applied sciences

MDPI

Article

An Ensemble Approach Based on Fuzzy Logic Using Machine Learning Classifiers for Android Malware Detection

İsmail Atacak

Department of Computer Engineering, Faculty of Technology, Gazi University, Ankara 06560, Turkey; iatacak@hotmail.com

Abstract: In this study, a fuzzy logic-based dynamic ensemble (FL-BDE) model was proposed to detect malware exposed to the Android operating system. The FL-BDE model contains a structure that combines both the processing power of machine learning (ML)-based methods and the decision-making power of the Mamdani-type fuzzy inference system (FIS). In this structure, six different methods, namely, logistic regression (LR), Bayes point machine (BPM), boosted decision tree (BDT), neural network (NN), decision forest (DF) and support vector machine (SVM) were used as ML-based methods to benefit from their scores. However, through an approach involving the process of voting and routing, the scores of only three ML-based methods which were more successful in classifying either the negative instances or positive instances were sent to the FIS to be combined. During the combining process, the FIS processed the incoming inputs and determined the malicious application score. Experimental studies were performed by applying the FL-BDE model and ML-based methods to the balanced dataset obtained from the APK files downloaded in the Drebin database and Google Play Store. The obtained results showed us that the FL-BDE model had a much better performance than the ML-based models did, with an accuracy of 0.9933, a recall of 1.00, a specificity of 0.9867, a precision of 0.9868, and an F-measure of 0.9934. These results also proved that the proposed model can be used as a more competitive and powerful malware detection model compared to those of similar studies in the literature.

Keywords: mobile security; malware detection; machine learning; fuzzy logic; ensemble learning

Citation: Atacak, İ. An Ensemble Approach Based on Fuzzy Logic Using Machine Learning Classifiers for Android Malware Detection. *Appl. Sci.* **2023**, *13*, 1484. https://doi.org/10.3390/app13031484

Academic Editors: Jerry Chun-Wei Lin, Stefania Tomasiello and Gautam Srivastava

Received: 25 December 2022
Revised: 14 January 2023
Accepted: 19 January 2023
Published: 23 January 2023

1. Introduction

The impact of smartphones in our daily life is growing rapidly. It is obvious that these devices' popularity and usage rates have significantly risen because of the wide range of functions they can provide us. As of 2022, Statista [1] estimates that there will be 6.56 million mobile phone users worldwide. In fact, UN statisticians have [2] predicted that by October 2022, among a population of 8 billion, approximately 82% of the total population in the world own a smartphone. Android is the most widely used smartphone operating system, with a 71.85% market share as of July 2022, according to Statista statistics [3]. This results from Android's open-source nature and other characteristics.

Given its open-source nature, it is inevitable that an operating system with such widespread use would attract malware [4]. Due to these circumstances, mobile malware has emerged. The term "malware" describes software applications that have been installed on a device with a variety of purposes, such as to annoy users, steal personal user data, consume system resources without permission, or harm the device [5]. After being installed on the target device, these programs operate to fulfill their intended functions, either visibly or invisibly to the user, in the background [6]. During the first two quarters of 2021, more than 2.2 million new instances of mobile malware were reported, according to Statista [7]. Given all of these facts, it is obvious that Android malware represents a significant threat to mobile systems.

Today, it is widely known that smartphones contain personal data about users, including their banking details, confidential information, and health data. Therefore, it is evident that both the security and integrity of these sensitive data will be compromised if these smartphones are exposed to a malware. When malware is planted on a smartphone, it can acquire the user's bank account if the user is using online banking apps, as well as their health records and private, sensitive data such as personal information. Using the Android smartphone, the malware can send emails and SMS messages without the user's consent to spread the malware to even more users. It is inevitable that malware developers will target smartphones more frequently considering all of these capabilities. The fact that the frequency of mobile malware rose by approximately 1800% in 2016 is among the most obvious and concrete indicator of this problem [8].

These issues have led to a wide range of models being proposed for malware detection on Android mobile devices. In the past, methods based on signatures have emerged. These methods search the application code to detect known patterns of malicious code. Although signature-based approaches are currently effective in terms of their performance and in detecting malware that has already been identified, they cannot be used to detect new types of malware, and as a result, more modern techniques and models have replaced them. These models can be examined in three categories:

- Static malware detection methods.
- Dynamic malware detection methods.
- Hybrid malware detection methods.

All three of the aforementioned methods are frequently applied using machine-learning-based techniques. The ability of machine-learning-based approaches to detect previously unknown malware [9] makes them vital. Because static detection techniques evaluate software codes without running suspicious applications, they are unable to detect malicious programs that obfuscate the program code or employ encryption [10] to protect the malware. On the other hand, approaches based on dynamic detection methods run the application in a secure environment to search for any malicious behavior [11]. However, since the performance cost of the application rises in dynamic detection-based methods, it becomes challenging to perform a real-time analysis due to the extension of the execution time [12]. Both strategies have advantages and disadvantages in comparison with each other. Although static approaches offer a thorough analysis of the code, it can be challenging to detect changes in the application code that might take place while the application is running. On the other hand, the time and performance costs for dynamic approaches are quite high. Hybrid methods have been developed to overcome the disadvantages of both of these methods.

The FL-BDE model was developed in this study as a strong and effective malware detection tool to reduce the negative effects of malware on Android operating systems. The proposed model provides us with a structure that interprets and combines the scores of ML-based NN, SVM, DF, LR, BPM and BDT methods in two separate groups of three according to whether the incoming data are voted as benign or malicious applications. The contributions of this study to the literature are summarized as follows:

- To develop a strong and dynamic model, the results of ML-based approaches with high classification performance are integrated over the FIS, which has an effective inference capacity.
- Depending on whether the data obtained through the voting and routing process were voted as benign or malicious, our model combines the results of only three ML-based algorithms. While this allows us to increase the performance of our model by taking advantage of the scoring power of the six methods, it also simplifies the design of the FIS model by reducing the number of inputs.
- The model proposed in this study has outperformed comparable studies in the literature, and as a result, it has revealed that it has a strong and dynamic structure.

The rest of the study is organized as follows. A brief overview of recent research on the subject is provided in Section 2. The dataset, the proposed model and its features, the ML-based methods and the performance metrics to be employed in the assessment of experiments are described in Section 3. The outcomes of the experiments are discussed along with the literature in the "Results and Discussion" section. The last section concludes, along with suggestions for further research on the topic.

2. Literature Review

Malware is defined as software that was created with malicious intent and serves this function. The widespread use of mobile devices and the increasing number of users have caused malware developers to focus their attention on this area [11]. Malware can have a variety of purposes. Some of these malicious purposes can be categorized as disrupting the normal operation of the Android operating system it is working on, obtaining the user's personal and sensitive information illicitly and without the user's consent, seizing the user's device, obtaining information for ransom, or displaying unwanted advertising content to the user. To stop malware developers from achieving these goals, a lot of research has been carried out in the field of malware detection. The three main categories of studies conducted in this context are static analysis, dynamic analysis, and hybrid analysis.

Static analysis techniques examine an application's source code without running it [13] to detect malicious behavior in the suspicious software. Predominantly, these methods use either permission-based [14] or signature-based [15] techniques. Malware detection in signature-based approaches is carried out by comparing the application codes to the codes in a database that contain previously known malicious code fragments. However, in permission-based approaches, the permissions requested by the application are compared with the permissions frequently requested by the malicious applications. Although static analysis methods are faster than dynamic analysis methods are [16], they are inefficient at providing information about the behavior of the application while they are running because they are unable to detect the application code that the application will load dynamically during execution [17].

On the other hand, dynamic analysis techniques involve running the program in a secure, isolated environment and analyzing its behavior to determine if it is malicious or not [18]. Dynamic analysis methods detect malware by examining features such as system calls [19] and network traffic [20] that occur while the application is running.

Martín et al. [21] created a model utilizing a combination of static analysis and dynamic analysis, employing both a Random Forest classifier and a Bagging classifier. With this fusion model, researchers have achieved an accuracy of 89.7% and a precision of 89.7%. Mat et al. [22] developed a Bayesian classifier and used system-based permission features to detect malicious mobile applications. The proposed model achieved a precision of 91.1%, an F-measure of 91% and an accuracy of 94%. In the Machine Learning and Natural Language Processing-based model created to detect malware, Nguyen et al. [23] examined the behavioral characteristics of Android smartphone users and the anomalies in these behaviors. They obtained 99.8% accuracy, 90% precision, 97.6% recall and a 93.6% F-measure by incorporating SVM into their proposed model.

In the study of Lu and Wang [24], the network traffic generated by the application and the network traffic matrix produced by the CNN deep learning model were both evaluated using the model, which the researchers called F2DC. The proposed model achieved an F-measure of 96.08%.

Using the hybrid model that Amer and El-Sappagh created [25], they analyzed API and system calls utilizing dynamic analysis techniques and LSTM. In addition, they used ensemble machine learning to examine the Android permissions and find malware by combining Random Forest (RF), MLP, AdaBoost, SVM and Decision Tree (DT) classifiers. This proposed model reached a 99.3% accuracy and 99% F-measure. Yang et al. [26] analyzed the characteristics of the Android software, such as permission and activity, and detected malware in accordance using the Contrastive Learning model. This model was

created using Bi-LSTM, which is commonly referred to as "Double-Sided LSTM", and the feature extraction was carried out using a Text-CNN-based model. The researchers reported that the proposed model achieved an accuracy of 97.53%, a precision of 96.66%, a recall of 98.41% and an F1-Score of 97.53%, reflecting the result of the evaluations using the AMGP and Drebin datasets.

Jerbi et al. [27] transformed the process of generating rules for malware detection into a two-fold optimization problem. As a result, their proposed Two-Level Malicious Application Detection (BMD) model had an F-measure of 97.79% and an accuracy of 98.18%.

With the method that they called DEEPSEL, Azad et al. [28] evaluated whether the application was malicious or not by evaluating several features of the dataset. The name of the model refers to the deep feature selection process they utilized in the process. In this study, machine learning-based algorithms and deep learning (DL)-based models were combined to perform the classification, and Particle Swarm Optimization was utilized for the feature selection. As a result of the evaluation performed on the CICAndMal2017 [29] dataset, the proposed model obtained 83.6% accuracy, 82.4% precision, 82.5% recall and an 82.5% F-measure.

D'Angelo et al. [17] detected malware by training two artificial neural networks, which they built using two-dimensional API images. These images serve as a signature of a program's activities over time. Subsequently, the features with the biggest impact on the findings were selected from the given features using an autoencoder and conventional artificial neural networks. The accuracy, precision and F-measure metrics for this model, which consists of two encoders and a SoftMax artificial neural network, were 94%, 98% and 97%, respectively.

Taha et al. [30] developed a novel fuzzy integral-based multi-classifiers ensemble model for Android malware detection. They combined the results of the XGBoost, RF, DT, AdaBoost and Light-GBM classifiers over the Choquet fuzzy integral. The experimental results obtained through the dataset, consisting of 9476 benevolent and 5560 malicious applications, showed that their proposed approach based on the Choquet fuzzy integral technique achieved a higher performance, with an accuracy value of 95.08% compared to those of the classifiers that were used individually. The risk-based fuzzy analytical hierarchy process approach was applied by Mohamad Arif et al. [18] in their Multi-Criteria Based Decision System and Mobile Malware Detection system. This methodology, which involved a static analysis, evaluated the permission-based features with the purpose of increasing the user's awareness of high-risk permissions through a risk analysis. The accuracy value for evaluations on the Drebin and AndroZoo datasets was 90.54%.

Mazaed Alotaibi and Fawad [31] built a Multifaceted Deep Generative Adversarial Networks (MDGAN) model for effective malware detection. With the three-stage proposed model, they converted the APK files into a binary image and an API sequence in the first stage. Then, by sending the image files to GoogleNet and sending the API sequence to the LSTM network, they obtained and determined the distinctive and stable features, respectively. Then, they applied the data with the combined feature to the Generative Adversarial Networks (GAN) and determined whether the data were a malicious application. They used the AndroZoo and Drebin databases as the dataset. They proved that the proposed model, with an accuracy value of 96.2% and an F-score value of 94.7%, which were obtained from the experimental studies, outperforms the studies that have been conducted in this context recently.

