Universiteit Leiden

Master Computer Science

Explainable AI in Android Malware Detection

Name:	Shihang Yu
Student ID:	s3240924
Date:	[23/05/2024]
Specialisation:	Artificial Intelligence
1st supervisor:	Olga Gadyatskaya
2nd supervisor:	Nusa Zidaric
3rd supervisor:	Rui Li

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

1. Abstract

In recent years, the open source nature of the Android platform has made it a prime target for malicious applications. It is difficult for users to distinguish between benign and malicious applications. To bridge this gap, machine learning models are applied to Android malware detection. However, since these models often lack interpretability, it is difficult for malware analysts to understand the model decision-making process, making it difficult to judge the reliability of models. XAI methods show good promise, allowing analysts to explain the model decision-making process, thereby increasing trust in the prediction system. Although previous researches have applied a variety of local XAI techniques to Android malware analysis, global explanation methods have not been systematically studied in this direction, especially feature-based global XAI techniques. In addition, there is a lack of comprehensive comparison and evaluation of the performance and explanation effects of such methods on different black-box models, as well as their application in security scenarios.

In this study, we aim to measured the quality of XAI methods for Android malware detection. We validated the four local explanation methods-LIME, SHAP, EDC and Anchors based on the consistency rate (CR) and soundness rates (SR) metrics proposed by Soulaimani [78], and formulated three metrics for feature-based global explanation methods: stability, robustness, and effectiveness to evaluate the four global explanation methods-ALE, PD, PDV and Morris methods. The experimental results show that among the local explanation methods, the SHAP method has the best soundness rate than the other four methods, which is 97.41%; the LIME method has the highest consistency rate of 69.94%. Among the global explanation methods, the Partial Dependence method (PD) has the best stability, which is 93.97%, the Morris method has the highest robustness of 95%, and the Accumulated Local Effects (ALE) method has the best effectiveness, which is 100%. Our results provide malware analysts with more comprehensive, transparent, and effective insights into explanation method selection.

Keywords: Interpretability, Android malware detection, local explanation, global explanation

Contents

2. Introduction

With the development of smart devices, Android system has gradually dominated the global market. According to statistics released by Statista[1], Android smartphones accounted for more than 70% of the market in 2024, and Android also accounted for approximate 44% of tablets. As of June 2023, there are nearly 2.6 million Android applications available for download in the largest app store, Google Play Store. The success of the Android system stems from its open source platform, which greatly enhances the user experience. However, this openness also increases security risks. Various malicious Android applications have emerged, becoming one of the biggest security threats to the current mobile internet. Malware (Malicious software) refers to any code or applications installed on a device without the user consent with the purpose of harming the device system or violating the user rights. Malware can cause a range of harms, including data leakage and unauthorized surveillance [79]. Distinguishing between safe and harmful applications can be a challenge for users due to their sophisticated disguises. To cope with this situation, researchers have invested considerable efforts in developing innovative techniques for analyzing malware.

Android malware detection methods based on machine learning (ML) have proven to be a relatively effective detection technology. This method can detect malware types that have never been seen before and can provide better detection performance and efficiency [64]. Machine learning-based Android malware detection also has some limitations, with the main challenge being the black-box nature of the model as the decision-making process is not transparent or understandable to humans. This opacity makes it difficult to fully trust and verify test results [39]. Additionally, machine learning models require extensive and representative training data to be effective, and malware evolves rapidly. Therefore, these models quickly become outdated, requiring constant updating and retraining [35], [21]. Moreover, methods such as adversarial strategies can be used to evade detection, which only requires minor changes to the malware samples to effectively evade detection [51], [23], [25]. Bostani and Moonsamy [19] also proposed EvadeDroid, a problem-space adversarial attack, and achieved an average evasion rate of 79% on five popular commercial antiviruses. Given these challenges, it becomes increasingly difficult for malware analysts to judge the effectiveness of detection models in real-world applications.

To solve the problem of lack of transparency in the decision-making process of black-box models, researchers have increasingly turned to explainable AI (XAI) methods. XAI method allow analysts and users to understand the complex, difficult to explain results produced by a black-box model, thereby enhancing trust in model decisions and detection results [90]. Applying XAI methods to cybersecurity can effectively improve the transparency and interpretability of cybersecurity-related decision models. That said, there are domain-specific requirements when applying XAI methods to malware detection. For example, security analysts want to obtain consistent explanations across similar applications and different runs of the model [56]. Therefore, XAI methods proposed for malware detection have to meet these specific requirements. There are already some specific quantitative metrics have been

[1] https://www.statista.com/topics/876/android/topicOverview

proposed to address this problem, such as stability, robustness and effectiveness [30], [87].

Explanation methods can be divided into local explanation methods and global explanation methods. Local explanations focus on the decision rationale of a single instance, i.e., why a single Android application is labeled as benign or malicious, while global explanation methods focus on analyzing the decision process of the model on the entire dataset. Although previous research has made significant strides in applying local XAI techniques to Android malware detection and evaluating their performance, global explanation methods remain underexplored in the security domain. To address this gap, Li and Gadyatskaya [56] proposed three quantitative metrics—stability, robustness, and effectiveness, and evaluated five state-of-the-art rule-based XAI techniques within the context of Android malware detection. However, feature-based XAI techniques in security still lack any dedicated global evaluation metrics or a systematic framework tailored to their interpretability assessment.

In this paper, we focus on post-hoc XAI methods for black-box models for Android malware detection and evaluates its performance. Our main study and findings can be divided into two parts as follows:

1. For local explanation methods, we experiment with four widely used local explanation methods, LIME [69], Anchors [70], SHAP [60] and Explaining Data-Driven Document Classifications (EDC) [61] as described in Section 3.2, using four traditional black-box ML methods, SVM, random forest, XGBoost and MLP we have discussed in Section 3.1, and evaluate their performance using two metrics, consistency rate and soundness rate formulated by Soulimani [78]. Our experimental results show that LIME has the highest consistency rate of 69.94%, while SHAP has the highest soundness rate of 97.41%.
2. To fill the gap in the evaluation of feature-based global explanation methods, we formulated three metrics based on important features: stability, robustness, and effectiveness to evaluate the four global explanation methods we describe in the 3.2 section, namely ALE [7], partial dependence (PD) [33], PD variance [37], and Morris method [62]. Our experimental results show that the global explanation methods PD achieves the highest stability of 93.97%, Morris method has the highest robustness of 95% and ALE obtained the best effectiveness of 100%.

Our results cover both local and global explanation methods and can provide more comprehensive, transparent, and effective insights for malware analysts to choose appropriate explanation methods.

The rest of this paper is structured as follows: Section 3 briefly introduces some classic models that can be used for Android malware detection classification tasks and the explanation methods used in this paper. Section 4 introduces some related works. The evaluation metrics we selected for local and global explanation methods are presented in Section 5. Section 6 mainly describes the experimental setup and related methods. Section 7 presented our experiment results. The overall conclusion, limitations and future work presented in Section 8.

3. Background

This section briefly introduces some classification models that are widely used for Android malware detection as well as some knowledge of local and global explanation methods.

3.1. Classification Algorithms

In machine learning (ML), classification and regression are two core tasks in supervised learning. Supervised learning means that the training data contains input vectors and their corresponding target vectors [41]. The difference between classification and regression lies in the type of output variables. Specifically, the prediction result of the classification task is a qualitative output, that is, a discrete variable prediction. The regression task is a quantitative output, which is a continuous variable prediction [18].

In Android malware detection, sample labels are usually labeled as benign or malicious, so it can be regarded as a binary classification problem in machine learning. As we described in Section 2, some models have high accuracy, but their decision-process is difficult to explain, while some models have high interpretability but low accuracy. Figure 3.1 shows the trade-off between interpretability and accuracy for some relevant machine learning models.

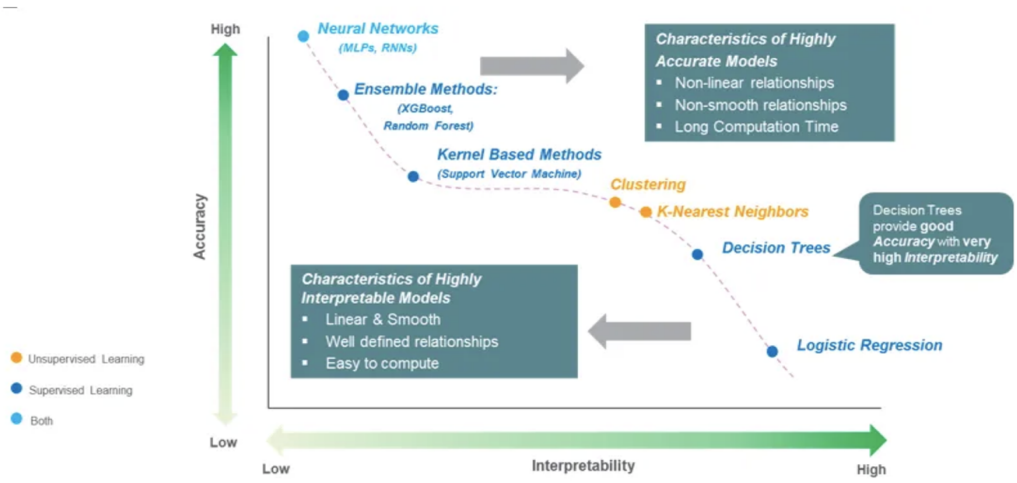


Figure 3.1: The trade-off between interpretability and accuracy of some ML models [68].

In this section, we mainly discuss some classification algorithms that are commonly used in Android malware detection with high accuracy but low interpretability.

3.1.1 Support Vector Machine (SVM)

Support vector machine [17] is a widely used supervised learning model mainly used for classification and regression analysis. Although SVM was originally designed for binary classification problems, it can also be used for multi-classification problems and regression tasks. In a binary classification problem, the goal of SVM is to find a hyperplane (a straight line in two-dimensional space and a plane or hyperplane in higher dimensions) to distinguish data points of different categories. SVM attempts to determine the optimal hyperplane by maximizing the separation between two categories of data points. Support vectors are those data points closest to the hyperplane. These points are key elements when the model determines the optimal hyperplane, as they directly affect the size of the interval [18]. When the data is not linearly separable, SVM maps the data into a higher-dimensional space by using kernel functions (such as RBF kernel, polynomial kernel, etc.), making the data become linearly separable in this new space. This approach enables SVM to handle more complex nonlinear relationships.

