and DEX bytecode. Every single view is firstly used to train, and Fuzzy based FPT classifier. There are six different FPT classifiers based on each view.

Similarly, these views are clustered based on the soft clustering technique Fuzzy C-means. The clustering algorithm considers the number of clusters c=200 and fuzziness parameter m=1.75. A partition matrix based on each view is generated as a result. These FPT and FCM models are aggregated and fed into a Decision Tree with its actual class, which yields a trained model with an accuracy of 95.75\%.

The performance of machine learning algorithms is evaluated based on four core metrics True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). Based on these metrics, precision, recall, and F1 Score is calculated to assess the proposed methodology. The confusion matrix visualizes the performance of any machine learning algorithm. The predicted class is represented by the row of the confusion matrix and the actual class by column. The accuracy of the trained model for each Android malware family is depicted through diagonal cells.

View	F1 Score	Accuracy	Precision	Recall
Permission View	0.748592	0.690205	0.690205	0.921700
Image View	0.443169	0.492027	0.492027	0.445675
Count_Opcode View	0.602364	0.665148	0.665148	0.628156
TF-IDF_Opcode View	0.708458	0.724374	0.724374	0.737401
Count_Bytecode View	0.429397	0.448747	0.448747	0.508666
TF-IDF_Bytecode View	0.517084	0.517084	0.4779082	0.51708
Combined View	0.957495	0.957554	0.957764	0.957554

Table 2 Performance Metrics

The performance metrics of every single view are represented in Table 2 and compared with the performance metrics of the combined view. The results obtained based on a single view need a significant improvement to classify the android malware family. In order to achieve this improvement, when these views are aggregated, then the performance metrics improved significantly. Combined view F1 Score is achieved up to 95.74%. Figure 6 represents the confusion matrix of the integrated view. Finally, it can be justified that classifying the android malware family based on multiple views and combining fuzzy logic algorithms for decision trees yield classification accuracy up to 95.74\%. The consideration of multiple aspects ensures that any single view cannot perform, balanced by the other views.