In a different study, Atacak et al. [32] used permission-based analysis and created a hybrid model by fusing CNN with FL. In the study, they used two different datasets and 500 benign and malicious applications. They analyzed the APK file of the applications and acquired a manifest.xml file. In the next stage, they obtained the permission information of the applications from this file. They performed a feature extraction and a feature reduction with two convolution layers and two pooling layers using the permission information. In the last layer, they estimated that the features equated to five neurons with ANFIS

architecture. With their model, they reached an accuracy of 92% in the first dataset and an accuracy of 94.66% in the second dataset.

In this study, unlike prior ensemble learning (EL) methods in the literature, the FL-BDE model combines the results of ML-based methods using a fuzzy logic (FL)-based inference system (FIS), which is based on the intuitive view and inference philosophy of a human solving a problem.

3. Materials and Methods

This section describes the materials and methods used in the study. The details of the dataset are explained in the "Dataset" section. In the "Dynamic Model Proposed for Android Malware Detection" section, the proposed method and used ML methods and FL-based method are explained. The section "Evaluation Metrics" describes the metrics used to evaluate the classification models.

3.1. Dataset

The applications installed on the Android operating system have the APK extension. In this study, a static analysis is performed on the APK files. A total of 2000 applications—1000 benign and 1000 malicious ones—are included in the dataset gathered for this study. The malicious instances are obtained from the Drebin [33] dataset. Drebin is the most widely used publicly available dataset for android malware detection. The Drebin dataset contains a total of 5560 malicious instances from 179 different malware families.

The benign instances are collected from the APKPure website. The APKPure web environment hosts applications in the Google Play Store environment, and they can be downloaded. One thousand and thirty-one different applications are downloaded in total via the APKPure online environment. All of the downloaded software applications underwent the Virus Total analysis and are deemed to be safe [34]. In the virus total application, the applications that are not considered to be harmless by at least one anti-virus software are excluded from the dataset.

3.2. Dynamic Model Proposed for Android Malware Detection

In this paper, a dynamic ensemble model, which utilizes a fuzzy inference system to interpret and combine the output scores of machine learning-based methods, is proposed as an effective approach for Android malware detection. As it can be seen from the schematic structure in Figure 1, the proposed model performs its task using six basic processes, namely, a feature extraction one, a feature selection one, a data splitting one, a multi-classification one, a voting and routing one and one that involves combining the scores. In the feature extraction process of the model, the reverse engineering method is applied to the android applications in the APK file format, and the permission information in the manifest.xml file of these applications is accessed and tagged. Following that, the labeled data and permission information are vectorized and saved in the database.

The feature selection process obtains a dataset with fewer features by eliminating the features that have no or a little impact on the classification process. The data splitting process makes it possible to produce the training and testing datasets needed for the classification process. The multi-classification process generates the output scores of the classifiers based on the testing dataset through 6 ML-based classifiers trained using a training dataset. The voting and routing process, as the name implies, first votes the output scores of the ML-based classifiers, and then, in accordance with the outcome of the vote, assigns three of them as the inputs of the FIS, which is a part of the process of combining the scores. In the last process of the model, namely, combining the scores, the scores of three classifiers, which were sent to FIS as inputs in the previous step, are combined, and then, the degree of malware (DOM) is determined for the data that are applied to the model as an input from the database. The sections below provide more information about these processes.

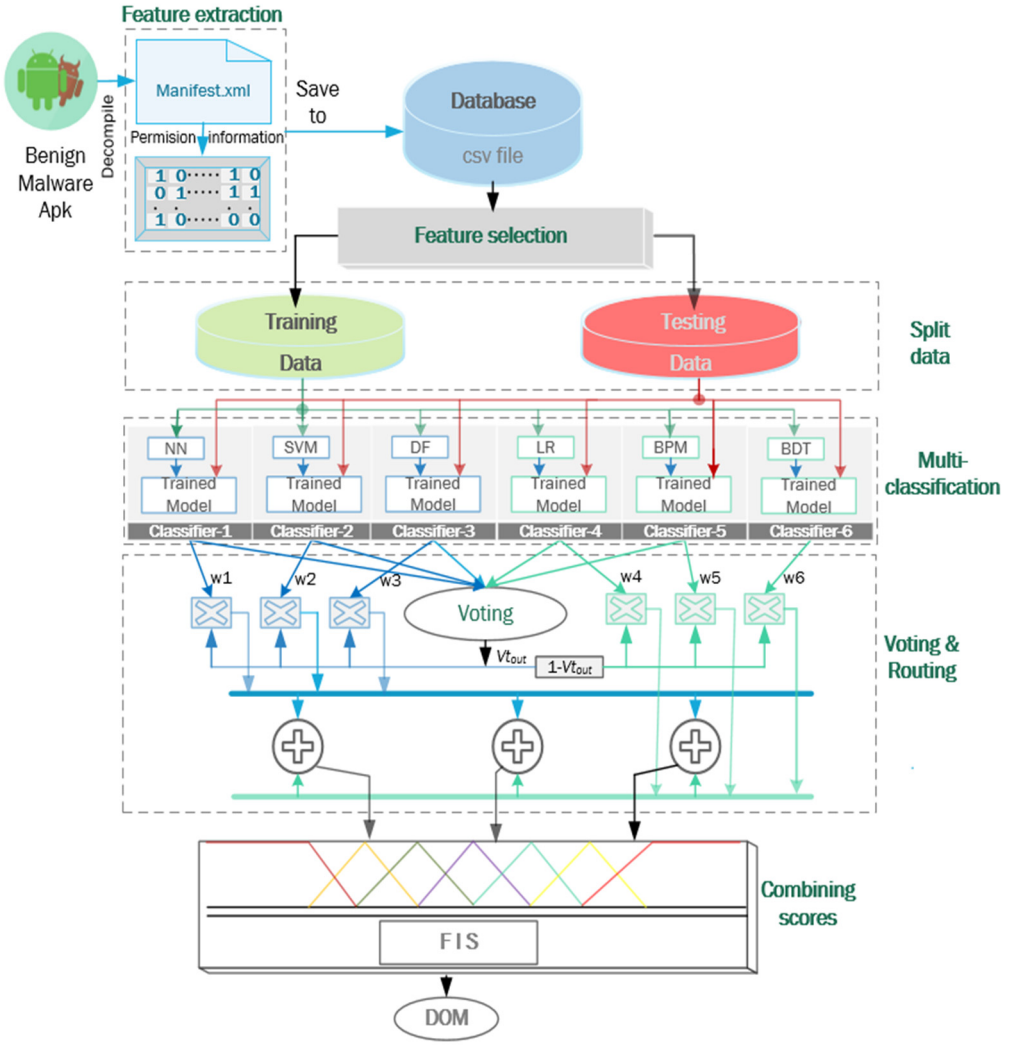


Figure 1. The schematic structure of the dynamic model proposed for Android malware detection.

3.2.1. Feature Extraction

Android applications are distributed and installed using the APK (Android Packet Kit) files. The APK file format contains files such as classes.dex and manifest.xml, from which the features of the application can be obtained. In this study, malware is detected by utilizing features that provide information about the behaviors, goals and permissions of the application. The manifest.xml file contains this information. To access the Manifest.xml file, the APK files need to be decompiled. The decompile process is conducted using the Jadx module. However, not every application can be decompiled. Applications that cannot be decompiled are, therefore, found and eliminated from the dataset. For the APK files that can be decompiled and be used for malware detection, the permissions, intentions and activities are identified by accessing the manifest.xml file. Feature vectors were generated for each application by assigning values of "1" if it includes the relevant property and "0" if it does not. As a result, the dataset that is to be used for the detection procedure is generated by creating feature vectors for all of the applications and written in an csv file. In order to prevent a class imbalance in the dataset, the feature vectors of 2000 applications in total, 1000 in each of the malicious and benign classes, are used.

3.2.2. Feature Selection

In studies using ML-based classifiers, feature selection is a vital process step that should be employed, as it can reduce the classifier training time and improve the classification performance by removing the irrelevant features. Feature selection is widely used in

permission-based android malware detection studies [35–39]. Although the applications' permission requests are effective in malware classification, not every permission request may affect the classification. For this reason, the feature selection process is generally applied in the studies. When the studies are examined, the most widely used method for the selection of permission-based features in android malware detection is the information gain one [18,36–38]. In [18,36–38], the information gain method was used, and the classification accuracy was greatly increased. information gain and Fisher score methods are filter-based methods which give us the effect value of the attributes in the classification. Filter-based methods collect the intrinsic properties of the measured features through univariate statistics. These methods are faster and less computationally costly than wrapper (Forward Feature Selection, Exhaustive Feature Selection and Recursive Feature Elimination, etc.) and embedded (LASSO Regularization (L1) and Random Forest Importance, etc.) methods are. In a comparison study conducted to obtain the information gain and Fisher score, it was seen that more successful results were obtained with the Fisher score method [40]. However, existing studies have used the information gain method, and their accuracy values are lower. Therefore, in this study, the Fisher score method is implemented in the feature selection process of the proposed model. This method uses a feature selection strategy that calculates the gain score so that the distance between the data points within the same class is as small as possible and the distance between the data points within different classes is as great as possible [41]. The feature set was then selected in accordance with this calculation. The Fisher score of a feature for the given classes a and b can be obtained using Equation (1).

$$FS = \frac{\sum_{a=1}^{b} na(\mu_{i,a} - \mu_i)^2}{\sum_{a=1}^{b} na\sigma_{i,a}^2} \quad (1)$$

In this equation, na represents the number of instances in the dataset, μ_i corresponds to the mean score of the features, $\mu_{i,a}$ stands for the mean score of the features in class a and $\sigma_{i,a}$ corresponds to the variance score of the features in class a. In the experimental studies, Fisher scores of other features were obtained according to the actual class label "CLASS". The experiments were performed for 1000, 750, 500, 250, 100, 50 and 25 features with the ensemble learning-based BDT classifier, and since 50 features gave the highest accuracy value, the number of features that had to be selected by the Fisher score method was determined as being 50. Therefore, at the end of the feature selection process, the 2000 × 1134-dimensional dataset obtained from the database is reduced to a 2000 × 50-dimensional dataset.

3.2.3. Splitting Data

The data splitting process is used to produce the training and testing datasets needed by the ML-based classifiers in the multi-classification process. Experimental studies are performed on two different datasets that had been split into 60% training: 40% testing data and 70% training: 30% testing data ratios. Therefore, the split ratios of 0.60 and 0.70 are used, respectively. The data are randomly distributed to the training and testing datasets based on these ratios, such that the numbers of benign and malicious instances are equal. At the end of the process, 800 test instances containing 400 benign and 400 malicious data and 1200 training instances containing 600 benign and 600 malicious data are obtained for the split ratio of 0.60. For the split ratio of 0.70, these numbers consisted of 1400 training data containing an equal number of benign and malicious instances and 600 testing data containing an equal number of benign and malicious ones.

3.2.4. Multi-Classification

This process collectively generates performance scores at the end of the training and testing phases of six different ML-based classifiers. The initial step of the multi-classification process is training the classifiers using the data from the data splitting process.

Subsequently, a performance score in the range of "0–1" is calculated for each classifier by testing the trained models using the test data. Binary form classifiers are utilized in the related process since the data are received in one out of two classes; they are either benign or malicious. As for the classifiers, a two-class LR, a two-class BPM, a two-class BDT, a two-class NN, a two-class DF and a two-class SVM, which are known to perform well in binary classification problems, are used.

Classifier 1: The two-class NN is a binary classification algorithm that consists of a series of layers containing nodes that transmit information to each other. It is typically represented by a structure that combines an input layer, an output layer and one or more hidden layers. The input layer is used to transmit input data to the network. The hidden layer(s) processes the incoming data over the nodes using the weights between itself and the input layer, and then it sends it to the output layer. The input layer's weighted data are processed in a node element by the output layer, which then produces the classified results. Following the model training, the relationship between the inputs and the outputs can be discovered by passing the transfer function [42], which computes the sum of the weighted inputs from the network's previous layers, through an evaluator, whose activation function is given, as shown in Equation (2).

$$Output = f\left(\sum_{i=1}^{n} x_i * w_i + b\right) \quad (2)$$

Here, x_1, $x_2, \ldots . x_n$ represents the incoming inputs from the previous layer, w_1, $w_2 \ldots \ldots . w_n$ represents the connection weights of the inputs, b represents the bias value and $f(.)$ represents the activation function. The step function, the sigmoid function and the ReLU function are commonly used as activation functions.