SVM has strong generalization ability, especially in high-dimensional data. With a suitable kernel function, SVM can perform well on complex and non-linearly separable data sets. However, choosing the appropriate kernel function and parameters (such as C parameters and gamma parameters) has a great impact on the performance of SVM. And when processing large data sets, the training time of SVM may be longer [63].

3.1.2 Random Forest

Decision tree (DT) [72] is a tree-structured supervised learning method that can be used in machine learning (ML) for both classification and regression tasks. The goal of DT is to create a model that is able to learn and derive simple decision rules by analyzing data. These rules will be used to identify key variables in the data characteristics so that effective predictions can be made on these variables. Usually, a decision tree contains a root node, several internal nodes and several leaf nodes. Among them, the root node is the starting point of the decision-making process and contains the entire data set. Each internal node tests the corresponding feature attributes in the items to be classified and decides the output branch. Leaf nodes represent decision results. Decision trees are the basis for many advanced tree-structured models.

The core idea of random forest [20] is to ensemble the prediction results of multiple decision trees to improve the performance of the overall model. Random forest adopts bagging idea, as shown in figure 3.2, each tree is trained independently, and during the training process, the new training set is formed by random taking n training samples from the training set with replacement each time, and using the new training set, train and obtain many sub-models. This is also the meaning of "random" in random forest. For classification tasks, random forest uses a voting mechanism, and the final classification results is the sub-model with the most votes. Due to the independence of each tree, even if some trees predict inaccurately, other trees can correct the errors, thereby improving the overall accuracy and fault tolerance of the model. Random forest has excellent performance in classification tasks.

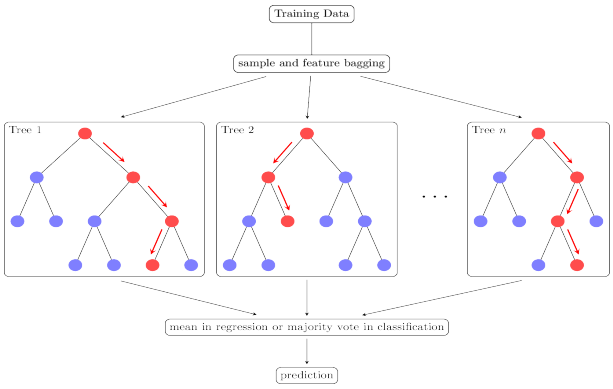


Figure 3.2: The concept of a random forest classifier [71].

3.1.3 Extreme Gradient Boosting (XGBoost)

Extreme gradient boosting [22] is an tree ensemble machine learning algorithm widely used in regression and classification tasks. It is an implementation of the gradient boosting [33] framework based on decision trees (GBDT). The core idea of GBDT is to use boosting to combine a set of weak classifiers into a strong classifier. The final prediction of GBDT is the weighted sum of all tree predictions, that is, each iteration uses the error residual of the previous model to fit the next model to gradually reduce the prediction error.

XGBoost optimizes and improves upon GBDT in several ways. For example, XGBoost adds a regularization term to the objective function to control the complexity of the model, which helps reduce model variance and prevent overfitting. Additionally, XGBoost supports column subsampling and row subsampling, which simplifies calculations without traversing all features. Moreover, XGBoost can automatically handle missing values and support parallel computing. Figure 3.3 shows a simplified structure of XGBoost.

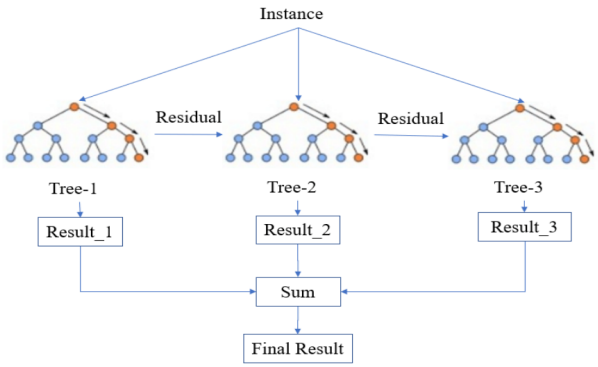


Figure 3.3: A simplified structure of XGBoost [85].

3.1.4 Multilayer Perceptron (MLP)

A multilayer perceptron [73] is a feedforward artificial neural network (ANN) that can be used to solve a variety of machine learning problems. MLP consists of multiple layers of neurons, usually an input layer, hidden layer(s), and an output layer. Each layer is fully connected, meaning that each neuron in the previous layer is connected to all neurons in the next layer. The input layer is used to receive data, and the hidden layer, which usually

consists of a single or multiple layers, is used to perform nonlinear transformations on the input and output the results by the output layer. The figure 3.4 shows an example of a MLP.

Therefore, the parameters of the MLP are the connection weights and biases between each layer. During training process, the parameters of the model are updated by backpropagation, and the final output is completed by forward propagation. The purpose of backpropagation is to minimize the loss function, and the weights and biases are updated by calculating the gradient of the loss function for each parameter using the gradient descent method. During the forward propagation process, the input data passes through the neurons of each layer in turn, and the output result is finally obtained after weighted summation and activation function transformation.

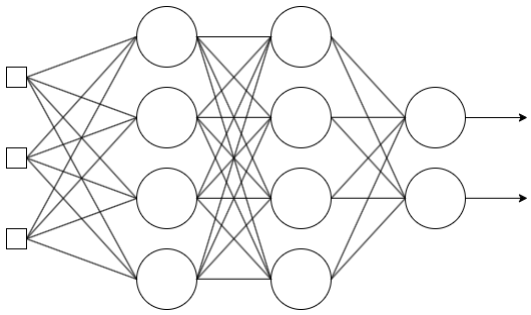


Figure 3.4: MLP with three inputs, two hidden layers, and two outputs [80].

3.1.5 Deep Neural Network (DNN)

Deep Neural Network (DNN) is composed of multiple layers of Artificial Neural Networks (ANNs) [2], where each layer consists of multiple nodes. A DNN contains an input layer, hidden layer(s) and an output layer, and each hidden layer consists of multiple neurons. The input layer is used to accept raw data and is the first layer of the DNN. The output layer is used to generate the final prediction results. In classification tasks, the number of neurons in the output layer usually corresponds to the number of categories in the problem. The hidden layer is the core concept of DNN and is usually composed of one or more layers. Each hidden layer consists of multiple neurons, each of which receives the output of the previous layer and performs a nonlinear transformation through an activation function.

Forward propagation and back propagation are two important steps in the training process of DNNs. The forward propagation of DNN is similar to that of the perceptron, which uses the input of the previous layer to calculate the output of the next layer. Backpropagation adjusts network parameters by calculating the error feedback between the predicted value and the true value, and backpropagates layer by layer to update the weights and biases. Its purpose is to minimize the loss function through optimization algorithms [36].

DNN can effectively handle complex pattern recognition tasks through multiple hidden layers and is widely used in many fields such as image recognition and audio processing. However, due to its complex internal structure, the decision making process of the model is difficult to explain.

3.1.6 Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN) [65] is a type of neural network, which is particularly powerful in the field of image classification. The architecture of a CNN usually includes an input layer, convolutional layers, pooling layers, fully connected layers and an output layer. Among them, the input layer is used to receive the pixel values of the image. In the convolution layer, an important concept is the convolution kernel, also called a filter. The convolution kernel is a weight matrix, usually of a fixed size. The convolution kernel slides over the region of the input image, a process also called convolution. The convolution process can extract important features of the local region. When the convolution kernel passes through all the regions, the important features of the input image, also called feature maps, can be extracted. can be extracted. The feature maps are then passed to the pooling layer [26]. The purpose of the pooling layer is to reduce the number of parameters and the complexity of computation by reducing the data dimension. The input data passes through the convolutional layers and the pooling layers and is then input into the fully connected layers. The structure of the fully connected layer is similar to the fully connected layer structure of the DNN mentioned in Section 3.1.5. However, in CNN, the feature map processed by the convolution layer and the pooling layer must be flattened into a one-dimensional vector before entering the fully connected layer. Finally, the results are output according to different tasks.

The activation functions are also the core component of a CNN. The activation functions can effectively solve nonlinear problems and increase the expression ability of CNN [86]. Commonly used activation functions include sigmoid function [16], tanh function [58], ReLU function [29], etc.

Due to the powerful functions and excellent performance of CNN, it is also widely used in the field of malware detection [46]. However, the complexity of its internal structure also makes the decision-making process of the model lack of transparency and difficult for analysts to understand.

3.2. Model Explanation Methods

The interpretability of machine learning can be divided into different types according to different criteria [24]. As described in Section 2, this paper mainly focuses on post-hoc XAI methods. Post-hoc explanation requires pre-training the model and analyzing its results to understand why the model makes decisions, so it can also be understood as an explanation step after the prediction [48]. Moreover, post-hoc explanation methods can be divided into model-agnostic methods or model-specific methods. Model-agnostic means that researchers can use any model to explain, so this method is more flexible. On the other hand, model-specific explanation methods are limited to specific model types [24]. This section focuses on model-agnostic interpretation methods. Post-hoc model explanation methods [10] including local explanation and global explanation methods. Figure 3.5 shows an example of global and local explanation. This section introduces some classifiers of local explanation and global explanation.