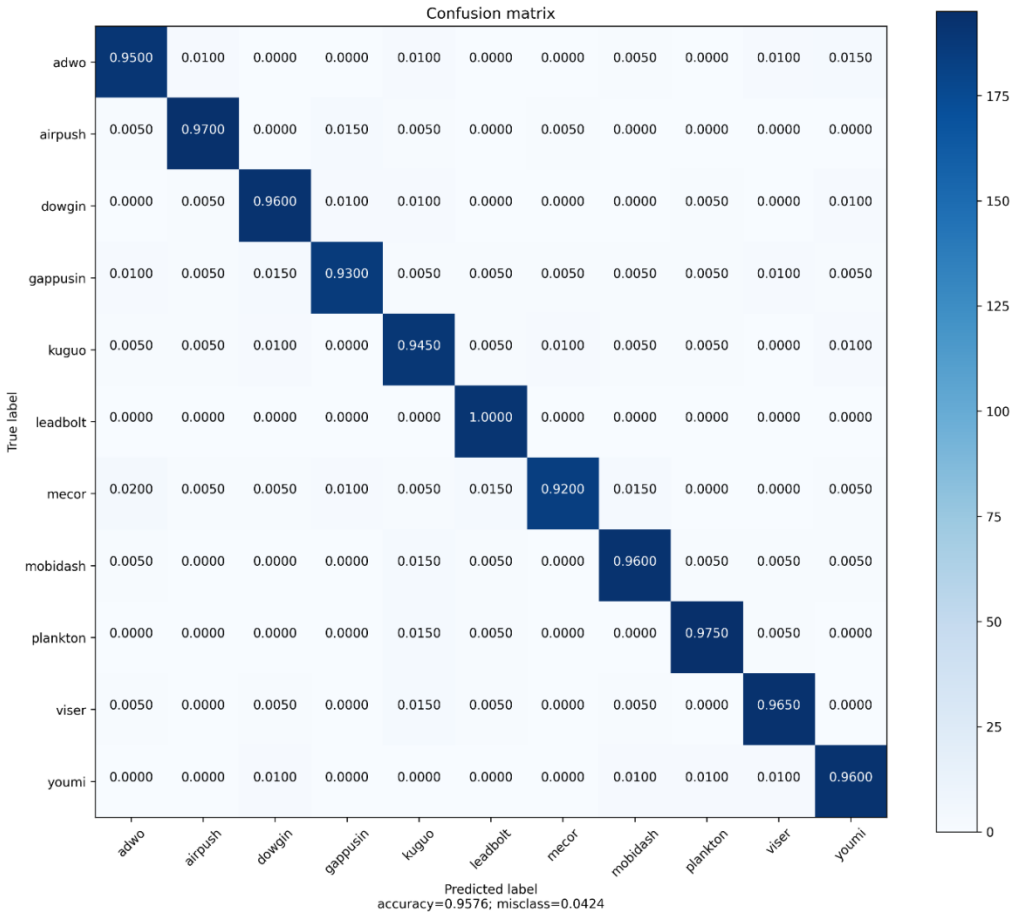

Figure 6 Confusion Matrix on Combined View

4.3 Discussion

This section discusses the methodology adopted in this paper with similar work done for the Android malware family classification. [Fang, Gao, Jing, & Zhang, 2020] methodology is based on computer vision domain. The authors transformed the DEX file into images and applied an SVM multi-kernel for the classification of Android malware classes. [Haddadpajouh, Azmoodeh, Dehghantanha, & Parizi, 2020]

mechanism is based on multiple views for attributing the Advanced persistent threat payloads and achieved accuracy is 95.2\% on five APT groups. [Dovom, et al., 2019] utilizes FPT for classifying IoT malware as 0/1 classification. [Mercaldo & Saracino, 2018] done android malware classification based on fuzzy classification algorithms into the following classes: Botnet, Rootkit, SMS Trojan, Spyware, Installer, and Ransomware. The proposed methodology uses the concept of ensemble learning and implements it on Android Malware Family classification. The methodology in this research is based on multiple views of Android applications, which includes permission, image, Dalvik Opcode, and Bytecode. These views adopt ensemble learning of fuzzy-based classification and clustering algorithms to generate the input for the Decision Tree.

5. Conclusion and Future Work

Android malware family classification is one of the most demanding tasks in the android malware threat domain. This research is done based on multiple aspects of an Android application which is termed as view. The views considered for attaining the classification problems are permission, image representation of DEX file, underlying Dalvik opcodes, and bytecode of the DEX file. These multiple views are vectorized and fuzzified using triangular fuzzification. The fuzzified vectors are used to train the FPT classifier and soft-clustered using the FCM algorithm. Finally, the FPT and FCM vectors are combined, and a decision tree model is used to final classification of android malware families. From the future point, the views considered during this research are based on static analysis, but also these views may be based on dynamic analysis.

References

[1] H.-J. Zhu, Z.-H. You, Z.-X. Zhu, W.-L. Shi, X. Chen and L. Cheng, "DroidDet: effective and robust detection of android malware using static analysis along with rotation forest model," Neurocomputing, vol. 272, p. 638–646, 2018.

[2] W. Zhou, Y. Zhou, X. Jiang and P. Ning, "Detecting repackaged smartphone applications in third-party android marketplaces," in Proceedings of the second ACM conference on Data and Application Security and Privacy, 2012.

[3] W. Z. Zarni Aung, "Permission-based android malware detection," International Journal of Scientific & Technology Research, vol. 2, p. 228–234, 2013.

[4] R. R. Yager, "On ordered weighted averaging aggregation operators in multicriteria decisionmaking," IEEE Transactions on systems, Man, and Cybernetics, vol. 18, p. 183–190, 1988.

[5] H. Wang, J. Si, H. Li and Y. Guo, "Rmvdroid: towards a reliable android malware dataset with app metadata," in 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), 2019.

[6] VirusTotal, Virus Total, Virus Total, 2021.

[7] A. Utku, I. A. Dogru and M. A. Akcayol, "Permission based android malware detection with multilayer perceptron," in 2018 26th Signal Processing and Communications Applications Conference (SIU), 2018.

[8] T. Touili and others, "Extracting Android malicious behaviors," in International Workshop on FORmal methods for Security Engineering, 2017.

[9] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer and Y. Weiss, ""Andromaly": a behavioral malware detection framework for android devices," Journal of Intelligent Information Systems, vol. 38, p. 161–190, 2012.

[10] R. Senge and E. Hüllermeier, "Top-down induction of fuzzy pattern trees," IEEE Transactions on Fuzzy Systems, vol. 19, p. 241–252, 2010.

[11] B. Schweizer and A. Sklar, Probabilistic metric spaces, Courier Corporation, 2011.

[12] A.-D. Schmidt, R. Bye, H.-G. Schmidt, J. Clausen, O. Kiraz, K. A. Yuksel, S. A. Camtepe and S. Albayrak, "Static analysis of executables for collaborative malware detection on android," in 2009 IEEE International Conference on Communications, 2009.

[13] J. Sahs and L. Khan, "A machine learning approach to android malware detection," in 2012 European Intelligence and Security Informatics Conference, 2012.