Classifier 2: The two-class SVM is a supervised learning method widely used in solving linear or nonlinear binary classification problems. This method, which was introduced in [43] for the solution of pattern recognition and classification problems, is used to manage both the linear and nonlinear classification processes. In the classification process of nonlinear problems, it makes use of several kernel-based functions such as linear kernel, polynomial kernel and radial basis ones, which are called kernel functions. Finding the hyperplane that maximizes the decision boundary to two classes labeled from any point in a vector space with training data forms the basis of the classification process in linear SVMs [44]. The points closest to the decision boundary are referred to as "supports" in this context, and the distance between the relevant supports is referred to as the "margin". As the margin increases, the discrimination between the classes will increase. The SVM for each input with feature D in a space with n data points can be given by Equation (3).

$$\{x_i, y_i\} \Big| i = 1 \ldots n,\ y_{i \in \{-1,+1\}}, x \in \Re^D \quad (3)$$

In this equation, the response variable is represented by y_i, an element from one of the classes -1 or +1, and x_i, an element of the input vector (real vector). The hyperplane that increases the distance between the classes -1 and +1 can be calculated using Equation (4).

$$w.x - b = 0 \quad (4)$$

The hyperplane offset, or the difference in the distance between the two classes, is represented by the value of $b/||w||$, where w is the normal vector perpendicular to the hyperplane.

Classifier 3: The two-class DF is an ensemble learning model based on the principle of combining the results of independent decision trees that perform as well as possible individually. In ensemble models, the results are generally aggregated through several statistical techniques such as Bagging and voting. In this study, the Bagging technique is used to combine the results obtained from the decision trees. This method's first step involves uniformly sampling the initial dataset to create a new dataset of the same size. Then, each instance in the dataset is authorized to grow a set of decision trees. The probability of new

data points is established in the following step using the class distributions, following the independent training of each tree in the decision forest [45]. The class probability of the decision forest is obtained by averaging the class probabilities of all of the trees, as shown in Equation (5).

$$p(c|v) = \frac{1}{T}\sum_{t}^{T} p_t(c|v) \tag{5}$$

Here, c represents the class label, v represents the new input vector, T represents the number of trees in the ensemble and $p_t(c|v)$ represents the class probability of each tree.

Classifier 4: The two-class LR method is a supervised learning algorithm widely used in classification problems. When the variable to be estimated is categorical, it is preferable to linear regression. The method for resolving two-class problems involves fitting the training data for the binary event into a logistic function, and then estimating the probability of the categorical dependent variable in accordance with that estimate [46,47]. Equation (6) can be used to obtain the logistic regression model's conditional probability.

$$\pi(x_i) = P(Y = 1|x_i) = \frac{e^{(\beta_0+\beta_1 x_1)}}{1 + e^{(\beta_0+\beta_1 x_1)}} = \frac{1}{1 + e^{-(\beta_0+\beta_1 x_1)}} \tag{6}$$

In this equation, $\beta' = [\beta_0, \beta_1, \ldots., \beta_p]$ represents the vector of the unknown parameters, x_j represents the $p \times 1$ dimensional vector of the arguments and $\pi(x_i) = P(Y = 1|x_i)$ represents the conditional probability of the model. The parameter estimates are obtained by maximizing the log-likelihood function obtained from this equation. By maximizing the log-likelihood function derived from this equation, the parameter estimations can be obtained.

Classifier 5: Using the kernel approach, the two-class BPM method successfully transforms a linear Bayesian classifier into a nonlinear classifier. The basic principle of the BMP method in classification is based on finding the Bayesian point, which is the center of this space, among the possible solutions in the version space created by a set of hypotheses for a certain set of training data. Each hypothesis in the version space corresponds to a classifier that assigns input vectors to the output vectors [47]. The BMP is trained with m-dimensional training data, $z = (x, y) = (\{x_1, y_1\}, \ldots\ldots\ldots, \{x_m, y_m\})$, where *h(x)* is the function representing the hypothesis for an input vector x. Equation (7) describes how the version space *V(z)* of BMP is created.

$$V(z) := \{h \in H | \forall_i \in \{1, \ldots\ldots, m\} : h(x_i) = y_i\} \tag{7}$$

Equation (8) illustrates the Bayesian method's approach, which labels a test instance provided to it throughout the classification process with the lowest loss in accordance with the posterior probability $P_{H\setminus Z^m=z}(h)$.

$$Bayes_z(x) := arg^{min}_{y\in Y} E_{E\setminus Z^m=z}[l(H(X), y)] \tag{8}$$

In this approach, the loss value is obtained through the function given in Equation (9).

$$l(y, y) = \begin{cases} 0 & y = \acute{y} \\ 1 & y \neq \acute{y} \end{cases} \tag{9}$$

Classifier 6: The two-class BDT is a form of EL where each tree corrects the classification mistakes of the one before it. It is implemented to solve both the classification and regression problems. Predictions are made on the basis of the entire tree community. By iteratively combining each weak learner, the BDT algorithm seeks to create a single strong learner. For this purpose, the algorithm first creates an empty weak learner community. It then obtains the available outputs for each training instance and calculates the gradient descents. Then, decision trees are created according to these gradients, and the classification errors of the trees are minimized. The algorithm iteratively repeats these steps until the loss function's

value is minimized [46]. This algorithm performs very well in solving problems related to tabular data. Another advantage is that it can also work when some of the data are missing.

3.2.5. Voting and Routing

As the name suggests, this process performs a two functions: voting and routing. The voting function produces a score of 0 or 1 according to the numerical status of the outputs of the six classifiers included in the multi-classification process with a score of 0.5 or more. Below is the equation for the function that produces the output for this process.

$$Vt_{out} = \begin{cases} 1 & 3 \leq n_{clf} \leq 6 \\ 0 & 0 \leq n_{clf} < 3 \end{cases} \tag{10}$$

Here, n_{clf} represents the number of classifiers that produce an output score of 0.5 or more. If this number is 3 or more, the result of the voting function is 1, otherwise, this result is equal to 0. According to experimental studies on the six classifiers, the NN, SVM and DF methods successfully detected the true positive rate (TPR), while LR, BPM and BDT effectively detected the true negative rate (TNR). The voting and routing processed are used to take advantage of the good performances of various classifiers, as well as to enable a simpler design by reducing the number of FIS entries that must be employed in the process of combining the scores. Therefore, if the voting function produces a score of 1, which means that the application is believed to be malicious according to the classifiers, then the NN, SVM and DF algorithms are selected to combine the scores due to having a high TPR ratio. Otherwise, classifiers LR, BPM and BDT with high TNR ratios are chosen, and their outputs are used in the following process. The values that must be sent to the FIS entries are obtained by multiplying the classifier scores selected over the routing function by their weights. Equation (11) shows the formula for the routing process.

$$I_{FIS}(i) = Vt_{out}.O_{clf}(i).w(i) + [1 - Vt_{out}].O_{clf}(j).w(j), \begin{matrix} i = 1,2,3 \\ j = 4,5,6 \end{matrix} \tag{11}$$

In this equation, $I_{FIS}(i)$ represents the i_{th} input that is used to the FIS, Vt_{out} shows the result of the voting function, $O_{clf}(i)$ represents the output of i_{th} classifier and $w(i)$ shows the weight of the i_{th} classifier on the input. The voting function (Vt_{out}) result determines which classifiers' output will be used in the combining scores process (FIS) based on its weight, as shown in the formula provided in Equation (11). The weights are calculated using the false positive rate (FPR) and the false negative rate (FNR) obtained from the experimental studies of the classifiers in the positive and negative sample classifier groups. With the experimental studies involving classification errors, both the positive and negative sample classifiers are determined, and the FNR and FPR error rates are determined for each classifier, and the necessary parameters for calculating the weights are obtained. The process of finding the weights for the positive sample classifiers is given in Equation (12), and the process of finding weights for the negative sample classifiers is given in Equation (13).

$$w_i = (1 - FNR_i)\ i = 1,2,3 \tag{12}$$

$$w_j = (1 - FPR_j)\ j = 4,5,6 \tag{13}$$

3.2.6. Combining Scores

The degree of malware (DOM) of the test data is calculated using the combined outputs from three machine-learning-based methods from the previous process. Based on the DOM value, it is determined whether the test data are malicious or benign. The malware of the data is assessed using the FIS by combining the output scores of the classifiers. Although there are numerous FIS structure types, the Mamdani and Sugeno types are the most frequently employed ones. Because it is more similar to human perception and can produce more generalizable results when it is combining classification output scores, the Mamdani-

type FIS is chosen for this study. The block diagram of the proposed Mamdani-type FIS, which combines the output scores of the classifiers chosen through the voting and routing process, is shown in Figure 2.

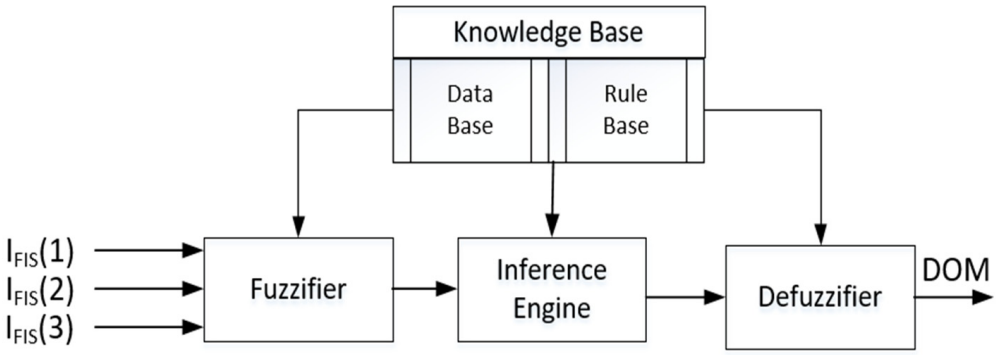


Figure 2. Block diagram of the FIS proposed to combine the scores of the specified classifiers.

In this structure, three weighted outputs of the voting and routing processes ($I_{FIS}(1)$, $I_{FIS}(2)$ ve $I_{FIS}(3)$) of the ML-based classifiers serve as the inputs that are applied to the FIS, and the DOM calculated using these inputs serves as the FIS' output. The boundary values of the input and output fuzzy universes, the type of set and the number of sets that need to be used in these universes are chosen in the first stage of the FIS' design. The voting and routing process generated ML-based classifier outputs as normalized values of between 0 and 1. Each input fuzzy universe determined five fuzzy sets for the verbal label very small (VS), small (S), medium (M), big (B) and very big (VB). Fuzzy sets were created using generalized bell-shaped (Gbell) membership functions in the input universes because ML-based classifiers' nonlinear structure caused them to produce a nonlinear output. In the FIS output universe, 13 Gbell fuzzy sets with the names D0, D1, D2, . . . , D11 and D12 were used to define a more precise rule. The output fuzzy sets labeled D0, D1, . . . , D12 represent the degree of malware. We tested the output of FIS with 5, 7, 9, 11 and 13 fuzzy sets to determine its output set number. For the FIS inputs and outputs, the defined membership functions are shown in Figure 3.

The rules that specify how the FIS' inputs and outputs relate to each other are defined in the second design stage. A total of 125 rules are defined in 3 input single output FIS, with each input being represented by 5 fuzzy sets. In the creation of the rules, the effect of each of the inputs on the output are considered to be equal. However, the performances of the classifiers are not the same. This effect was generated by multiplying the oriented classifier's score by the weights calculated according to the true positive or true negative ratio before the outputs of the ML-based classifiers from the voting and routing process were applied to the FIS input. A section of the rules created to combine ML-based classifiers are given in Table 1.