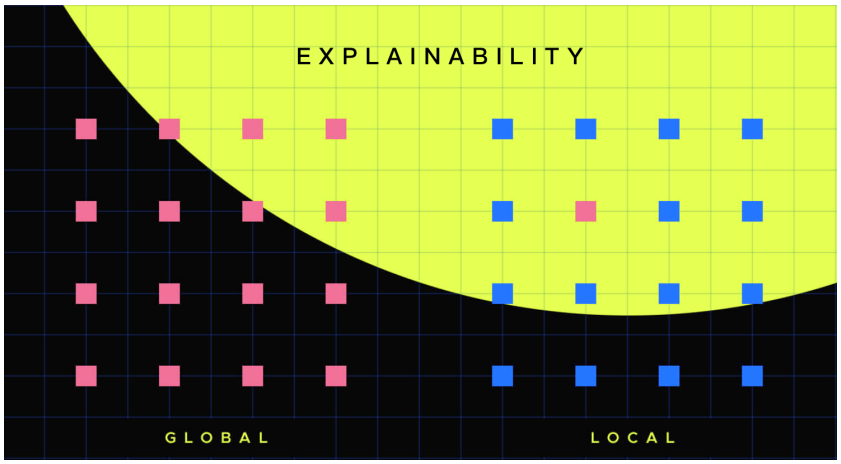


Figure 3.5: A simple example showing the difference between global and local explanation methods. Local explanations focus on a single instance, while global explanations focus on the entire dataset. The pink color graph in the figure represents the sample instance being explained [66].

3.2.1 Local Explanation Methods

Local explanation methods usually focus on explaining a specific instance, i.e., why the model makes a certain prediction for this instance.

Local Interpretable Model-agnostic Explanations (LIME)

LIME [69] is a **local** explanation method proposed by Ribeiro et al., applicable to any machine learning model of regression and classification tasks. The core idea of LIME is to train a local surrogate model to predict a single instance [24]. Specifically, for an instance that needs to be explained, perturbations are first performed around the sample point to generate new perturbation samples, and then obtain the prediction value based on the corresponding black box model. Subsequently, LIME trains an interpretable model, such as a decision tree, on the new perturbation samples, so that it can well approximate the prediction behavior of the black box model in the local area rather than the global area.

LIME first locally perturbs the target data point x whose prediction result is to be explained to obtain a sample set Z. The predictions of the model are observed, and then weights are assigned according to the distance between the perturbed data point and the original data point. The closer the instance to the explanation is, the greater the weight assigned. LIME gives the following explanation:

$$\xi(x) = \arg\min_{g \in G} \mathcal{L}(f, g, \pi_x) + \Omega(g) \tag{3.1}$$

where:
G: a class of potentially interpretable models,

f: primitive models of higher complexity, i.e. a random forest model,
$f(x)$: the probability that x belongs to a certain class,
π_x: the similarity weight with the explained instance x,
$\pi_x(z)$: proximity measure between an instance z to x, so as to define locality around x, $z \in Z$
$\mathcal{L}(f, g, \pi_x)$: measure of how unfaithful g is in approximating f in the locality defined by π_x,
$\Omega(g)$: a measure of complexity of the explanation $g \in G$.

In equation 3.1, $\Omega(g)$ is used to control the complexity of the model to avoid the model being too complex and difficult to explain, and the function $\mathcal{L}(f, g, \pi_x)$ is used to measure the difference between the complex model f and the simple model g. Locality is captured by π_x. When the value of $\Omega(g)$ is low enough, the results are easy for humans to interpret. At this time, we minimize the $\mathcal{L}$ function to get the explanations.

LIME represents the local behavior of the model in a linear manner, such that $g(z') = w_g \cdot z'$. The distance between samples before and after the perturbation is defined as equation 3.2:

$$\pi_x(z) = exp(-D(x, z)^2/\sigma^2) \tag{3.2}$$

where D is distance function and σ is width. While $\pi_x(z)$ is an exponential kernel defined on D with σ. The D for text and image are cosine distance and L2 distance respectively. Thus, the locality-aware loss L can be expressed as equation 3.3:

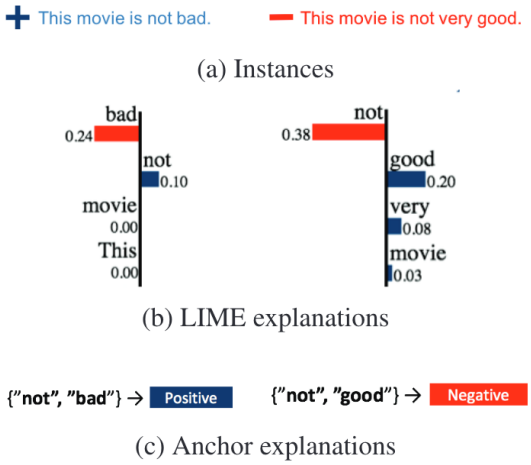
$$\mathcal{L}(f, g, \pi_x) = \sum_{z, z' \in Z} \pi_x(z)(f(z) - g(z'))^2 \tag{3.3}$$

where $f(z)$ are sample of instances around x' by drawing non-zero elements of x' uniformly at random. For tabular data, LIME samples are collected from the mass center of the training data [24].

Anchors

Anchors, which is a high-precision model-agnostic explanations, also purposed by Ribeiro et al. (2018) [70]. It is a **local** explanation method. Anchors addresses the main drawback of the LIME method (presented in Section 3.2.1), which is to represent the local behavior of the model in a linear manner [49]. The explanation results of the Anchors method are based on IF-THEN rules, also called anchors. Simply, it is to find a set of rules, when these rules are all satisfied, the predicted results are almost always the same. Figure 3.6 (c) shows two examples of sentiment predictions made with LSTM (Long Short-term Memory [43], a deep learning model that excels at processing sequence data) and explained with Anchors. The rule can be expressed as IF this sentence contains "not bad", THEN the interpretation result is likely to be predicted as positive. IF a sentence contains "not good", THEN the prediction result is likely to be negative. In LIME, as shown in figure 3.6 (b), although the analysis of the two sentences is accurate, users cannot understand when "not" is positively or negatively affected. This is because the interpretation result on the left does not apply

to the right ("not" has a positive impact on the left sentence, but a negative impact on the right sentence).



(a) Instances

(b) LIME explanations

(c) Anchor explanations

Figure 3.6: Two examples of sentiment predictions made with LSTM and explained with LIME and Anchors. Figure (a) shows two sentence examples, Figure (b) shows the explanation result of LIME and Figure (c) shows the explanation result of Anchors [70].

The definition of the rule can be defined as equation 3.4:

$$\mathbb{E}_{D(z|A)}[\mathbb{1}_{f(x)=f(z)}] \geq \tau, A(x) = 1. \tag{3.4}$$

where:
x: an instance, $x \in X$,
f: a black box model,
$f(x)$: the individual prediction for instance x,
A: a rule of set of predicates,
$D(z|A)$: conditional distribution on sample z when the rule A applies,
τ: desired level of precision.

Given an instance x and a rule A, the consistency rate between samples z and x that satisfy rule A must reach an precision of at least τ. Also, if $A(x) = 1$, and A is a sufficient condition with high probability for $f(x)$, then A is an anchor. According to equation 3.4, the A can be defined as:

$$prec(A) = \mathbb{E}_{D(z|A)}[\mathbb{1}_{f(x)=f(z)}] \tag{3.5}$$

After generating a set of candidate rules A, each subsequent iteration will generate a new candidate rule, and the best rule is selected by comparing the precision. Multi-armed Bandit [47] is used to estimate the precision. However, it is difficult to directly calculate the precision for any D and f, thus, A needs to satisfy the constraint of $prec(A) \geq \tau$ with high probability. The equation show as 3.6:

$$P(prec(A) \geq \tau) \geq 1 - \delta \tag{3.6}$$

When there are multiple A meet the conditions, the anchor should be selected with the largest coverage.

Local Rule-based Explanations (LORE)

Local Rule-based Explanations (LORE) [38] is a **local** explanation method proposed by Guidotti et al. that can be applied to any type of machine learning model. LORE also generates explanations based on rules, which are extracted from neighborhood data through an interpretable surrogate model, usually a decision tree. In addition, LORE also generates counterfactual rules that explain how the predictions of models change based on changing the features of the instance. LORE is suitable for black-box result explanations of relational tabular data and is based on a genetic algorithm [44] to explore decision boundaries near data points. In the neighborhood generation process, a set of N neighbor instances Z is generated for the instance x to be explained through a genetic algorithm, with the goal of making it close to the features of instance x to achieve the local decision behavior of the black box b. $Z = Z_{=} \cup Z_{\neq}$ is instances containing two decision values, that is, when instances $z \in Z_{=}$, $b(x) = b(z)$ and when $z \in Z_{\neq}$, $b(x) \neq b(z)$. The selection, crossover, and mutation operations of the genetic algorithm can evolve the initial instance population to make it close to the target. The fitness function is used to select the best result for shaping the next generation. The generation of $Z = Z_{=} \cup Z_{\neq}$is completed by maximizing the fitness function, which is defined as follows:

$$fitness^{x}_{=}(z) = I_{b(x)=b(z)} + (1 - d(x, z)) - I_{x=z}$$
$$fitness^{x}_{\neq}(z) = I_{b(x)\neq b(z)} + (1 - d(x, z)) - I_{x=z}$$

where:
d : a distance function,
I: $I_{true} = 1$ and $I_{false} = 0$,
$1 - d(x, z)$: instances z similar to instances x,
$I_{x=z}$: $I_{x=z} = 1, x = z$; $I_{x=z} = 0, x \neq z$.

The fitness functions find instances z that are similar to x but not equal to x, for the first fitness function, b produces the same result as x, while for the second fitness function, b returns a different decision.

After neighborhood generation process, LORE constructs an interpretable decision tree model, aiming to simulate the local behavior of the black box in the neighborhood. After the neighborhood generation process, LORE constructs an interpretable decision tree model that aims to simulate the local behavior of the black box in the neighborhood. For the decision tree c, the interpretation obtained is $e = <r, \Phi>$. Among them, r is the decision tree rule and Φ is the counterfactual rule. The rule extracted from the decision tree is $r = (p \rightarrow y)$, where p is the conjunction of the conditions, y is the decision result, and the rule r is formed by the path matched by x. Counterfactual rules are extracted by

identifying paths in a decision tree that lead to different decisions than the original ones, for each path in the decision tree that results in a decision $\hat{y} \neq y$, we consider it as a potential counterfactual rule. The counterfactual rule is expressed as $q \rightarrow \hat{y}$, q represent the conjunction of split conditions along such a path. If the conditions in q are met, the prediction would change to $\hat{y}$. The set of counterfactual rules consists of the paths that do not satisfy x the least.