[14] N. R. Pal and J. C. Bezdek, "On cluster validity for the fuzzy c-means model," IEEE Transactions on Fuzzy systems, vol. 3, p. 370–379, 1995.

[15] F. Mercaldo and A. Saracino, "Not so Crisp, Malware! Fuzzy Classification of Android Malware Classes," in 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), 2018.

[16] I. Martín, J. A. Hernández, A. Muñoz and A. Guzmán, "Android malware characterization using metadata and machine learning techniques," Security and Communication Networks, vol. 2018, 2018.

[17] A. Mahindru and P. Singh, "Dynamic permissions based android malware detection using machine learning techniques," in Proceedings of the 10th innovations in software engineering conference, 2017.

[18] Z. Ma, H. Ge, Y. Liu, M. Zhao and J. Ma, "A combination method for android malware detection based on control flow graphs and machine learning algorithms," IEEE access, vol. 7, p. 21235–21245, 2019.

[19] X. Liu, X. Du, Q. Lei and K. Liu, "Multifamily Classification of Android Malware With a Fuzzy Strategy to Resist Polymorphic Familial Variants," IEEE Access, vol. 8, p. 156900–156914, 2020.

[20] Koodous, Koodous, Koodous, 2021.

[21] E. P. Klement, R. Mesiar and E. Pap, "On the order of triangular norms—comments on “A triangular norm hierarchy” by E. Cretu," Fuzzy Sets and Systems, vol. 131, p. 409–413, 2002.

[22] J. Jung, H. Kim, D. Shin, M. Lee, H. Lee, S.-j. Cho and K. Suh, "Android malware detection based on useful API calls and machine learning," in 2018 IEEE First International Conference on Artificial Intelligence and Knowledge Engineering (AIKE), 2018.

[23] C. Herzog, V. V. T. Tong, P. Wilke, A. van Straaten and J.-L. Lanet, "Evasive Windows Malware: Impact on Antiviruses and Possible Countermeasures," arXiv preprint arXiv:2009.12204, 2020.

[24] A. M. Hay, "The derivation of global estimates from a confusion matrix," International Journal of Remote Sensing, vol. 9, p. 1395–1398, 1988.

[25] H. Haddadpajouh, A. Azmoodeh, A. Dehghantanha and R. M. Parizi, "Mvfcc: A multi-view fuzzy consensus clustering model for malware threat attribution," IEEE Access, vol. 8, p. 139188–139198, 2020.

[26] J. Fu, J. Xue, Y. Wang, Z. Liu and C. Shan, "Malware visualization for fine-grained classification," IEEE Access, vol. 6, p. 14510–14523, 2018.

[27] Y. Fang, Y. Gao, F. Jing and L. Zhang, "Android malware familial classification based on DEX file section features," IEEE Access, vol. 8, p. 10614–10627, 2020.

[28] M. Fan, J. Liu, W. Wang, H. Li, Z. Tian and T. Liu, "Dapasa: detecting android piggybacked apps through sensitive subgraph analysis," IEEE Transactions on Information Forensics and Security, vol. 12, p. 1772–1785, 2017.

[29] M. Fan, J. Liu, X. Luo, K. Chen, Z. Tian, Q. Zheng and T. Liu, "Android malware familial classification and representative sample selection via frequent subgraph analysis," IEEE Transactions on Information Forensics and Security, vol. 13, p. 1890–1905, 2018.

[30] E. M. Dovom, A. Azmoodeh, A. Dehghantanha, D. E. Newton, R. M. Parizi and H. Karimipour, "Fuzzy pattern tree for edge malware detection and categorization in IoT," Journal of Systems Architecture, vol. 97, p. 1–7, 2019.

[31] T. Bhatia and R. Kaushal, "Malware detection in android based on dynamic analysis," in 2017 International Conference on Cyber Security And Protection Of Digital Services (Cyber Security), 2017.

[32] S. Arshad, M. A. Shah, A. Wahid, A. Mehmood, H. Song and H. Yu, "Samadroid: a novel 3-level hybrid malware detection model for android operating system," IEEE Access, vol. 6, p. 4321–4339, 2018.

[33] M. Arefkhani and M. Soryani, "Malware clustering using image processing hashes," in 2015 9th Iranian Conference on Machine Vision and Image Processing (MVIP), 2015.

[34] A. Altaher and O. BaRukab, "Android malware classification based on ANFIS with fuzzy c-means clustering using significant application permissions," Turkish Journal of Electrical Engineering & Computer Sciences, vol. 25, p. 2232–2242, 2017.

[35] S. Abbasbandy and T. Hajjari, "A new approach for ranking of trapezoidal fuzzy numbers," Computers & Mathematics with Applications, vol. 57, p. 413–419, 2009.