Table 1 shows that examples of rules are as follows:

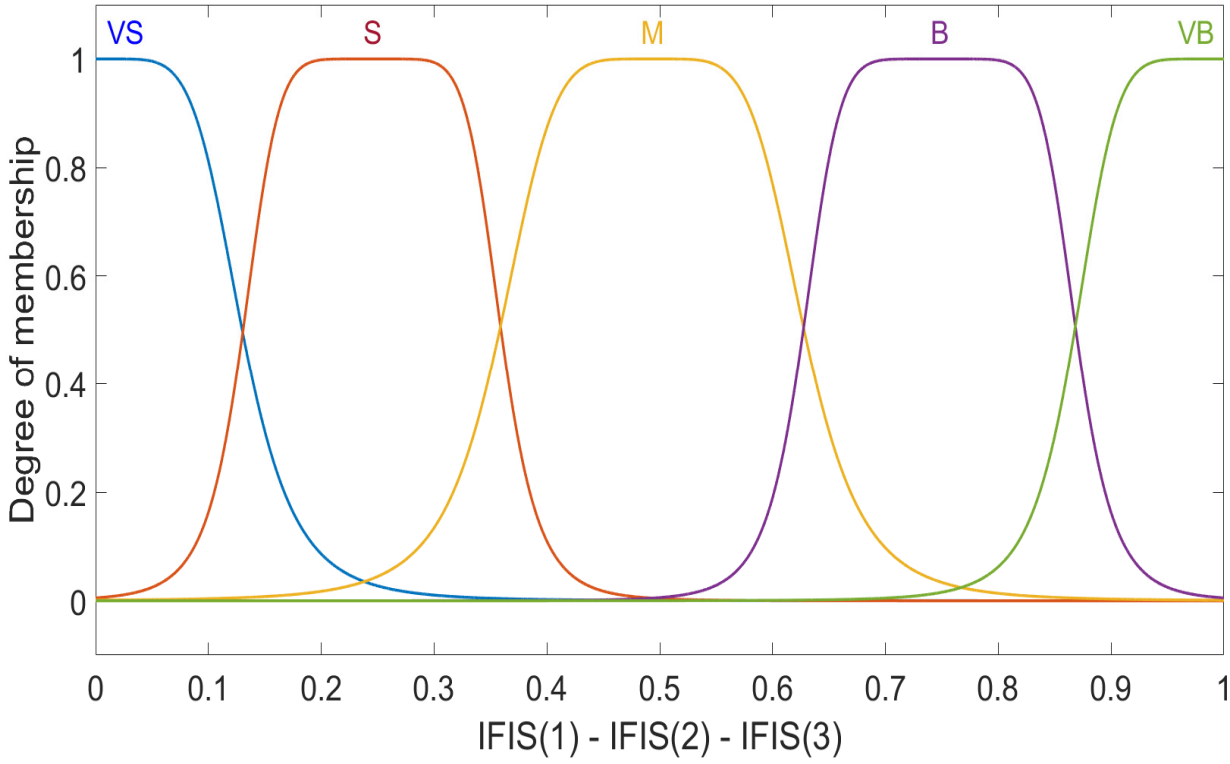
If $I_{FIS}(1)$ is VS and $I_{FIS}(2)$ is VS and $I_{FIS}(3)$ is VS, then DOM is D0.

If $I_{FIS}(1)$ is S and $I_{FIS}(2)$ is S and $I_{FIS}(3)$ is S, then DOM is D3.

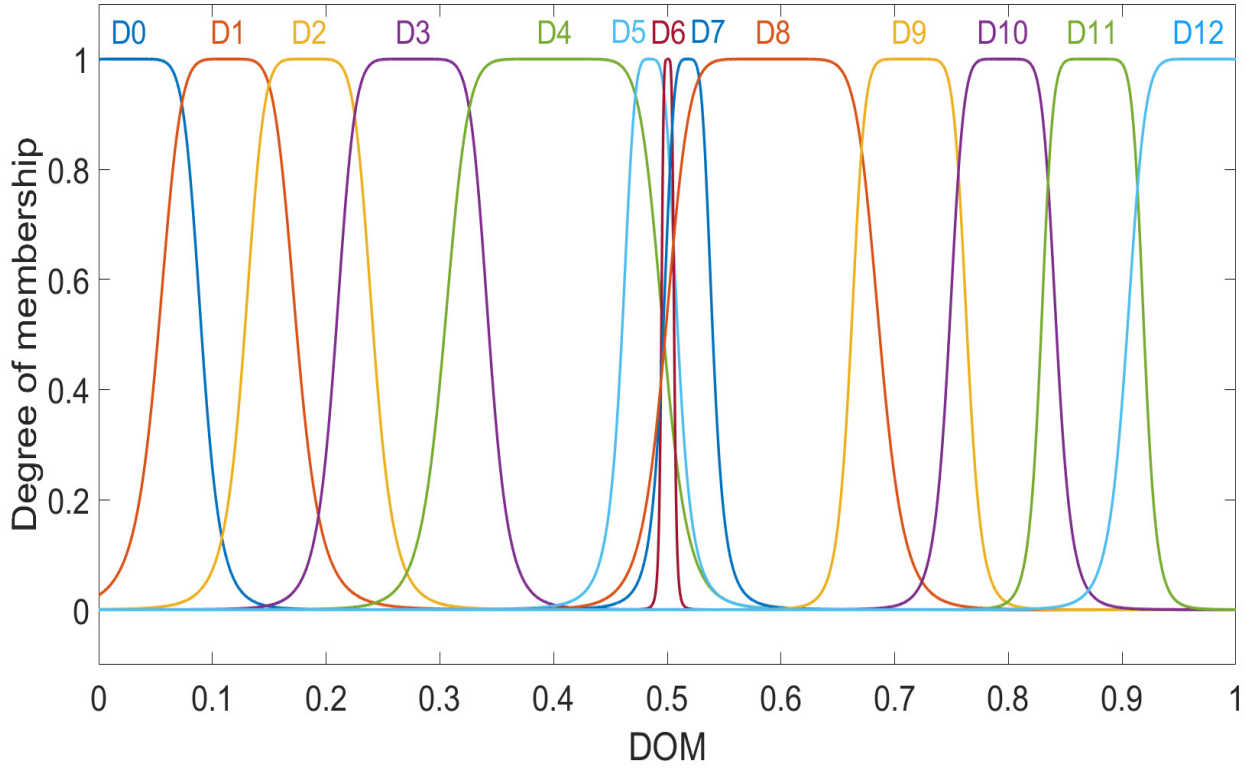
If $I_{FIS}(1)$ is VB and $I_{FIS}(2)$ is VB and $I_{FIS}(3)$ is VB, then DOM is D12.

The methods we employ in the inference process include two steps: inference and aggregation, which are chosen in the third stage of design. The two steps are carried out using the "min" and "max" operators, respectively. The defuzzification approach is selected at the end of the process. The centroid approach, the formula of which is provided in Equation (14), is applied in this procedure as the defuzzification method.

$$DOM = \frac{\sum_{i=1}^{n} \mu(x_i).x_i}{\sum_{i=1}^{n} \mu(x_i)} \quad (14)$$



(**a**)



(**b**)

Figure 3. The membership functions defined for the inputs and outputs of the FIS: (**a**) IFIS(1), IFIS(2) and IFIS(3) inputs; (**b**) DOM output.

Table 1. A cross-section of the rules used to combine ML-based classifiers.

	$I_{FIS}(1)$						
		VS	**S**	**M**	**B**	**VB**	
$I_{FIS}(2)$	**VS**	D0	D1	D2	D3	D4	**VS**
	S	D1	D2	D3	D4	D5	
	•	•	•	•	•	•	
	VB	D5	D6	D7	D8	D9	
	VS	D1	D2	D3	D4	D5	**S**
	S	D2	D3	D4	D5	D6	
	•	•	•	•	•	•	
	VB	D6	D7	D8	D9	D10	
	•	•	•	•	•	•	•
	VS	D4	D5	D6	D7	D8	**VB**
	S	D5	D6	D7	D8	D9	
	•	•	•	•	•	•	
	VB	D8	D9	D10	D11	D12	

$I_{FIS}(3)$

Here, n is the discrete universe's total number of instance values for the output, x_i is the output value at the ith instance value and $\mu(x_i)$ is the membership value for the x output at the ith instance value.

3.3. Performance Assessment

Confusion matrix-based metrics are widely used to measure the performance of the methods used in solving binary classification problems. The confusion matrix is an indicator that numerically represents the accuracy or inaccuracy of the instance predictions made using the classifiers that are proposed in the classification problems. This indicator gives us the number of correctly predicted positive instances, "True Positive (TP)" ones, the number of incorrectly predicted positive instances, "False Negative (FN)" ones, the number of correctly predicted negative instances, "True Negative (TN)" ones, and the number of incorrectly predicted negative instances, "False Positive (FP)" ones. Based on this information, the evaluation metrics for the classifiers based on the confusion matrix can be calculated. In this study, the accuracy (Acc), recall (Rec), precision (Prec), specificity (Spec), F-measure (F-m) and area under the curve (AUC) metrics are used to measure the performance of the proposed FL-BDE model and the ML-based models for comparison purposes. The information about these metrics and their calculations is presented below.

Accuracy: This metric, which represents a heuristic performance measure, gives the correct prediction ratio among all of the instances. As it can be understood from Equation (15), its score can be found by dividing the number of correctly predicted instances by the total number of instances.

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{15}$$

Recall: It is also known as sensitivity. This metric, which gives the correct prediction ratio within positive instances, can be obtained using Equation (16).

$$Rec = \frac{TP}{TP + FN} \tag{16}$$

Precision: It is a ratio that shows the relationship between the total number of positively predicted instances and the number of correctly predicted positive instances. Its score can be calculated by the formula given in Equation (17).

$$Prec = \frac{TP}{TP + FP} \tag{17}$$

Specificity: It is simply a metric that gives the ratio of the number of correctly predicted negative instances to the number of negative instances. The formula that gives its score is Equation (18).

$$Spec = \frac{TN}{TN + FP} \tag{18}$$

F-measure: It is a generally preferred useful metric in performance measurements, especially in cases where there is an unbalanced class distribution. This metric determines its own score based on the harmonic mean of precision and recall, as seen in Equation (19).

$$F - m = 2 \times \frac{Prec \times Rec}{Prec + Rec} \tag{19}$$

Area under the curve: It is an important metric used to test whether the model can work correctly to solve classification problems. Its score can be obtained using the formula given in Equation (20), which gives the area of the receiver operating characteristic (ROC) curve.

$$AUC = \int TPR.d(FPR) \tag{20}$$

Here, TPR is the true positive rate, and FPR is the false positive rate. Since the ROC curve is a graph showing the relationship between these two ratios, the AUC score is calculated by integrating the TPR with respect to the FPR as the area under this curve.

4. Results and Discussion

In this section, the experimental results of the proposed FL-BDE model for detecting Android malware and nine different ML- and EL-based methods used to verify the model's performance are presented. The success of the proposed model in malware detection is proven by comparing the experimental results with the studies in the literature that use similar models and techniques. The FL-BDE model's experimental setup was built in the Microsoft Azure ML environment using the components that are classified into several categories in the Materials and Methods section (Chapter 3). In the first phase of the setup, the data were read from the csv file containing 2000 applications, with a total of 1134 features, which had been uploaded to the Azure ML platform through the "My Datasets" component. The relevant data were therefore applied to the "Filter Based Feature Selection" component and reduced to a 2000 $\times$ 50 dataset. The "Feature Selection Method" and "Number of Desired Features" parameters in this component were set to the "Fisher score" and "50", respectively. To obtain the training and testing data for the ML-based models, the dataset with the decreased feature set was then subject to the data splitting procedure. The Azure ML environment's "Split Data" component was used to carry out this process. Depending on the two-stage splitting strategy that had to be employed in the test phase, the "fraction of rows in the first output dataset" parameter, which displays the split ratio within the component, was set to 0.60 or 0.70, respectively. Three components, "Two Class Algorithm Name," "Train Model" and "Score Model", were used to produce the malicious rating results in the 0–1 range of the models developed using the SVM, LR, BPM, BDT, DF and NN approaches utilizing the training and test data for the multi-classification process. The parameters adjusted for the mentioned classification algorithms and their value changes are shown in Table 2.

Table 2. The setting parameters and their values used in the Azure ML environment of the classifiers used in the multi- classification process.