Kernel Shapley Additive Explanations (SHAP)

Kernel SHAP (SHapley Additive exPlanations) proposed by Lundberg and Lee (2017) [60], can be used for both **local** and **global** explanations [50]. It is a a model-agnostic interpretability method that suitable for tabular data of any machine learning classification and regression model. Kernel SHAP is explained using the concept of Shapley value from cooperative game theory [74]. In cooperative game theory, Shapley value refers to the fair distribution of the total expenditures received by players in the alliance to the players [24]. Therefore, Shapley value can be used in machine learning to calculate the contribution of the model to the feature value of a single prediction. The classic Shapley value estimation for feature i can be defined as equation 3.7:

$$\phi_i = \sum_{S \subseteq F \backslash \{i\}} \frac{|S|!(|F| - |S| - 1)!}{|F|!} \left[f_{S \cup \{i\}} \left(x_{S \cup \{i\}} \right) - f_S \left(x_S \right) \right]. \tag{3.7}$$

where:
F: the set of all features,
S: the subset of F,
$F \backslash \{i\}$: F not include i,
$f_{S \cup \{i\}}$: model prediction when i is included in S,
X_S: the values of the input features in S.

Among them, $f_{S \cup \{i\}} \left(x_{S \cup \{i\}} \right) - f_S \left(x_S \right)$ is the marginal contribution and the term of $\frac{|S|!(|F|-|S|-1)!}{|F|!}$ represents the the weighting factor to ensure all possible subsets have fairly weight [76].

SHAP combined linear LIME (presented in Section 3.2.1) and Shapley value. In SHAP, the Shapley value interpretation is expressed as a linear model. SHAP specifies the interpretation as:

$$g(z)' = \phi_0 + \sum_{j=1}^{M} \phi_j^{z'} \tag{3.8}$$

where:
g: explanation model,
M: maximum simplified features size,
z': simplified feature vectors, $z' \in \{0, 1\}^M$,

ϕ_j: Shapley values, $\phi_{\{}j\} \in R$.

Lundberg and Lee also proposed three desirable properties of SHAP: local accuracy, missingness and consistency. Local accuracy describes the matches between the original model and explanation model, it can be formulated as:

$$f(x) = g(x') = \phi_0 + \sum_{i=1}^{M} \phi_i x'_i \tag{3.9}$$

where f denotes the original model and x is the input; g denotes the explanation model and x' is the simplified input. The desirable property of missingness means that missing features are imputed with a value of zero, that is, $x'_i = 0 \Rightarrow \phi_i = 0$, when $x'_i = 0$, feature have no attributed effect. Consistency means that if the model changes, the marginal contribution of the feature value and Shapley value increases or remains the same, regardless of other features. For model f and model f', z'_1 denote setting $z'_i = 0$ for $fx(z') = f(hx(z'))$ and z'_1.

$$f'_x(z') - f'_x(z' \backslash i) \geq f_x(z') - f_x(z' \backslash i) \tag{3.10}$$

for all inputs $z' \in \{0,1\}^M$, then $\phi_i(f', x) \geq \phi_i(f, x)$.

Kernel SHAP approximates Shapley values through kernel functions (Shapley kernel [60]) and weighted linear regression. The kernel function can be defined as equation 3.11 [24]:

$$\pi_x(z') = \frac{(M-1)}{\binom{M}{|z'|} |z'| (M - |z'|)} \tag{3.11}$$

where:
M: maximum coalition size,
$|z'|$: the number of present features in instance z'.

Kernel SHAP can effectively increase sampling efficiency. The minimum and maximum coalitions will take more weight to avoid blind adoption, so better Shapley value estimates can be obtained.

Explaining Data-Driven Document Classifications (EDC)

The EDC algorithm is a local explanation method proposed by Martens and Provost [61]. The core purpose of this method is to understand the decision-making process of document classifiers. Since textual data usually contains a large number of words, it is usually high-dimensional. Therefore, the EDC method is suitable for some high-dimensional and sparse data. The EDC method explains the classification decision by finding a minimal set of words in a document. The minimal set of words needs to satisfy two requirements: removing all words in the set will change the predicted class, and no other subset of the word set has the same effect. Formally, given a document vocabulary with m words and let μ_E denotes a mask vector to be a binary vector of length m, where each element of

the vector corresponds to a word in the vocabulary. Let E be an explanation represented by a mask vector μ_E with $\mu_E(i) = 1 \Leftrightarrow w_i \in E$, otherwise, $\mu_E(i) = 0 \Leftrightarrow w_i \notin E$. $D \setminus E$ is the Hadamard product of the feature vector of document D with the one's complement of μ_E, denotes the result of removing the words in E from document D. Thus, finding a mask vector μ_E such that $C_M(D \setminus E) \neq C_M(D)$ means finding the minimum explanation. However, if any bit of μ_E is set to zero to form E', then $C_M(D \setminus E') = C_M(D)$.

Martens and Provost proposed Search for Explanations for Document Classification (SEDC) algorithm to find such minimal explanation. The aim of SEDC is to find minimal explanations in a reasonable amount of time. SEDC uses a direct heuristic best-first search method, first listing all potential explanations for a word, and then calculating the category and score changes for each word. In other words, the algorithm removes words from the potential explanations separately. If the predicted category and score change after removing a single word, it means that the word constitutes a complete explanation and can be directly added to the explanation set. Otherwise, it means that the word cannot constitute a complete explanation and needs to be combined with other words for expansion. Subsequently, the algorithm selects the words that have the greatest impact on the classification, that is, the words with the greatest confidence reduction, as the best expansion candidates among all the words that cannot constitute an explanation alone. The expansion method combines candidate combinations with other words in the document to form new candidate subsets, removes and reclassifies each new combination, and updates the classification results and model output scores. This process is repeated until new explanations that can be added to the explanation set appear. In order to control the computational complexity, SEDC introduces a pruning strategy, that is, once a word combination is determined to be an explanation, there is no need to expand other combinations. In addition, the algorithm can prevent infinite expansion by setting a maximum number of iterations.

Contrastive Explanation Method (CEM)

Contrastive Explanation Method (CEM) is a **local** explanation method proposed by Dhurandhar et al. [28] in 2018. This method helps understand the decision-making process of the model by generating two explanations for the classification model: Pertinent Positives (PP) and Pertinent Negatives (PN). PP represents the minimum features required for the model to predict the same label as the original instance. In other words, PP means the minimum necessary conditions required for the model to make the current classification decision. PN represents the minimum number of features that must be missing for the model to make the current prediction. In other words, these features cannot be included for the model to maintain the original prediction category, otherwise the prediction result will change.

The CEM method transforms the finding PP and PN into a disturbance variable optimization problem. For finding PP, the optimization problem can be formulated as follows:

$$\min_{\boldsymbol{\delta} \in \mathcal{X} \cap \mathbf{x}_0} c \cdot f_\kappa^{\text{pos}}(\mathbf{x}_0, \boldsymbol{\delta}) + \beta \|\boldsymbol{\delta}\|_1 + \|\boldsymbol{\delta}\|_2^2 + \gamma \|\boldsymbol{\delta} - \text{AE}(\boldsymbol{\delta})\|_2^2 \tag{3.12}$$

where:

x_0: a natural example;
$\mathcal{X}$: feasible data space, $x_0 \in \mathcal{X}$;
$\boldsymbol{\delta}$: a perturbation applied to x_0;
t_0: class label obtained from classification model;
β, γ: regularization coefficients.

The aim of this formula is to find a minimum perturbation $\boldsymbol{\delta}$ so that the model does not change the current predicted category. The loss function $f_\kappa^{\text{pos}}(\mathbf{x}_0, \boldsymbol{\delta})$ can be defined as:

$$f_\kappa^{\text{pos}}(\mathbf{x}_0, \boldsymbol{\delta}) = \max\left\{\max_{i \neq t_0}[\text{Pred}(\boldsymbol{\delta})]_i - [\text{Pred}(\boldsymbol{\delta})]_{t_0}, -\kappa\right\} \tag{3.13}$$

By minimizing the loss function, we can find the minimum set of features that support the decision without changing the classification result.

For finding PN, the optimization problem can be formulated as:

$$\min_{\boldsymbol{\delta} \in \mathcal{X}/\mathbf{x}_0} c \cdot f_\kappa^{\text{neg}}(\mathbf{x}_0, \boldsymbol{\delta}) + \beta\|\boldsymbol{\delta}\|_1 + \|\boldsymbol{\delta}\|_2^2 + \gamma\left\|\mathbf{x}_0 + \boldsymbol{\delta} - \text{AE}(\mathbf{x}_0 + \boldsymbol{\delta})\right\|_2^2 \tag{3.14}$$

The loss function $f_\kappa^{\text{neg}}(\mathbf{x}_0, \boldsymbol{\delta})$ is defined as:

$$f_\kappa^{\text{neg}}(\mathbf{x}_0, \boldsymbol{\delta}) = \max\left\{[\text{Pred}(\mathbf{x}_0 + \boldsymbol{\delta})]_{t_0} - \max_{i \neq t_0}[\text{Pred}(\mathbf{x}_0 + \boldsymbol{\delta})]_i, -\kappa\right\} \tag{3.15}$$

Different from PP, the goal of PN optimization problem is to find the minimum perturbation that causes the model to change its classification results. By minimizing the loss function of PN, we can find the minimum set of features that support the decision changing the classification result.

Counterfactual Instances (CF)

Counterfactual instances (CF) is a **local** explanation method proposed by Wachter et al. (2017) [83]. Counterfactual explanations aim to enhance the transparency and interpretability of decisions and help understand the decision-making process of complex models without revealing the details of the internal models. Counterfactual explanations are based on causal reasoning, that is, "if X had not happened, Y would not have happened" [49]. A simple example, assuming that android.permission.CAMERA is an important feature, android.permission.CAMERA = 1 means that the APK has requested camera permission. For an APK sample labeled as malware, the counterfactual explanation can be "If the APK did not request permission to open the camera (android.permission.CAMERA = 0), then the APK would likely be labeled as benign". The generation of counterfactuals can be divided into two parts. The first step is to minimize the objective function by finding the optimal set of weights w, which is how most standard classifiers in machine learning are implemented. This process is defined in Equation 3.16:

$$\arg\min_{w} \ell(f_w(x_i), y_i) + \rho(w) \tag{3.16}$$

where:
y_i: the label for data point x_i;
$\rho(w)$: a regularizer over weights w.