ML-Based Models	Setting Parameters
SVM method	Number of iterations: 15 Lambda: 0.02
LR method	Optimization tolerance: 1×10^{-8} L1 regularization weight: 1 L2 regularization weight: 20 Memory size for L-BFGS: 40
BPM method	Number of training iterations: 20
BDT method	Maximum number of leaves per tree: 8 Minimum number of instances per leaf node: 2 Learning rate: 0.2 Number of trees constructed: 64
DF method	Resampling method: Bagging Number of decision trees: 32 Maximum depth of the decision trees: 128 Number of random splits per node: 256 Minimum number of instances per leaf node: 1
NN method	Number of hidden nodes: 30 Learning rate: 0.125 Number of learning iterations: 100

The stages, including the voting and routing processes and the combining scores process of the proposed model, with the execution of the single classifiers that are not part of Azure machine learning, were implemented with a program written on the "Execute R script" component in this environment.

Along with the proposed FL-BDE model in the experimental studies, the ML-based single classifiers such as SVM, LR, BPM, NN and RF and the EL-based single classifiers such as BDT, DF, AdaBoost and Bagging were also used to evaluate its performance in Android malware detection. The performances of the model and classifiers were verified on the training and testing data obtained by applying the random split and k-fold cross-validation conditions to the dataset. Splits of 0.60 and 0.70 were used as the random split condition. Data vectors of 1200×50 for the training process and 800×50 for the testing process were obtained in the random split condition of 0.60. For the split ratio of 0.70, the sizes of these vectors for the training and testing processes were 1400×50 and 600×50, respectively. In the cross-validation approach, five-fold cross-validation was used. Firstly, the dataset was divided into five equal parts, and five data groups were obtained, each consisting of 400×50 vectors. Then, after performing five iterative test processes, where there was one group of testing data and four groups of training data, the performance results of the classification models were obtained for each iteration step. After that, by averaging the performance results obtained in these iteration steps, the overall performance of the model and methods used for the Android malware detection was determined.

In Table 3, the classification errors and confusion matrix parameter values for the FL-BDE, ML- and EL-based models are given for the split ratio of 0.60. The proposed FL-BDE model, with a value of 1.88%, produced the best performance in terms of the misclassification rate (MCR), which represents the classification error in all of the positive and negative classes. When it is compared to the results produced by the ML- and EL-based models, this result corresponds to an extremely low value. In fact, it had an MCR of 1.87% less than the one (an MCR of 3.75%) of the model created using the RF method, which produced the closest value to that of our model. Given that the MCR is inversely correlated with the classification accuracy, it can be concluded that the proposed approach shows an excellent performance in this regard. In terms of the false positive rate (FPR), which indicates the proportion of negative instances that were predicted to be positive,

and the false negative rate (FNR), which indicates the proportion of positive instances that were predicted to be negative, the FL-BDE model also performed quite well. The results in Table 3 also show us that the proposed model's 2.75% FPR and 1% FNR values were considerably lower than the FPR and FNR values of the other models. Despite it having the same FNR value as the proposed model, the NN-based model performed the worst out of all of the models, since it had the greatest MCR and FPR values. The NN-based model achieved an MCR value of 10.63% and an FPR value of 20.25%.

Table 3. Confusion matrix parameters and error rates of FL-BDE, ML- and EL-based models at the split ratio of 0.60.

Method	Confusion Matrix Parameters				Error Rates		
	TP	FP	TN	FN	MCR%	FPR%	FNR%
SVM	374	25	375	26	6.38	6.25	6.50
LR	375	12	388	25	4.63	3.00	6.25
BPM	376	16	384	24	5.00	4.00	6.00
BDT	382	13	387	18	3.88	3.25	4.50
DF	386	17	383	14	3.88	4.25	3.50
NN	396	81	319	4	10.63	20.25	1.00
AdaBoost	381	20	380	19	4.88	5.00	4.75
Bagging	379	22	378	21	5.38	5.50	5.25
RF	387	17	383	13	3.75	4.25	3.25
FL-BDE	396	11	389	4	1.88	2.75	1.00

Figure 4 shows the performance results of the FL-BDE, ML- and EL-based models at the split ratio of 0.60. According to the performance values shown in Figure 4, the FL-BDE model offered the best performance among all of the models in terms of the accuracy, recall, specificity, precision and F-measure metrics. The proposed model achieved results with an accuracy of 0.9813, a recall of 0.9900, a specificity of 0.9725, a precision of 0.9730 and an F-measure of 0.9814. While the RF-based model was the most similar one the proposed model in terms of the accuracy and F-measure metrics, the LR-based model achieved a similar result, with values of 0.9700 and 0.9690 for the specificity and precision metrics. Of all of the models, the NN-based model exhibited the lowest performance in terms of the other metrics, except for recall, with values of 0.8938 accuracy, 0.7975 specificity, 0.8302 precision, and 0.9031 for the F-measure. However, this model provided the best performance in terms of the recall metric, along with the FL-BDE model, with a value of 0.9900.

Figure 5 illustrates ROC curves that demonstrate the relationship between the false positive rates and true positive rates for the FL-BDE, ML- and EL-based models at the split ratio of 0.60. The value of the area under the ROC curve (AUC) gives a critical measure of the model's success in separating the classes. The closer the AUC values of the models are to 1, then the more distinct their discriminatory power in distinguishing the classes is. In this regard, when the AUC values derived from the ROC curves, which are presented in Figure 5, were compared with each another, it was observed that all of the models, except for the SVM-based model, display a very good performance. With an AUC value of 0.997, the proposed model demonstrated an outstanding performance in terms of class discrimination. The DF and BDT-based models, with AUC values of 0.990 and 0.992, respectively, were the most similar to our model in terms of performance.

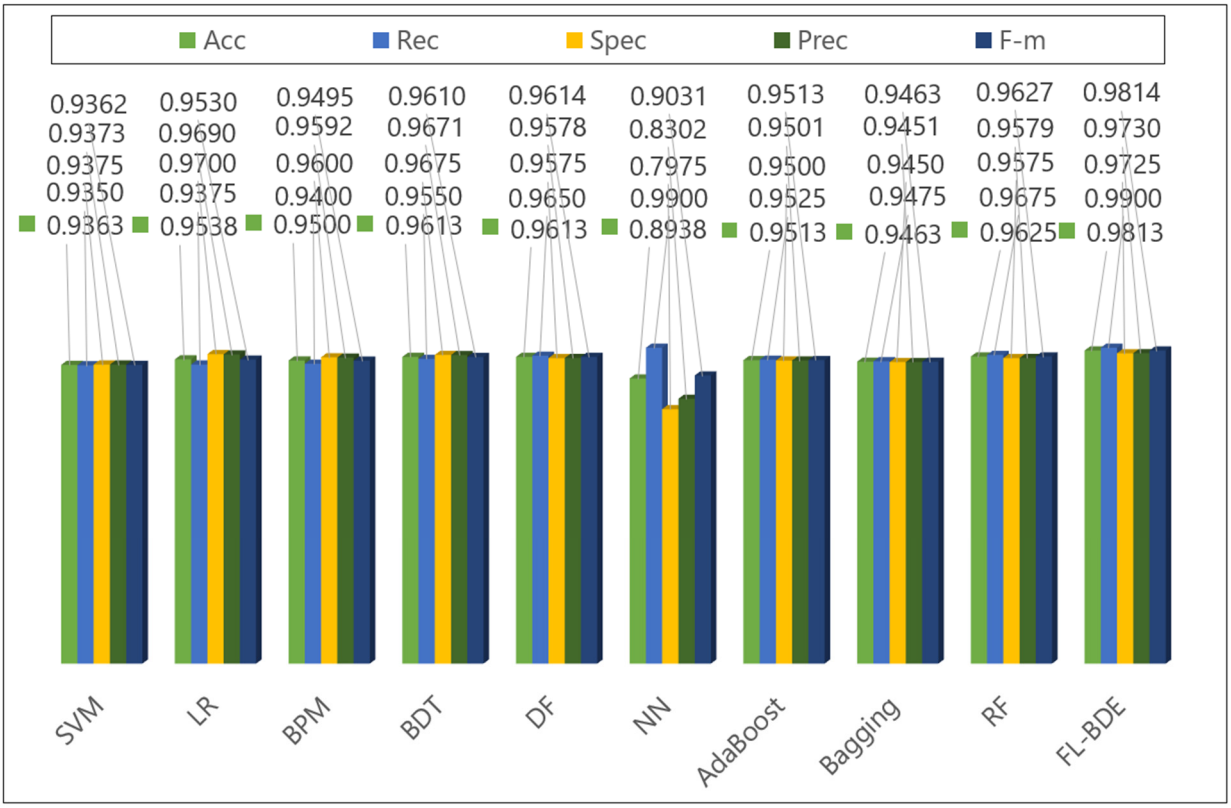


Figure 4. Performance results of FL-BDE, ML- and EL-based models at the split ratio of 0.60.

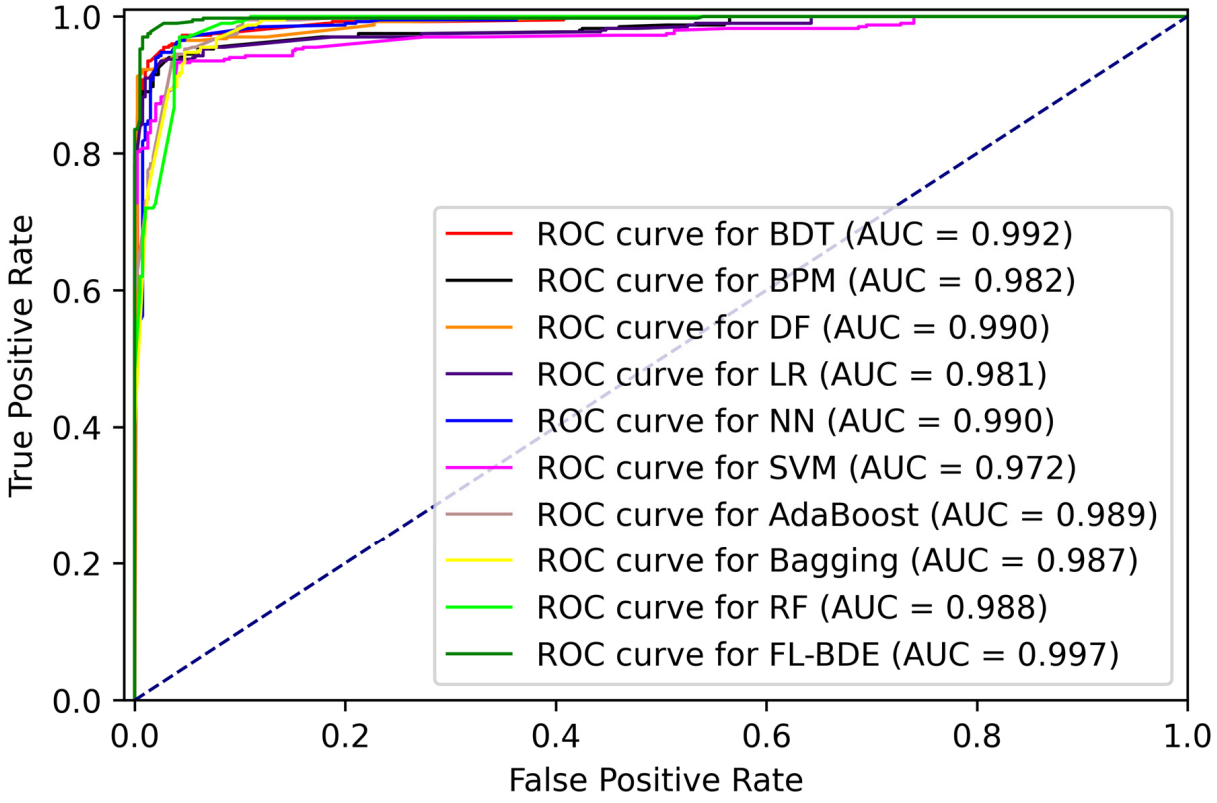


Figure 5. ROC curves of FL-BDE, ML- and EL-based models at the split ratio of 0.60.

Table 4 compares the confusion matrix parameters and error rates of the FL-BDE, ML- and EL-based models at the split ratio of 0.70. As in the 0.60 split conditions, it can be clearly seen from the data in Table 4 that the best performance is reflected in the lowest MCR, FPR and FNR error rates under these split conditions for the proposed model. The proposed model, with an MCR value of 0.67% and an FPR value of 1.33%, achieved error rates that were 3% and 1% lower for both of the performance values, respectively, than those of the BPM-based model, which produced the closest result to it. Furthermore, it correctly classified all of the positive instances, resulting in an FNR of 0%. In terms of the MCR and FPR error rates, the NN-based model had the worst performance, with values of 9.83% and 18.67%, respectively. However, this model was the model that provided the best performance at this error rate, after the proposed model, with an FNR value of 1%. With an error value of 6.67%, the Bagging-based model had the worst performance in terms of the FNR.

Table 4. Confusion Matrix parameters and error rates of FL-BDE, ML- and EL-based models at the split ratio of 0.70.

Method	Confusion Matrix Parameters				Error Rates		
	TP	**FP**	**TN**	**FN**	**MCR%**	**FPR%**	**FNR%**
SVM	283	18	282	17	5.83	6.00	5.67
LR	281	10	290	19	4.83	3.33	6.33
BPM	285	7	293	15	3.67	2.33	5.00
BDT	290	12	288	10	3.67	4.00	3.33
DF	287	14	286	13	4.50	4.67	4.33
NN	297	56	244	3	9.83	18.67	1.00
AdaBoost	282	9	291	18	4.50	3.00	6.00
Bagging	280	15	285	20	5.83	5.00	6.67
RF	286	9	291	14	3.83	3.00	4.67
FL-BDE	300	4	296	0	0.67	1.33	0.00

The performance results of the FL-BDE, ML- and EL-based models are shown in Figure 6 for the split ratio of 0.70. When these results were compared with the results obtained from the split conditions of 0.60, it was seen that all of the models, aside from the LR, DF, Bagging and RF based-models, improved their performances in terms of the most of the metrics, with the growth occurring in the number of instances trained at the split ratio of 0.70. The proposed model had the best performance in this period, improving its performance by 1.2% for accuracy, 1% for recall, 1.42% for specificity, 1.38% for precision and 1.2% for the F-measure, according to the performance results at the split conditions of 0.60. From the performance results at the split ratio of 0.70, it can be clearly seen that the FL-BDE model had a much better performance than the ML- and EL-based models did, with an accuracy of 0.9933, a recall of 1.00, a specificity of 0.9867, a precision of 0.9868 and an F-measure of 0.9934. It also produced a better performance in terms of the value differences, which are 3% or more for accuracy, 3.33% or more for recall, 1% or more for specificity, 1.08% or more for precision and 2.99% or more for the F-measure, than those of the RF, BPM and BDT-based models, which have the most similar performance results to it in terms of most of the confusion matrix metrics. The NN-based model had the worst performance in terms of all of the metrics, except for the recall metric. With a value of 0.99, it came the closest to that of the proposed model with regard to this metric.

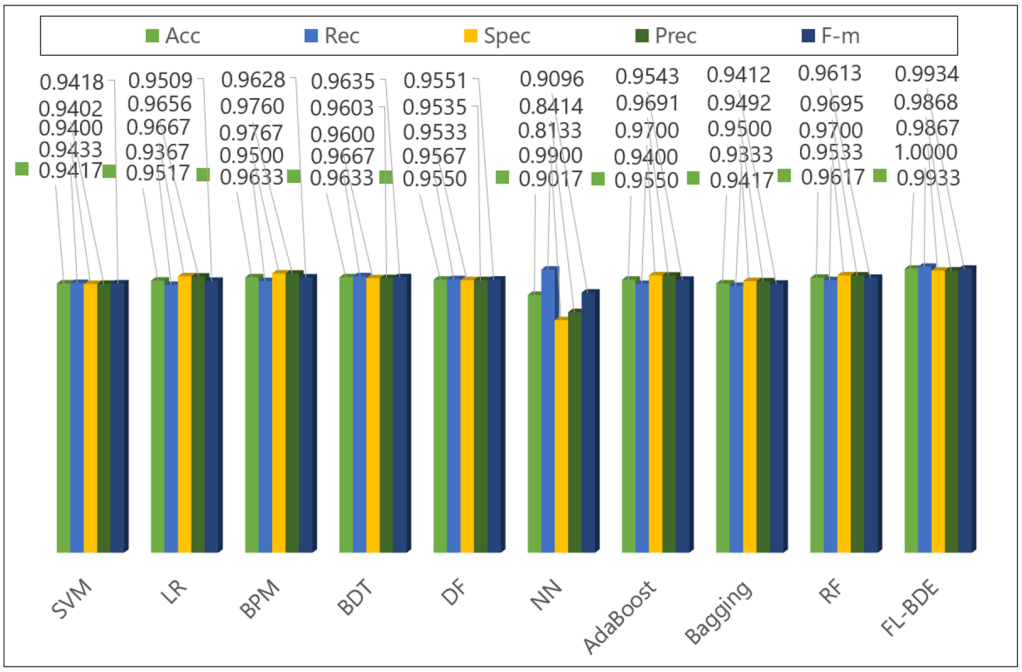


Figure 6. Performance results of FL-BDE, ML- and EL-based models at the split ratio of 0.70.

Figure 7 shows the ROC curves and their AUC values for the models built using the proposed FL-BDE, ML- and EL-based methods at the split ratio of 0.70. When the AUC values of the models built here were compared with the AUC values at the split ratio of 0.60, it was seen that while there was no change in the performance result of the BDT model, the ones of the DF and NN-based models showed a decreasing tendency of 0.1. This decrease was by 0.3% in the Bagging model. On the other hand, the performance of the remaining models in terms of this metric improved by 0.2% or more. From the obtained results, it can be said that all of the models show good performances since they all obtained an AUC performance of 0.98 or more in the split ratio of 0.70. When the AUC results of the models were evaluated among themselves, the proposed model showed an excellent performance, with a very high value of 0.999. The closest AUC performance to this model was provided by the AdaBoost and RF-based models, with values of 0.993.

Figure 8 shows the error rates related to the classification performance of the FL-BDE, ML- and EL-based models obtained using five-fold cross-validation. From the error rate values in the figure, it can be clearly understood that the FL-BDE model showed the best performance among all of the models, reaching the lowest error rate, with 1% and 0.1% values, respectively, in terms of both the MCR and the FPR. The proposed model achieved lower error rates of 2.1% in the MCR and 3% in the FPR than those of even the BDT model, which gave the most similar result to it. Although the NN model achieved the lowest error rate, with a value of 1.5 in terms of the FNR, it also has a high error rate value of 10.40% in terms of the MCR as a result of its very high FPR of 19%. This made it the best-performing model among all of the models. The LR was the model with the worst performance in terms of the FNR, with a value of 6.20%. When these results were compared with the results obtained in the conditions with the split ratios of 0.6 and 0.7 in terms of the MCR, it was seen that the error rate values of the Ensemble based-BDT, AdaBoost and Bagging models obtained by five-fold cross-validation were lower than the ones of the same models at both of the split ratios. Such as the model we proposed, most of the other models achieved an error rate value that was below the 0.6 split ratio, while providing an error rate value that was above the 0.7 split ratio for this metric in five-fold cross-validation. The FNR of the FL-BDE model in the relevant validation conditions produced an error rate that was above the values obtained in both of the split conditions. However, in all of the conditions, it

obtained a result that was much lower than the error rate values of the other models, except for the NN model.

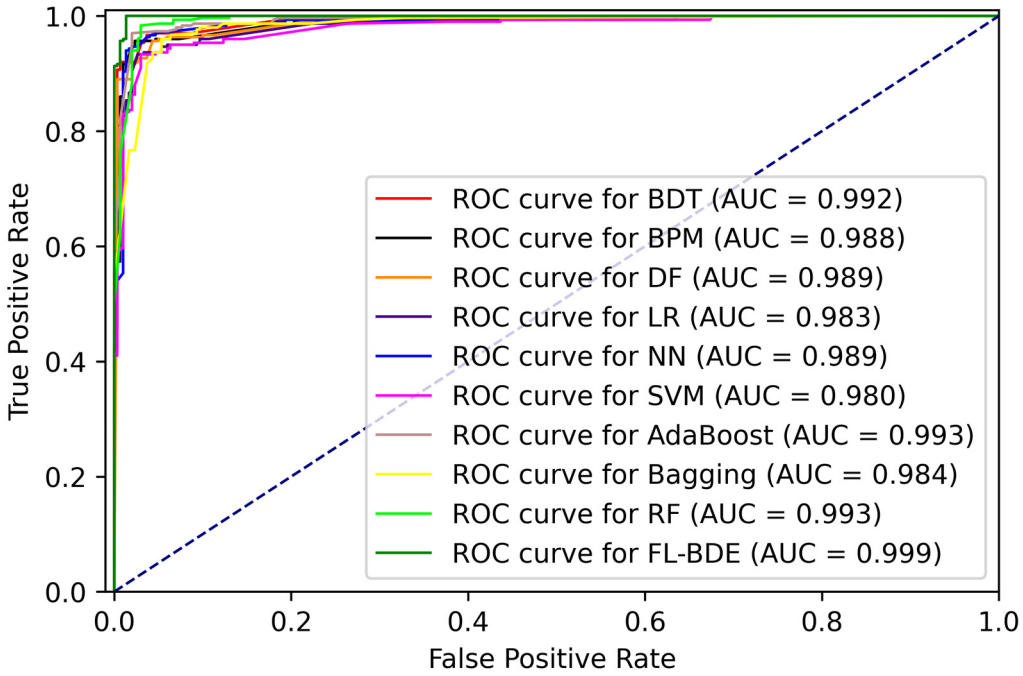


Figure 7. ROC curves of FL-BDE, ML- and EL-based models at the split ratio of 0.70.

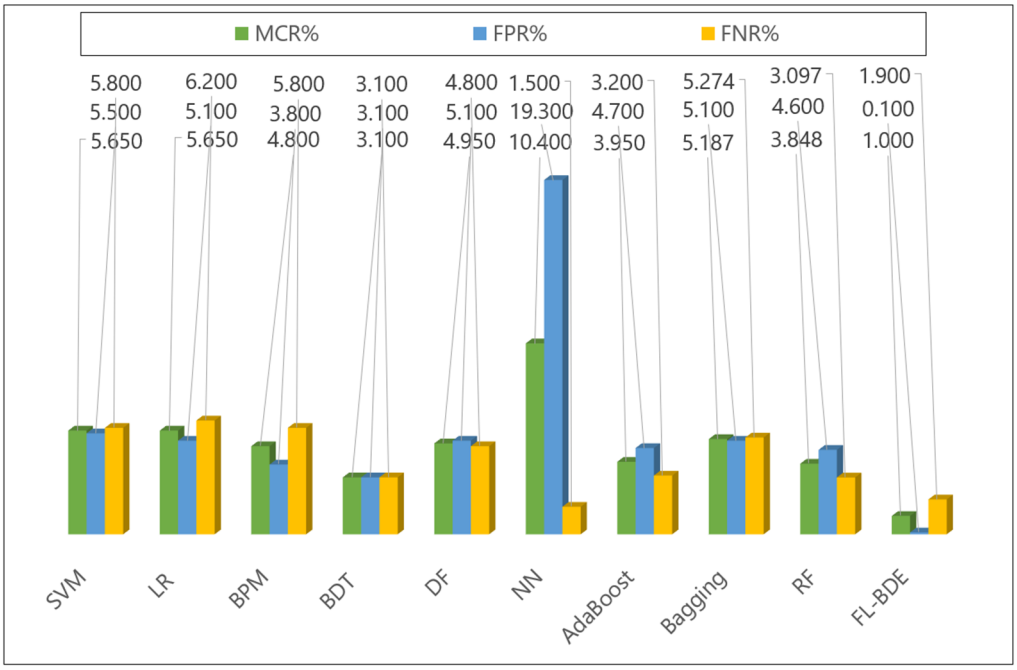


Figure 8. Error rates of FL-BDE, ML- and EL-based models for 5-fold cross-validation.