Then we need to minimize the distance between the original instance x_i and the counterfactual instance x'. The counterfactual instance needs to be as close to the original point as possible so that $f_w(x') = y'$, where y' is the new target. This process requires to hold weight w.

$$\arg\min_{x'} \max_{\lambda} \lambda \left(f_w\left(x'\right) - y'\right)^2 + d\left(x_i, x'\right) \tag{3.17}$$

where $d(x_i, x')$ denotes a distance function, usually using L_1 norm or Manhattan distance, is used to measure the distance between the original data point x_i and the counterfactual x'. Maximizing λ makes $f_w(x')$ close to the new target y' while maintaining the similarity with the original data point x_i as much as possible.

Prototype Counterfactuals (ProtoCF)

Prototype Counterfactuals (ProtoCF) is a **local** explanation method proposed by Looveren & Klaise [82] in 2021. The method is based on the counterfactual approach (presented in Section 3.2.1) but accelerates finding interpretable counterfactual explanations for classifier predictions by using class prototypes. The ProtoCF method improves the problem that the generation method in the traditional counterfactual method may generate instances that are unrealistic or do not conform to the data distribution by introducing the prototype of the target category.

ProtoCF can also be transformed into an optimization problem, which is to find the smallest input feature change compared to the original input, so that the model prediction result is transformed from the original category to the target category, while the generated counterfactual instance is close to the prototype of the target class. Its loss function can be defined as:

$$\text{loss} = cL_{\textbf{pred}} + \beta L_1 + L_2 + L_{\text{AE}} + L_{\text{proto}} \tag{3.18}$$

This loss function is similar to the CEM method (presented in Section 3.2.1), but adds the reconstruction error of the L_2 perturbed examples as a loss term to make the counterfactuals close to the training data distribution. In addition, the L_{proto} loss term attempts to minimize the distance between the L_2 counterfactuals and the nearest prototype. The reason for adding L_{proto} is that L_{AE} does not necessarily obtain interpretable solutions or speed up counterfactual search.

3.2.2 Global Explanation Methods

Global explanations focus on the entire dataset and explain why the model made its decision.

Partial Dependence (PD)

Partial Dependence (PD) focuses on the changes in one or two feature values affect the model predictions, while average the values of other features effects, is a **global** method of visualizing linear, monotonic or other relationships between labels and features [33]. In regression tasks, the calculation of the linear correlation between each feature and label can be defined as 3.19:

$$f_S(x_S) = \mathbb{E}_{x_C}[f(x_S, X_C)] = \int f(x_S, X_C) d\mathbb{P}(X_C) \tag{3.19}$$

where:
f: a black-box model,
F: a set of all features,
S: a set of features of interest, $S \subseteq F$.
C: complement, $C = F \backslash S$.
X: feature space,
X_C: random variables, features in set C,
x_S: features realizations in set S,
$\mathbb{P}(X_C)$: the probability distribution/measure for features set C.

The set S usually contains one or two features whose impact on prediction is expected to be explored. The principle of PD is to marginalize the output of the model to the feature distribution in the set C to reflect the relationship between the features in the set S and the model prediction. We approximate the integral in equation 3.19 by computing the average of the reference dataset. The PD of a set S can be approximated as:

$$f_S(x_S) = \frac{1}{n} \sum_{i=1}^{n} f(x_S, x_C^{(i)}) \tag{3.20}$$

where $x_C^{(i)}$ denotes the feature values. The meaning of this formula is to calculate the marginal impact of specific values of features in S on prediction.

In classification tasks, the output is the impact of features on category prediction probability.

Partial Dependence Variance (PD Variance)

Partial Dependence Variance (PD Variance) is a **global** explanation method proposed by Greenwell et al. in 2018 [37]. This method relies on PD (presented in Section 3.2.2), specifically, PD variance is a method for calculating global feature importance and feature interaction, which is obtained by calculating the variance in the PD method. For feature importance, the larger the fluctuation of the PD curve of a feature, the higher the variance, indicating that the feature is more important. Conversely, a flat PD curve corresponds to

a lower variance, indicating that the feature has less impact. The flatness measurement of PD can generally be defined as:

$$i(x) = F(\bar{f}_s, (z_s)) \tag{3.21}$$

where $F(\cdot)$ denotes any measure of the flatness of $\bar{f}_s, (z_s)$. Importance measure for predictor x_1 can be defined as:

$$i\left(x_1\right) = \begin{cases} \sqrt{\frac{1}{k-1}\sum_{i=1}^{k}\left[\bar{f}_1\left(x_{1i}\right) - \frac{1}{k}\sum_{i=1}^{k}\bar{f}_1\left(x_{1i}\right)\right]^2} & \text{if } x_1 \text{ is continuous} \\ \left[\max_i\left(\bar{f}_1\left(x_{1i}\right)\right) - \min_i\left(\bar{f}_1\left(x_{1i}\right)\right)\right]/4 & \text{if } x_1 \text{ is categorical} \end{cases} \tag{3.22}$$

where the flatness measure is the sample standard deviation for continuous features and the range statistic for factors with K levels divided by four for categorical features. The reason of dividing the range by four estimates the standard deviation for small to moderate sample sizes.

Accumulated Local Effects (ALE)

The Accumulated Local Effects (ALE) method proposed by Apley and Zhu [7], is utilized for computing feature effects. The algorithm can be used for classification and regression models of tabular data and provides a model-independent **global** explanation for black-box models. As mentioned in Section 3.2.2, ALE also describes how features affect the average prediction. However, PD has an obvious disadvantage that it will make wrong predictions when there is a strong correlation between features. ALE improves this defect. ALE calculates the average of the predicted changes and accumulates it on a grid. The formula of ALE [24] is shown below:

$$\begin{aligned} \hat{f}_{S,ALE}\left(x_S\right) &= \int_{z_{0,S}}^{x_S} E_{X_C|X_S=x_S}\left[\hat{f}^S\left(X_s, X_c\right) \mid X_S = z_S\right] dz_S - constant \\ &= \int_{z_{0,S}}^{x_S}\left(\int_{x_C} \hat{f}^S\left(z_s, X_c\right) d\mathbb{P}\left(X_C \mid X_S = z_S\right) d\right) dz_S - constant \end{aligned} \tag{3.23}$$

where:
X_S: explained features,
X_C: other features,
z_S: the variable value of feature S during the integration process,
$E_{X_C|X_S=x_S}$: the expectation of feature C with $X_S = x_S$,
$d\mathbb{P}\left(X_C \mid X_S = z_S\right)$: when $X_S = z_S$, probability distribution of C.

A constant is subtracted to make the average effect of the data zero. Integral is used to solve the problem of feature correlation. This formula 3.23 calculates the average of the predicted changes, also defined as the gradient, but since not all models include a gradient, ALE uses an approximate algorithm to estimate the value of the function:

$$\hat{\tilde{f}}_{j,ALE}(x) = \sum_{i:x_j^{(i)} \in N_j(k)} [\hat{f}(z_{k,j}, x_{-j}^{(i)}) - \hat{f}(z_{k-1,j}, x_{-j}^{(i)})] \tag{3.24}$$

where:
$\hat{f}$: the prediction function of the model,
z: grid values,
$z_{k,j}$: the upper boundary value of feature j in the kth interval,
$z_{k-1,j}$: the lower boundary value of feature j in the kth interval,
$N_j(k)$: the set of data points in the kth interval of feature j,
$x_{-j}^{(i)}$: features except j in data i.

The process of calculating the prediction difference can be seen from formula 3.24, this represents the local effect of the feature on the model prediction within the interval. Summing these differences gives us the local cumulative impact of the feature within the interval. Dividing the sum by the number of instances in the interval gives us the average difference in predictions for the interval. When the mean effect is zero, this effect is located at the center:

$$\hat{f}_{j,ALE}(x) = \hat{\tilde{f}}_{j,ALE}(x) - \frac{1}{n}\sum_{i=1}^{n} \hat{\tilde{f}}_{j,ALE}(x_j^{(i)}) \tag{3.25}$$

After removing the average effect of each effect, a positive feature effect indicates that the feature has a positive impact on the prediction result, otherwise it has a negative impact.

Permutation Importance (PI)

Permutation Importance (PI) is a **global** explanation method first proposed by Breiman [20] in 2001 and then refined by Fisher et al. [32] in 2019. PI is a method for tabular data that calculates feature importance by the degree of decrease in model performance when the feature values are permuted. PI was originally proposed for random forest classifiers. Out-of-bag samples (OOB) are generated during random forest training. After each tree is created, the value of a feature is randomly permuted to disrupt the relationship between the feature and the truth label, generating a new set of OOB data. The new data is input to the current tree to obtain a new prediction result and saved. The process is repeated for all features, and the noisy response is compared with the true label to obtain the error rate. The importance of a feature is represented by the percentage increase in the misclassification rate after permutation relative to the OOB rate when all features remain unchanged.

Then, this was subsequently extended by Fisher et al. to a model-independent version of permutation feature importance, also known as model reliance. For a model f, model reliance can be informally written as:

$$MR(f) = \frac{\text{Expected loss of } f \text{ under noise}}{\text{Expected loss of } f \text{ without noise}} \tag{3.26}$$

Model reliance can be formally defined as follows:
Let $Z^{(a)} = (Y^a, X_1^a, X_2^b)$ and $Z^{(b)} = (Y^b, X_1^b, X_2^b)$ be independent random variables, each of them should be following the same distribution $Z = (Y, X_1, X_2)$, where $Z = (Y, X_1, X_2) \in \mathcal{Z}$ be an independent and identically distributed (iid) random variable, $Y \in \mathcal{Y}$ be the outcome, $X = (X_1, X_2) \in \mathcal{X}$ are features, two subset $X_1 \in \mathcal{X}_1$ and $X_2 \in \mathcal{X}_2$ can be multivariate. The expected value of the loss after the feature subset X_1 is permuted is defined as follows:

$$e_{\text{switch}}(f) = \mathbb{E}[L\left\{f, (Y^{(b)}, X_1^{(a)}, X_2^{(b)})\right\}] \tag{3.27}$$

where the pair $(Y^{(b)}, X_2^{(b)})$ from $Z^{(b)}$ and $X_1^{(a)}$ from $Z^{(a)}$. The standard expected loss shown in equation 3.28 used for comparison is the loss without the feature values switched:

$$e_{\text{orig}}(f) = \mathbb{E}[L\left\{(Y, X_1, X_2)\right\}] \tag{3.28}$$

Thus, model reliance can be formally defined as their ratio:

$$MR(f) = \frac{e_{\text{switch}}(f)}{e_{\text{orig}}(f)} \tag{3.29}$$

The degree of model reliance on feature subsets can be assessed by comparing the loss changes before and after permutation. If the loss increases significantly after permutation, it means that the feature has an important impact on the model, otherwise the reliance is weak.