Figure 9 illustrates the performance results of FL-BDE, ML- and EL-based models created using five-fold cross-validation. From the performance values depicted in the figure, the FL-BDE model performed considerably better than the ML- and EL-based models did in terms of the accuracy, recall, specificity, precision and F-measure metrics, as well as the performance results in the split ratios of 0.60 and 0.70. The proposed model achevied the results of an accuracy of 0.990, a recall of 0.981, a specificity of 0.999, a precision of 0.999 and an F-measure of 0.990. In fact, the model showed a better performance of 2.1% in

terms of the accuracy and 3% in terms of the specificity, precision and F-measure metrics than the BDTmodel did, which produced the most similar result to it, except for the recall metric. The RF was the model that has the most similar result to that of our model, with accuracy values of 0.962, a recall of 0.969, a specificity of 0.954, a precision of 0.955 and an F-measure value of 0.962 after those of the BDT model. As in the split ratios of 0.60 and 0.70, the NN model gave the worst performance for all of the metrics, except the recall metric. When the performance results of Figures 4 and 9 were compared, it can be understood that the FL-BDE based model tested in five-fold cross-validation conditions had a better performance values in terms of all of the metrics, except the recall metric at the split ratio of 0.60. When Figures 6 and 9 were examined, a similar situation can be observed for the performance results at the split ratio of 0.70 in terms of the specificity, precision and F-measure metrics.

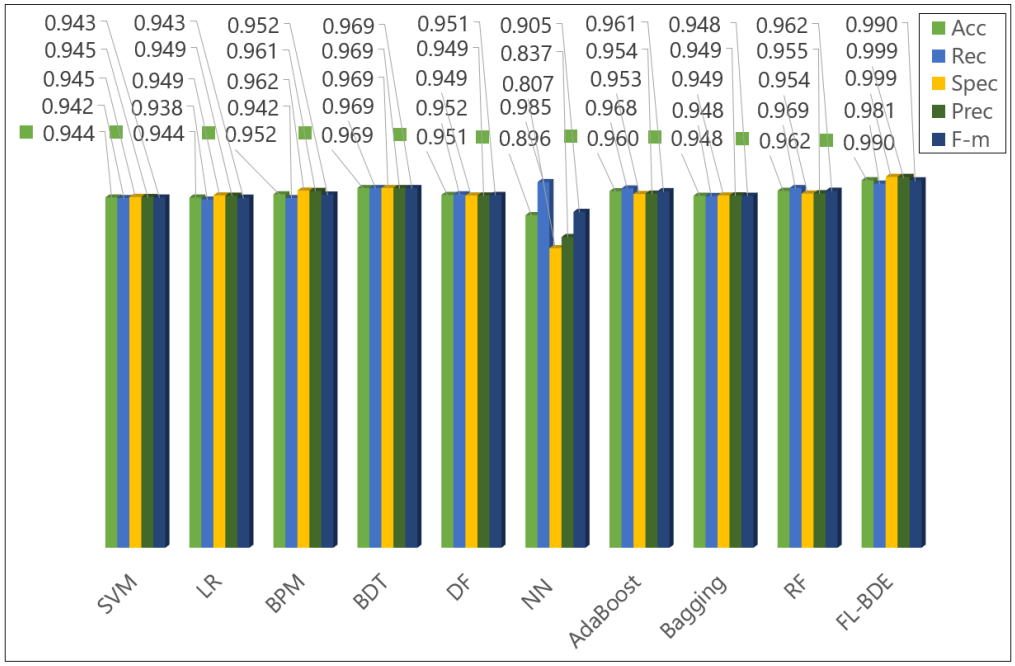


Figure 9. Performance results of FL-BDE, ML- and EL-based models for 5-fold cross-validation.

Figure 10 shows the ROC curves and AUC values obtained for the FL-BDE, ML- and EL-based models in the condition of five-fold cross validation. From the performance outputs that are depicted, it can be seen that the proposed model had a better performance than the other models did, with an AUC output of 0.997. As in the performance results of other metrics, the ensemble learning-based BDT model had the best performance after our model, with a value of 0.993 in terms of this metric. For the conditions of five-fold validation, the SVM model has the lowest AUC score among all of the models. In fact, the SVM showed the worst performance in both the random split conditions and the five-fold cross-validation conditions.

The experimental results showed that the proposed FL-BDE model produced the best performance in terms of most performance metrics in the random split ratio conditions and the five-fold cross-validation conditions. Here, it was decided which classifiers used in the voting and routing process would be included in the positive and negative classifier groups using the experimental results of the classifier errors. When the performances of ML-based models in both the split conditions and the cross-validation were evaluated according to the average FPR and FNR error rates, it was found that the LR, BPM, and BDT-based models provided the best performances in terms of the classification of negative instances, while the NN, DF and SVM-based models had the best performances in classifying the positive instances. This information is critical in structuring the proposed model.

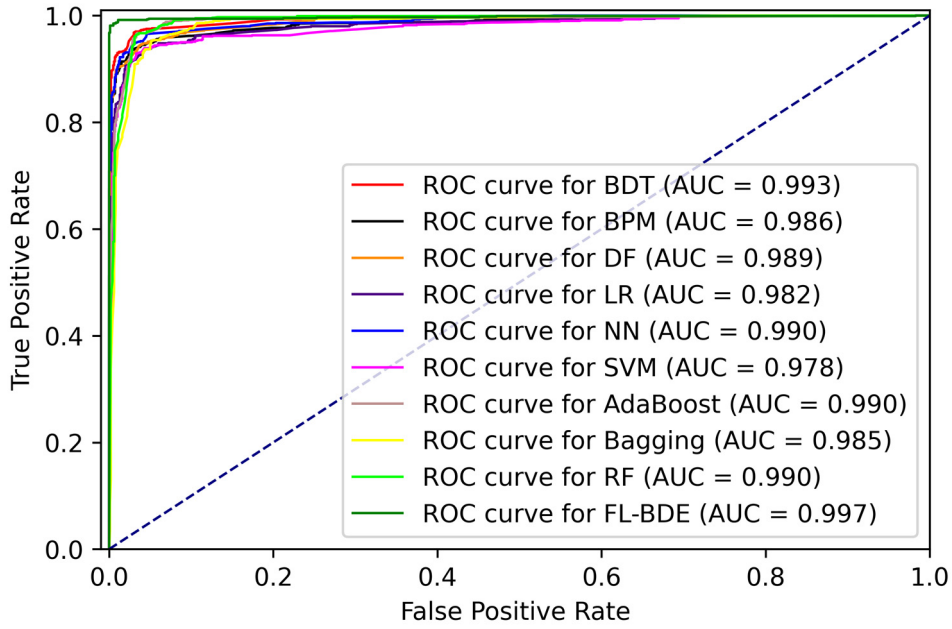


Figure 10. ROC curves of FL-BDE, ML- and EL-based models for 5-fold cross-validation.

In the literature, many studies have been conducted to detect malware through different classification methods by using static and dynamic analysis methods. Among them, artificial intelligence-based methods such as ML, DL and FL, as well as hybrid and EL-based approaches, have performed better in malware detection than the others have. A summary of the proposed model is presented in Table 5, along with information about the relevant studies in the literature and their performances in terms of the accuracy, recall, precision, F-measure and AUC metrics.

Table 5. A summary of the proposed model and the relevant studies in the literature and their performance results.

No.	Author	Method(s)	Dataset	Acc (%)	Prec (%)	Rec (%)	F-m (%)	AUC (%)
1	Lu and Wang [24]	CNN	Drebin and CICMalDroid	-	96.03	96.3	96.08	-
2	Jerbi et al. [27]	Bi-Level Optimization	Drebin and AAGM	98.18	98.06	98.34	97.79	86.80
3	Mat et al. [22]	Naïve Bayes	Drebin and AndroZoo	94	91.1	-	91	96.75
4	Yang et al. [26]	Contrastive Learning	Drebin and AMGP	97.53	96.66	98.41	97.53	-
5	Taha et al. [30]	Fuzzy-Integral Based Ensemble approach	C.E.R.T. Drebin	95.08	92.40	94.63	93.50	95.0
6	Mohamad Arif et al. [18]	Fuzzy AHP (Analytical Hierarchy Process)	Drebin and AndroZoo	90.54	-	-	-	-
7	Mazaed Alotaibi and Fawad [31]	A Multifaceted Deep Generative Adversarial Networks approach	AndroZoo and Drebin	96.2	95.1	94.6	94.7	

Table 5. *Cont.*

No.	Author	Method(s)	Dataset	Acc (%)	Prec (%)	Rec (%)	F-m (%)	AUC (%)
8	Atacak et al. [32]	Hybrid approach based on CNN and ANFIS	Drebin and CICMalDroid	94.67	94.78	94.67	94.66	94.87
9	FL-BDE model (Proposed approach)	Mamdani type-Fuzzy Inference System-based Ensemble approach	Drebin and Google Play Store	99.33	98.68	100.00	99.34	99.90

When the studies in Table 5 were compared with each other in terms of the performance metric results, it was found that the studies based on EL, comparative learning and DL were highly successful at malware detection. In comparison to the relevant models, the FL-BDE model, which was proposed as a FIS-based dynamic ensemble approach in our work, demonstrated a more competitive performance. The bi-level malware detection (BMD) model proposed by Jerbi et al. [27] and the Bi-LSTM model using comparative learning suggested by Yang et al. [26] are the other approaches that have performance results that are similar to that of the FL-BDE model. The data in the table make it clearly evident that the proposed model outperformed both of the models in terms of all of the performance metrics. When the performance of the proposed model is compared with the performance of the other models shown in Table 5, it can be seen that it performed much better than these models did.

5. Conclusions

Due to the explosive growth in the current usage of mobile devices, the market share of the Android operating system, which powers these devices, has increased dramatically. Consequently, malware has made them its target. So far, although the research and the developed applications for malware detection by Android application developers have made a partial contribution to the solution of the problem, they have been insufficient to completely eliminate this problem due to some of the characteristics and behavioral features of malware.

In this study, a dynamic model that combines the outputs of ML-based methods through FIS was proposed for Android malware detection. Two thousand application instances in the form of APK files were employed in the study, one thousand of which were malicious applications downloaded from the Drebin database, and one thousand of which were benign applications downloaded from the Google Play Store. The APK files were first analyzed by reverse engineering, and the Manifest.xml file was obtained. Then, the permissions, intentions and activities contained in this file were determined. After that, by querying each APK file, the data vector of 2000×1134 consisting of 1's and 0's was saved to the csv file to be used as a dataset in malware detection.

The experimental results of the proposed model were obtained by applying this dataset to the model whose feature extraction, feature selection, data splitting, multi-classification, voting and routing and combining scores processes were built using the necessary components in the Azure ML environment. The accuracy of this model was also evaluated against the ML-based models, including the SVM, LR, BPM, BDT, DF and NN methods built in this environment. The classification error rates including the MCR, FNR and FPR, the confusion matrix metrics including the accuracy, recall, specificity, precision and F-score and the AUC metric obtained from ROC curves were used for the assessment of the models. The experiments performed in the random split ratio and five-fold cross validation conditions showed that the proposed FL-BDE model outperformed the ML-based models in terms of both the classification error rates and the confusion matrix metrics. With performance differences of 0.5% or more for the 0.60 split conditions and 0.7% or more for the 0.70 split conditions, the proposed model outperformed the ML-based models in terms

of the ROC curves. When the proposed model was compared with similar ensemble-based current literature studies, it was observed that it performed better, with a smaller difference. In comparison with other studies in the literature that produced performance results that are similar to those of our model, it was seen that it had a much better performance in terms of all of the metrics.

In the future, real-time malicious detection applications can be realized by creating a hybrid model that obtains the feature vectors from APK application files with DL-based approaches, and then implementing the malicious application detection process by using the FL-BDE approach we propose here.

Funding: This research received no external funding.

Data Availability Statement: The data presented here are available in the article.