Morris Sensitivity Analysis (Morris method)

Morris sensitivity analysis, also known as Morris method, is a **global** explanation method proposed by Morris [62]. Morris method is a one-factor-at-a-time (OAT) analysis method, that is, only one factor is changed at a time to evaluate the impact of input variables on model output, and is useful for screening input variables to determine the relative impact of each feature. Morris method based on elementary effect (EE), in a grid input space, this method captures the changes that occur to the sampled trajectory points. The EE of ith input factor can be defined as:

$$d_i(x) = \frac{[y(x_1, ..., xi - 1 + \Delta, x_{i+1}, ..., x_k) - y(x)]}{\Delta} \tag{3.30}$$

where:
Δ: step, usually a multiple of $1/(p-1)$,
p: number of levels in input space,
x: vectors representing the input variables,
y: outputs.

Morris method has two important indicators, mean (μ): $\mu = \sum_{i=1}^{r} d_i/r$, used to assess the overall importance of input factors to the model output; and standard deviation (σ): $\sigma = \sqrt{\sum_{i=1}^{r} d_i(d_i - \mu)^2/r}$, used to account for non-linear effects or interactions between factors. A low standard deviation means that the output is linear in this factor, and a high standard deviation means that the indicator is non-linear in this factor. Where r is number of trajectories. Then, Saltelli et al. [75] describe another metric μ^* to address the robustness issues of non-monotonic models: $\mu^* = E[\psi(Y \mid X_{-i})]$, where ψ is an operator that performs absolute local variation.

3.3. Overview of XAI Methods

In Section 3.2 we discussed some explanation methods. To be brief, this section mainly summarizes these methods and their names, abbreviations, and brief descriptions so that they can be used as references in subsequent sections. The details are shown in Table 3.1:

Full name	Abbreviation	Description
Local Rule-based Explanations	LIME	Local
Anchors	Anchors	Local
Local Rule-based Explanations	LORE	Local
Kernel Shapley Additive Explanations	(Kernel) SHAP	Local / Global
Explaining Data-Driven Document Classifications	EDC	Local
Contrastive Explanation Method	CEM	Local
Counterfactual Instances	CF	Local
Prototype Counterfactuals	ProtoCF	Local
Partial Dependence	PD	Global
Partial Dependence Variance	PD Variance (PDV)	Global
Accumulated Local Effects	ALE	Global
Permutation Importance	PI	Global
Morris Sensitivity Analysis	Morris	Global

Table 3.1: A summary of some XAI methods discussed in this paper, including their full name, abbreviation and their description (local or global).

4. Related Work

Android malware detection methods can generally be divided into static detection, dynamic detection and hybrid detection technology [53], [57]. The difference between static detection and dynamic detection is whether the application is in a running state, and hybrid detection combines static detection with dynamic detection. Android malware detection methods based on machine learning (ML) have been proven to be one of the more effective detection methods at present, and can usually achieve a high accuracy in malware detection [4]. For Android applications, static features are mainly extracted by analyzing APK (application package) files, such as permissions, Dalvik opcodes and other components, etc. The analysis objects of dynamic features usually include network traffic, battery usage, CPU utilization, IP address, opcode, etc [57]. Some popular machine learning models and algorithms in Android malware detection methods include Decision Trees (DT), Naive Bayesian (NB), Linear Model (LM), Support Vector Machine (SVM), K-Nearest Neighbor (KNN), K-means Clustering Algorithm, Neural Networks (NN) and Deep Learning (DL) [57].
For static analysis, Lou et al. [59] proposed TFDroid, a SVM-based approach to detect malware, which incorporates topics and sensitive data streams as features and achieved 93.65% accuracy in identifying malicious applications. Tiwari and Shukla [81] proposed a Android malware detection technique based on optimized permissions and API. They generated two types of features, common features and features with low variance removed, and evaluated them on five ML classification models: SVM, DT, KNN, NB and Logistic Regression (LR). This method achieved the highest accuracy on LR, with 97.25% and 95.87% respectively. Li et al. [55] proposed a malware detection system for Android mobiles based on deep convolutional neural networks (DCNN). This method used an optimized DCNN to learn the original opcode sequences extracted from decompiled Android files and achieved an accuracy of 99%.
Dynamic analysis is often executed and tested in a sandbox. Yang et al. [92] proposed DroidWard, an Android malware detection technique based on dynamic analysis. The method runs applications with SVM, DT, and random forest to extracts the most relevant and effective features to improve the detection accuracy of malicious applications. The accuracy of this method for detecting malicious applications is 98.54%. Ananya et al. [11] proposed SysDroid, which designed a new feature selection method, selection of relevant attributes for improving locally extracted features using classical feature selectors (SAILS), tested and evaluated it using multiple different machine learning models, Logistic Regression (LR), Classification and Regression Tree (CART), random forest (RF), eXtreme Gradient Boost (XGBoost) and Deep Neural Networks (DNN). The accuracy of SysDroid in dynamic analysis of Android malware detection can reach up to 99%.
For hybrid analysis, Feng et al. [31] proposed a two-layer DL approach for Android malware detection. The first layer is used to extract static features and detect malware in combination with a fully connected neural network, and the results are passed to the second layer. The second layer uses a new method CACNN (cascades CNN and AutoEncoder) to detect malware through the network traffic characteristics of APPs. Bakir [15] proposed TuneDroid in 2025, a novel Android malware detection technology based on hybrid analysis. This method used Bayesian Optimization (BO) to adjust the pre-trained CNN model

to improve model performance and achieved 98% accuracy on the test set.

ML for Android malware detection can only classify applications as benign or malicious, but cannot explain the reasons for such classification. This is due to the black-box nature of ML models. XAI-based malware detection methods can effectively make up for this shortcoming [89], making ML-based methods more credible. XAI methods can be divided into intrinsic explanation methods and extrinsic (post-hoc) explanation methods according to the connectivity with the machine learning (ML) model [56]. Intrinsic explanation methods are those in which the ML model used is interpretable, so this method is based on specific models. Extrinsic explanation methods can be applied to any model, but require training data. As described in Section 3, post-hoc XAI methods can be divided into local explanation methods and global explanation methods according to the scope of explanation. Local explanations focus on a single apk and explain why an application is labeled as benign or malicious, while global explanations focus on the entire dataset and analyze why the model made its decision.
The most classic intrinsic method is Drebin [8], which combines static analysis and linear SVM models to classify malware and provide explanations to users. The Traffic AV [84] method combines network traffic analysis with DT algorithm to detect and analyze malware, while also providing users with analysis and explanation of the final decision.

In addition to intrinsic methods, more and more post-hoc XAI methods have also been adapted and applied to Android malware detection. Some of the explanation methods we described in Section 3.2 have been widely used to explain ML-based Android malware detection. For example, Fan et al. [30] have applied five widely used post-hoc local interpretable methods, LIME, SHAP, Anchors, LORE, and LEMNA [40] to Android malware detection. Galli et al. [34] proposed an XAI framework for Behavioral Malware Detection (BMD, analyze software behavior at runtime) systems, aiming to evaluate the applicability of four local XAI methods, SHAP, LIME, Layer-wise Relevance Propagation (LRP) [12] and attention mechanism [14] into malware detection systems. Smmarwar et al. [77] proposed XAI-AMD-DL, an Android malware detection framework based on a hybrid CNN and Bi-GRU (Bi-Gated Recurrent Unit, similar to LSTM but more simplified) model, and the Kernel SHAP method is integrated in the framework to provide local interpretability. Ambekar et al. [6] proposed TabLSTMNet, an Android malware classification model integrating the attention of TabNet (an interpretable deep learning architecture specialized for tabular data that uses a sequential attention mechanism to select features to infer at each decision step) mechanism and long short-term memory (LSTM, a recurrent neural network) architecture. The classification model combines LIME and SHAP to explain the contributions of local and global features, respectively. Kulkarni and Stamp [54] apply five XAI methods, LIME, SHAP, PDP, ELI5 (global explanation methods, shuffle the permutations of the feature values. The worse the classification results are after a feature is shuffled, the more the model relies on that feature) and Class Activation Mapping (CAM, a model-specific technique for explain CNNs, class-based localization, use the gradient information flowing into the last convolutional layer to obtain the localization of important regions in the image to determine the importance of each pixel of the input image for a given class), to classical ML (SVM, RF and KNN) and DL (MLP, CNN) models for Android malware classification and obtain local and global explanation results.
Moreover, there are some techniques based on leveraging semantic information or key code

snippets to provide explainability results. For instance, Qian et al. [67] proposed LAMD, the first Android malware detection technique based on LLMs (Large Language Models). This method filters irrelevant functions and analyzes application behavior step by step through key context extraction and hierarchical code reasoning, effectively capturing structural dependencies and semantics, and providing final predictions and explanations. Wu et al. [89] proposed XMAL, a XAI based ML method for Android malware detection. XMAL is divided into a classification phase and an explanation phase. The classification phase combines MLP and attention mechanisms to find the most critical features, and the explanation phase automatically generates neural language descriptions to explain malicious behaviors within the application. Korine et al. [52] proposed DAEMON, a dataset and platform-agnostic explainable malware classifier. DAEMON uses multi-stage mining to gradually retain relatively small features that are most effective in classifying malware families. These features have rich semantic information, such as readable strings or malicious code snippets, so the classification process is more transparent and easy to understand.

However, although XAI-based Android malware detection methods provide analysts with a more intuitive basis for judgment, their explanation results still face many challenges. For example, Abusitta et al. [1] believe that many XAI technologies lack robustness and stability, which leads to users or analysts weakening their trust in the interpretation results. Fan et al. [30] also proposed that some XAI methods that explain predictions by providing important features generally fail to reach a consensus on the interpretation results obtained in the malware analysis domain. Besides, some existing metrics for evaluating XAI methods, such as fidelity [93] and Faithfulness [5] etc., have not been applied to the security field, but in malware detection scenarios, it may be necessary to meet the needs of specific fields or introduce security-specific metrics.
To close this gap, Fan et al. [30] formulated three metrics, stability, robustness and effectiveness for local interpretability results in the field of Android malware detection to evaluate their explanation results. Soulimani [78] proposed two metrics, consistency rate and soundness rate based on Fan et al. Consistency rate used to measure the degree of variation in the explanation of the results using different classification methods for the same feature set, and soundness rate refers to the degree to which adding irrelevant features to the feature set explains the variation in the explanation results. The details of consistency rate and soundness rate will be explained in section 5.1. For global interpretability evaluation metrics in Android malware detection field, Li and Gadyatskaya [56] proposed three metrics also based on stability, robustness, and effectiveness to evaluate rule-based global explainable Android malware detection methods. Our work is based on Li and Gadyatskaya and formulated three evaluation metrics for the explanation results of global XAI methods that extract important features. The details of our work will be explained in section 5.2.

5. Evaluation Metrics

This section describes in detail two metrics for evaluating local explanation methods and three metrics for evaluating global explanation methods.

5.1. Local Explanation Methods

Local explanation methods aim to focus on the decision-making process of a single instance, and the methods have been described in detail in Section 3.2. To evaluate these methods, this section will analyze two metrics for local explanation methods.

5.1.1 Consistency Rate (CR)

The consistency rate (CR) [78] is a metric that can be used to calculate the extent of change in the explanatory features obtained when different classification algorithms are used on the same feature set. Let $\mathcal{C}$ denote a set of n classification models, $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n\}$, and the corresponding set of n explanations is represented by $\mathcal{S} = \{\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_n\}$, where $\mathcal{E}_i$ is the set of features generated using the classification model $\mathcal{C}_i$. We defined $\mathcal{Z} = \bigcup_{i=1}^{n} \mathcal{E}_i$ to represent all the individual features generated by the interpretations in $\mathcal{S}$. Thus, the formula for CR can be expressed as:

$$CR = \sum_{i \in Z} \frac{\sum_{s \in S} \frac{f(i,s)}{|S|}}{|Z|} \tag{5.1}$$

With:

$$f : Z \times S \Rightarrow [0, 1]$$
$$f(i, s) = \begin{cases} 1, & \text{i} \in \text{s} \\ 0, & \text{i} \notin \text{s} \end{cases}$$

Where:
$f(i, s)$: determine whether feature i exists in feature set S. If it exists, $f(i, s)$ returns 1, otherwise returns 0,
$|S|$: total number of classifiers,
$|Z|$: total number of features.

Specifically, the formula first sums the number of occurrences of each feature i in all explanation sets S, that is, check whether feature i exists in each explanation set s, set s is an explanation generated by a different classification model for each set, $s \in S$. Then accumulate all existences ($f(i, s) = 1$ or $f(i, s) = 0$). The result is then normalized by dividing it by the total number of explanation sets $|S|$ to obtain the average frequency of

feature i in different classification model explanations. Therefore, we can use the part of the formula $\sum_{s \in S} \frac{f(i,s)}{|S|}$ as a subformula to represent the feature representation on the classifier. After obtaining the average frequency of all features, the value is summed and divided by the total number of feature sets $|Z|$ to normalize again to calculate the consistency rate (CR). An example is used to further illustrate the calculation process of the consistency rate of this indicator:
Consider an explanation method S and two classification models C_1 and C_2. The method generates an explanation feature set $s_1 = \{a, b, c\}$ under the C_1 model and an explanation feature set $s_2 = \{b, c, f\}$ under the C_2 model. Feature a only appears in s_1, feature f only appears in s_2, so the frequency of occurrence of feature a and feature f is $\frac{1}{2}$, and feature b and feature c appear in both s_1 and s_2, so the frequency of occurrence is $\frac{2}{2}$. By calculating $s_1 \cup s_2$, we can get the total number of features $|Z|$ to be 4. Thus, according to equation 5.1, the value of $CR = \frac{\frac{1}{2}+\frac{2}{2}+\frac{2}{2}+\frac{1}{2}}{4} = 0.75$.

The higher the CR value, the closer the selection of explanatory features between different classification models is, and the lower the CR value, the greater the difference in the selection of explanatory features between different classification models. The range of CR value is $[0, 1]$, this means that ideally, the value of CR should be 1, that is, using different classification models for the same explainer, the resulting explanation features are exactly the same. On the contrary, the CR value of $\frac{1}{|S|}$ is the worst case, which means that the original feature set resulting in $\sum_{s \in S} f(i, s) = 1$, that is, each feature appears in only one explanation set and not in the others. The worst case indicates that the explanation results between the classification models are completely inconsistent. A higher CR value means that the explanation feature set generated by the explanation method under different classification models is more consistent, indicating that the explanation method is more reliable. A low CR value indicates that the explanation method is less reliable.

5.1.2 Soundness Rate (SR)

Soundness rate (SR) [78] is a metric that can be used to evaluate the extent of change in explanatory features obtained when an irrelevant dataset (features filtered out from the original dataset during the data preprocessing process) is added to the original dataset. Therefore, this metric requires comparing two different explanation feature sets: the interpretation results obtained using the original feature set and the interpretation results obtained by adding irrelevant features to the original feature set.

The irrelevant feature set needs to meet the property requirement of not contributing to the application classification. Specifically, a feature is only a part of the application used to describe a specific behavior, and different feature combinations lead the classification model to distinguish it as a malicious or benign application. Adding irrelevant features to an application only means adding some behaviors to the program, and the behavior does not affect the prediction results of classification models of the program as malware or benign based on other feature combinations. Therefore, irrelevant features can be regarded as noise, and they will not have a substantial impact on the application classification. Irrelevant features are features with the lowest variance obtained by calculating the variance

of features between applications in the dataset during feature extraction. These features usually do not change much and will not affect the classification results. This property can be formulated as the following:

$$Exp(c_i, F) = Exp(c_i, F \cup S) \tag{5.2}$$

where:
$Exp(.)$: explanation function,
c_i: $c_i \in C^2$, where C is the set containing all classification models,
F: the original feature set,
S: a set of irrelevant features.

The soundness rate (SR) can be defined as follows:
Let $\mathcal{F}_1$ be the original feature set and $\mathcal{F}_2$ be the feature set with irrelevant features added to the original feature set $\mathcal{F}_1$, make $\mathcal{F}_1 \cap \mathcal{F}_2 = \emptyset$. Then, let $\mathcal{S}_1$ and $\mathcal{S}_2$ be the explained set features generated by the explanation method using a certain classification model $\mathcal{C}_i$ and the feature sets $\mathcal{F}_1$ and $\mathcal{F}_2$, respectively. Irrelevant features are those whose variance is 0 during feature selection process, and the rest are relevant features. Based on formula 5.2, the explanation function using the original dataset $\mathcal{F}_1$ can be expressed as $Exp(c_i, F) = S_1$, and the explanation function of the dataset $\mathcal{F}_2$ with irrelevant features added can be expressed as $Exp(c_i, F_1 \cup F_2) = S_2$. Furthermore, let $\mathcal{R}_{irr}$ and $\mathcal{R}_{or}$ be the set of irrelevant features and the original features present in $\mathcal{S}_2$, respectively. So, $S_2 = R_{or} \cup R_{irr}$ with $R_{or} \subseteq F_1$ and $R_{irr} \subseteq F_2$.

Thus, the soundness rate (SR) can be defined as equation 5.3:

$$SR = 0.25 \cdot (1 - \frac{|\mathcal{R}_{irr}|}{|\mathcal{F}_2|}) + 0.75 \cdot CR \tag{5.3}$$

while $\frac{|\mathcal{R}_{irr}|}{|\mathcal{F}_2|}$ describes the ratio of generated features to added features. The more irrelevant features exit in the explanatory feature set, the closer this ratio is to 1. The addition of CR is to take into account the impact of the original feature set after adding irrelevant features. In other words, after adding irrelevant features, the addition of CR can retain the core features of the original feature set. When calculating SR, Z in CR 5.1 is $\{\mathcal{R}_{or} \cup \mathcal{S}_1\}$, making $\mathcal{S} = \{\mathcal{R}_{or}, \mathcal{S}_1\}$, this allows CR to be used only to measure the feature set.

An example to explain the principle of SR: assume that $\mathcal{F}_1 = \{a, b, c, d, e\}$ is the original feature set, $\mathcal{S}_1 = \{b, d, e\}$ is the explained feature set obtained after using a certain explanation method based on $\mathcal{F}_1$, and $\mathcal{F}_2 = \{x, y, z\}$ is an irrelevant feature set with a variance of 0 during the feature selection process. By adding $\mathcal{F}_2$ to $\mathcal{F}_1$ we get $\mathcal{F}_1 \cup \mathcal{F}_2 = \{a, b, c, d, e, x, y, z\}$, and $\mathcal{S}_2 = \{b, d, y\}$ is the explained feature set obtained by using $\mathcal{F}_1 \cup \mathcal{F}_2$, where $\mathcal{R}_{or} = \{b, d\}$ and $\mathcal{R}_{irr} = \{y\}$. According to equation 5.3 we can get the value of SR is about 0.79.

A higher SR value indicates that a certain explanation method has a higher soundness

rate when using different classification models. Ideally, the value is 1, which means that the explanation method is not affected by irrelevant features at all. A lower SR value means that the explanation method is extremely sensitive to irrelevant features. In the worst case, the SR value is 0.375 ($0.25 \cdot 0 + 0.75 \cdot \frac{1}{2} = 0.375$), which means that the explanation feature set is completely composed of irrelevant features and the core features are all replaced.

5.2. Global Explanation Methods

Global explanation methods aim to focus on a set of instances to explain the impact of features on the overall prediction results of the model, and the methods have been described in detail in Section 3.2.