Acknowledgments: The author would like to thank Gazi University Academic Writing Application and Research Center for proofreading the article.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Statista. Smartphone Subscriptions Worldwide 2027. Available online: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/ (accessed on 23 October 2022).
2. Population Division United Nations. World Population Prospects. Available online: https://population.un.org/wpp/ (accessed on 23 October 2022).
3. Statista. Global Mobile OS Market Share. 2022. Available online: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/ (accessed on 23 October 2022).
4. Oh, T.; Stackpole, B.; Cummins, E.; Gonzalez, C.; Ramachandran, R.; Lim, S. Best security practices for Android, BlackBerry, and iOS. In Proceedings of the 2012 the 1st IEEE Workshop on Enabling Technologies for Smartphone and Internet of Things, ETSIoT 2012, Seoul, Republic of Korea, 18 June 2012; pp. 42–47.
5. Felt, A.P.; Finifter, M.; Chin, E.; Hanna, S.; Wagner, D. A survey of mobile malware in the wild. In Proceedings of the ACM Conference on Computer and Communications Security, Chicago, IL, USA, 17 October 2011; pp. 3–14. Available online: https://dl.acm.org/doi/10.1145/2046614.2046618 (accessed on 19 December 2022).
6. Eslahi, M.; Salleh, R.; Anuar, N.B. MoBots: A new generation of botnets on mobile devices and networks. In Proceedings of the ISCAIE 2012—2012 IEEE Symposium on Computer Applications and Industrial Electronics, Kota Kinabalu, Malaysia, 3–4 December 2012; pp. 262–266.
7. Statista. Volume of Detected Mobile Malware Packages. 2021. Available online: https://www.statista.com/statistics/653680/volume-of-detected-mobile-malware-packages/ (accessed on 23 October 2022).
8. Caviglione, L.; Gaggero, M.; Lalande, J.F.; Mazurczyk, W.; Urbański, M. Seeing the unseen: Revealing mobile malware hidden communications via energy consumption and artificial intelligence. *IEEE Trans. Inf. Forensics Secur.* **2016**, *11*, 799–810. [CrossRef]
9. Ahvanooey, M.T.; Li, Q.; Rabbani, M.; Rajput, A.R. A Survey on Smartphones Security: Software Vulnerabilities, Malware, and Attacks. *Int. J. Adv. Comput. Sci. Appl.* **2017**, *8*, 30–45.
10. Das, S.; Liu, Y.; Zhang, W.; Chandramohan, M. Semantics-based online malware detection: Towards efficient real-time protection against malware. *IEEE Trans. Inf. Forensics Secur.* **2016**, *11*, 289–302. [CrossRef]
11. Liu, K.; Xu, S.; Xu, G.; Zhang, M.; Sun, D.; Liu, H. A Review of Android Malware Detection Approaches Based on Machine Learning. *IEEE Access* **2020**, *8*, 124579–124607. [CrossRef]
12. Bulazel, A.; Yener, B. A survey on automated dynamic malware analysis evasion and counter-evasion: PC, Mobile, and Web. In Proceedings of the ACM International Conference Proceeding Series. Association for Computing Machinery, Vienna, Austria, 16–17 November 2017. Available online: https://dl.acm.org/doi/10.1145/3150376.3150378 (accessed on 19 December 2022).
13. Mat, S.R.T.; Ab Razak, M.F.; Kahar, M.N.M.; Arif, J.M.; Mohamad, S.; Firdaus, A. Towards a systematic description of the field using bibliometric analysis: Malware evolution. *Scientometrics* **2021**, *126*, 2013–2055. Available online: https://link.springer.com/article/10.1007/s11192-020-03834-6 (accessed on 19 December 2022). [CrossRef] [PubMed]
14. Li, J.; Sun, L.; Yan, Q.; Li, Z.; Srisa-An, W.; Ye, H. Significant Permission Identification for Machine-Learning-Based Android Malware Detection. *IEEE Trans. Industr. Inform.* **2018**, *14*, 3216–3225. [CrossRef]
15. Onwuzurike, L.; Mariconti, E.; Andriotis, P.; de Cristofaro, E.; Ross, G.; Stringhini, G. MaMaDroid. *ACM Trans. Priv. Secur.* **2019**, *22*, 14. Available online: https://dl.acm.org/doi/10.1145/3313391 (accessed on 19 December 2022). [CrossRef]
16. Venkatraman, S.; Alazab, M. Use of Data Visualisation for Zero-Day Malware Detection. *Secur. Commun. Netw.* **2018**, *2018*, 1728303. [CrossRef]
17. D'Angelo, G.; Ficco, M.; Palmieri, F. Malware detection in mobile environments based on Autoencoders and API-images. *J. Parallel Distrib. Comput.* **2020**, *137*, 26–33. [CrossRef]

18. Arif, J.M.; Ab Razak, M.F.; Tuan Mat, S.R.; Awang, S.; Ismail, N.S.N.; Firdaus, A. Android mobile malware detection using fuzzy AHP. *J. Inf. Secur. Appl.* **2021**, *61*, 102929.
19. Jerlin, M.A.; Marimuthu, K. A New Malware Detection System Using Machine Learning Techniques for API Call Sequences. *J. Appl. Secur. Res.* **2017**, *13*, 45–62. Available online: https://www.tandfonline.com/doi/abs/10.1080/19361610.2018.1387734 (accessed on 19 December 2022). [CrossRef]
20. Wang, S.; Chen, Z.; Yan, Q.; Yang, B.; Peng, L.; Jia, Z. A mobile malware detection method using behavior features in network traffic. *J. Netw. Comput. Appl.* **2019**, *133*, 15–25. [CrossRef]
21. Martín, A.; Lara-Cabrera, R.; Camacho, D. Android malware detection through hybrid features fusion and ensemble classifiers: The AndroPyTool framework and the OmniDroid dataset. *Inf. Fusion* **2019**, *52*, 128–142. [CrossRef]
22. Mat, S.R.T.; Razak, M.F.A.; Kahar, M.N.M.; Arif, J.M.; Firdaus, A. A Bayesian probability model for Android malware detection. *ICT Express* **2022**, *8*, 424–431. [CrossRef]
23. Nguyen, G.; Nguyen, B.M.; Tran, D.; Hluchy, L. A heuristics approach to mine behavioural data logs in mobile malware detection system. *Data Knowl. Eng.* **2018**, *115*, 129–151. [CrossRef]
24. Lu, T.; Wang, J. F2DC: Android malware classification based on raw traffic and neural networks. *Comput. Netw.* **2022**, *217*, 109320. [CrossRef]
25. Amer, E.; El-Sappagh, S. Robust deep learning early alarm prediction model based on the behavioural smell for android malware. *Comput. Secur.* **2022**, *116*, 102670. [CrossRef]
26. Yang, S.; Wang, Y.; Xu, H.; Xu, F.; Chen, M. An Android Malware Detection and Classification Approach Based on Contrastive Lerning. *Comput. Secur.* **2022**, *123*, 1–15. Available online: https://dl.acm.org/doi/10.1016/j.cose.2022.102915 (accessed on 19 December 2022). [CrossRef]
27. Jerbi, M.; Chelly Dagdia, Z.; Bechikh, S.; ben Said, L. Android malware detection as a Bi-level problem. *Comput. Secur.* **2022**, *121*, 102825. [CrossRef]
28. Azad, M.A.; Riaz, F.; Aftab, A.; Rizvi, S.K.J.; Arshad, J.; Atlam, H.F. DEEPSEL: A novel feature selection for early identification of malware in mobile applications. *Future Gener. Comput. Syst.* **2022**, *129*, 54–63. [CrossRef]
29. Taheri, L.; Kadir, A.F.A.; Lashkari, A.H. Extensible android malware detection and family classification using network-flows and API-calls. In Proceedings of the International Carnahan Conference on Security Technology, Chennai, India, 1–3 October 2019.
30. Taha, A.; Barukab, O.; Malebary, S. Fuzzy Integral-Based Multi-Classifiers Ensemble for Android Malware Classification. *Mathematics* **2021**, *9*, 2880. [CrossRef]
31. Mazaed Alotaibi, F.; Fawad. A Multifaceted Deep Generative Adversarial Networks Model for Mobile Malware Detection. *Appl. Sci.* **2022**, *12*, 9403. [CrossRef]
32. Atacak, İ.; Kılıç, K.; Doğru, İ.A. Android malware detection using hybrid ANFIS architecture with low computational cost convolutional layers. *PeerJ Comput. Sci.* **2022**, *8*, e1092. Available online: https://peerj.com/articles/cs-1092 (accessed on 18 December 2022). [CrossRef] [PubMed]
33. Arp, D.; Spreitzenbarth, M.; Hubner, M.; Gascon, H.; Rieck, K.; Internet Society. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. Available online: https://www.scinapse.io/papers/2122672392 (accessed on 19 December 2022).
34. Getting Started with v2. Available online: https://developers.virustotal.com/v2.0/reference/getting-started (accessed on 19 December 2022).
35. MFDroid: A Stacking Ensemble Learning Framework for Android Malware Detection. Available online: https://www.researchgate.net/publication/359593753_MFDroid_A_Stacking_Ensemble_Learning_Framework_for_Android_Malware_Detection (accessed on 14 January 2023).
36. Şahin, D.Ö.; Kural, O.E.; Akleylek, S.; Kılıç, E. A novel Android malware detection system: Adaption of filter-based feature selection methods. *J. Ambient Intell. Humaniz. Comput.* **2021**, 1–15. Available online: https://link.springer.com/article/10.1007/s12652-021-03376-6 (accessed on 14 January 2023). [CrossRef]
37. Abdulla, S.; Altaher, A. Intelligent approach for android malware detection. *KSII Trans. Internet Inf. Syst.* **2015**, *9*, 2964–2983.
38. Altaher, A.; Barukab, O. Android malware classification based on ANFIS with fuzzy c-means clustering using significant application permissions. *Turk. J. Electr. Eng. Comput. Sci.* **2017**, *25*, 2232–2242. Available online: https://www.researchgate.net/publication/317343365_Android_malware_classification_based_on_ANFIS_with_fuzzy_c-means_clustering_using_significant_application_permissions (accessed on 14 January 2023). [CrossRef]
39. Feng, P.; Ma, J.; Sun, C.; Xu, X.; Ma, Y. A novel dynamic android malware detection system with ensemble learning. *IEEE Access* **2018**, *6*, 30996–31011. Available online: https://www.researchgate.net/publication/325612491_A_Novel_Dynamic_Android_Malware_Detection_System_With_Ensemble_Learning (accessed on 14 January 2023). [CrossRef]
40. Rustam, Z.; Kristina, A.L.; Satria, Y. Comparison between Fisher's Ratio and Information Gain with SVM classifier for 3 levels of enthusiasm classification through face recognition. *J. Phys. Conf. Ser.* **2021**, *1752*, 012042. Available online: https://www.researchgate.net/publication/349326381_Comparison_between_Fisher\T1\textquoterights_Ratio_and_Information_Gain_with_SVM_classifier_for_3_levels_of_enthusiasm_classification_through_face_recognition (accessed on 14 January 2023). [CrossRef]
41. Rajagopal, S.; Kundapur, P.P.; Hareesha, K.S. Towards Effective Network Intrusion Detection: From Concept to Creation on Azure Cloud. *IEEE Access* **2021**, *9*, 19723–19742. [CrossRef]

42. Yetginler, B.; Atacak, İ. Sentiment Analyses on Movie Reviews using Machine Learning-Based Methods. *Artif. Intell. Stud.* **2020**, *3*, 1–12. [CrossRef]
43. Cortes, C.; Vapnik, V.; Saitta, L. Support-vector networks. *Mach. Learn.* **1995**, *20*, 273–297. Available online: https://link.springer.com/article/10.1007/BF00994018 (accessed on 19 December 2022). [CrossRef]
44. Schölkopf, B.; Smola, A.J. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*; The MIT Press: Cambridge, MA, USA, 2018. Available online: https://direct.mit.edu/books/book/1821/Learning-with-KernelsSupport-Vector-Machines (accessed on 19 December 2022).
45. Lindström, J. Predictive Maintenance for a Wood Chipper using Supervised Machine Learning. 2018. Available online: http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-149304 (accessed on 19 December 2022).
46. Hung, Y.H. Investigating how the cloud computing transforms the development of industries. *IEEE Access* **2019**, *7*, 181505–181517. [CrossRef]
47. Syed, A.H.; Khan, T. Machine learning-based application for predicting risk of type 2 diabetes mellitus (t2dm) in saudi arabia: A retrospective cross-sectional study. *IEEE Access* **2020**, *8*, 199539–199561. [CrossRef]