The meanings expressed by important features are different in local and global explanation methods. As mentioned in Section 5.1, the important feature of local explanation methods is to reveal which features contribute most to the prediction result of a single input instance. Therefore, different input individuals will generate different important features. However, in global explanation methods, important features are used to evaluate their overall impact on the predictions made by the model, rather than being limited to the predictions of a single instance. In other words, the important features generated in global explanation methods are those whose changes should significantly affect the output of the model across all samples in the dataset.

Based on the above understanding, this section will analyze three feature-based metrics to evaluate some global explanation methods.

5.2.1 Stability (stb)

Stability metric are designed to verify the extent to which the generated explanation results vary between multiple runs [30], that is, whether the generated explanation results are consistent between multiple runs.

Stability can formally defined by the following:
Let $M = \{m_1, m_2, m_3, ..., m_n\}$ be a set of classification models and E be a global explainer. Each model m_i generates a set of explanation results through the explainer E, the top_k important features was selected in this result and represent it as r_i. Thus, for each model m_i, we have $r_i = E(m_i)$. After T runs, this result can be expressed as $r_i^t = E(m_i^t)$, where $t \in \{T\}$. According to the context of stability, for each model m_i, we aim to measure the consistency of the explanation results r_i^t of the model across T runs, that is, the similarity of the top_k important features in the explanation results of a model after multiple runs. This similarity can be measured using the Dice coefficient. The reason for using Dice coefficient is that this method gives higher weight to the overlapping parts. The similarity between two feature sets $r_i^{(t_1)}$ and $r_i^{(t_2)}$ can be expressed as 5.4:

$$D(r_i^{(t_1)}, r_i^{(t_2)}) = \frac{2 \times |r_i^{(t_1)} \cap r_i^{(t_2)}|}{|r_i^{(t_1)}| + |r_i^{(t_2)}|} \tag{5.4}$$

where $r_i^{(t_1)}$ and $r_i^{(t_2)}$ are the top_k important feature sets generated by the model m_i through the interpreter E in two different runs t^1 and t^2.
The average similarity of model m_i in all runs can be calculated by averaging the Dice coefficients between all different run pairs. For T runs, there are $\binom{T}{2}$ possible run pairs, so the average similarity of model m_i in T runs can be expressed as 5.5:

$$sim(m_i, T) = \frac{2}{T(T-1)} \sum_{1 \le t_1 < t_2 \le T} D(r_i^{(t_1)}, r_i^{(t_2)}) \tag{5.5}$$

where $\frac{2}{T(T-1)}$ is the number of different run pairs $\binom{T}{2}$ of model m_i in T runs.
Thus, for model set M, by calculating the average similarity of each model in all runs, the overall stability of the model set can be obtained. The stability formula is as follows 5.6:

$$stb(M, T) = \frac{1}{n} \sum_{i=1}^{n} sim(m_i, T) \tag{5.6}$$

where n is the number of models in M.

The higher the stability value, the higher the similarity of the interpretation results (top_k features) after multiple runs of the global interpretation method, which also means that the interpretation method has high stability. Ideally, this value is equal to 1. On the contrary, a lower stability value means that the interpretation results of the interpretation method after multiple runs are quite different, which proves that the interpretation method has poor stability.

5.2.2 Robustness (rob)

Robustness refers to the ability of an explanation method to remain unaffected when slight perturbations are applied to the input [30].

Robustness can formally defined by the following:
Let $M = \{m_1, m_2, m_3, ..., m_n\}$ be a set of classification models and E be a global explainer. For a dataset d, each model m_i generates a set of explanation results through the explainer E, the top_k most important features was selected in this result and represent it as r_i. Thus, for each model m_i, we have $r_i = E(m_i, d)$. After r_i is generated, We randomly select a set of not the most important features s_i with the same number as r_i from the remaining features (i.e., all features except r_i) so that

$$|s_i| = |r_i| \quad \text{and} \quad s_i \cap r_i = \emptyset,$$

The reason for avoiding choosing the least important features for perturbation is that if the perturbation of important features obtains lower robustness, we cannot judge whether it

is because the explanation method itself is not robust or because the change of important feature values destroys key information, resulting in a large difference in the explanation results. We add perturbations to the feature values corresponding to s_i and generates a new dataset d'. We use the new dataset d' to generate new perturbation results p_i, so for each model m_i, we have $p_i = E(m_i, d')$. The similarity between r_i and p_i can be measured by the dice coefficient. In the context of robustness, we aim to measure the sensitivity of an explanation method to small changes in input features, i.e., the consistency between the original explanation results r_i and the explanation results after perturbations p_i. This part can be formulated as equation 5.7:

$$R(r_i, p_i) = \frac{2 \times |r_i \cap p_i|}{|r_i| + |p_i|} \tag{5.7}$$

For the entire model set M, we can get the overall robustness evaluation by averaging the robustness values of each model m_i:

$$rob(M, d) = \frac{1}{n} \sum_{i=1}^{n} R(r_i, p_i) \tag{5.8}$$

High robustness value means that the interpretation method remains consistent under slightly perturbed inputs. When the robustness value is close to 1, it means that the interpretation results before and after the perturbation are almost the same, and the interpretation method is insensitive to small changes in the input and has high stability. Low robustness value means that the interpretation method is very sensitive to changes in the input, resulting in unreliable interpretation results.

5.2.3 Effectiveness (eff)

Effectiveness is a measure of how important the explanation results (important features) are to the predictions. If important features are the basis for the model predictions, removing important features should result in significant changes in the classification prediction results [30].

Effectiveness can formally defined by the following:
For model set $M = \{m_1, m_2, m_3, ..., m_n\}$ and a original dataset d, run each model m_i on the original dataset d and generate predictions y_i, then use explainer E to extract the top_k important features for each model m_i, denote as $r_i = E(m_i, d)$. After that, for each model m_i, we remove the important features r_i of the model obtained by the explainer and obtain a new dataset d'. Use model m_i again on the new dataset d' to get a new prediction result y_i'. In the context of effectiveness, we aim to measure the prediction results change when features considered important are removed. This change can be measured by comparing the predictions y_i of the original dataset with the predictions y_i' of the new dataset after removing the important features. For each model m_i, the effectiveness measure e_i can be expressed as:

$$e_i(m_i, d) = \begin{cases} 1, & |y_i - y_i'| \geq \lambda, \\ 0, & |y_i - y_i'| < \lambda. \end{cases} \tag{5.9}$$

where λ is a threshold value. The reason for adding the threshold is to allow a certain degree of error, that is, to ignore the too small fluctuations produced by the model. Only when the error exceeds the threshold, we consider these features to have an important influence on the decision of the model. Otherwise, they have no influence. If the prediction results after deleting important features are different (decreasing) from the original prediction results, it is considered that these features have a positive impact on the prediction basis of the model, so $e_i = 1$, otherwise it is $e_i = 0$. For the model set M, the effectiveness can be measured by averaging the value of each e_i:

$$eff(M, d) = \frac{1}{n} \sum_{i=1}^{n} e_i \tag{5.10}$$

where n denotes the number of models in model set M.
The higher the average effectiveness value, the higher the proportion of models whose predictions change after deleting important features, indicating that the important features identified by the interpretation method play a key role in the model prediction and the stronger the effectiveness. Conversely, the lower the effectiveness value, the less significant the change in the model prediction results after deleting the features, indicating that the results obtained by the interpretation method are not effective.

The higher the effectiveness, the more reliable the results obtained by the interpretation method are if the results obtained by the interpretation method are the basis for the model predictions. Conversely, if the model prediction results do not change significantly after deleting important features, it means that the decision basis of the model is irrelevant to the results obtained by the interpretation method. In this case, it is difficult for the feature-based global interpretation method to trust the interpretation results generated by the interpretation method because it fails to identify features that actually affect the model predictions.

6. Experiments Setup

In this study, we intend to evaluate the selected XAI methods based on 5 selected metrics. Our experiment is divided into two parts: local XAI methods and global XAI methods. The specific process is shown in Figure 6.1.

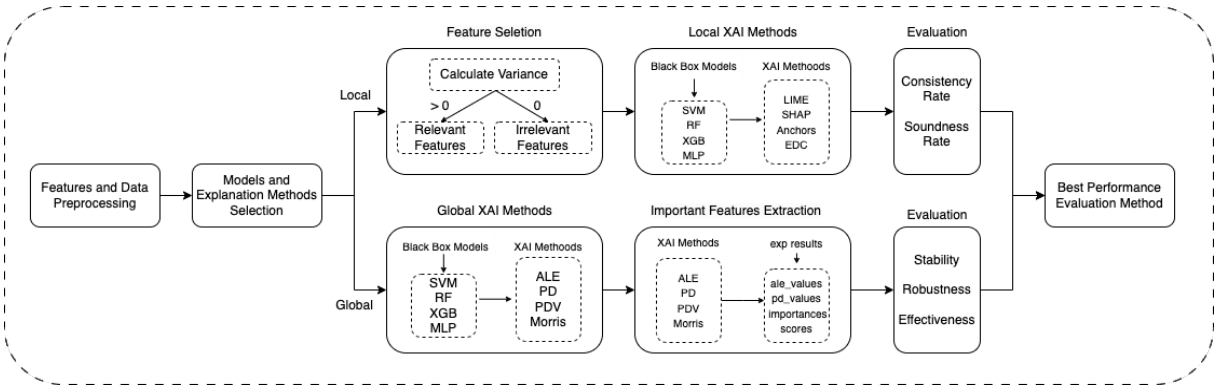


Figure 6.1: Experiment set-up for evaluating local and global XAI Android malware detection methods

We first preprocessed the data to filter out outliers and missing values, and selected XAI methods that generated meaningful results and their supported black-box models. For local explanation methods, we divided relevant and irrelevant features by calculating the variance of the data. Then, black-box malware detection models based on SVM, RF, XGB, and MLP were trained. Subsequently, we used the four selected local XAI methods to explain the results generated by the candidate models. Finally, the performance of XAI methods was evaluated based on consistency rate and soundness rate. For global explanation methods, we trained black-box malware detection models based on SVM, RF, XGB, and MLP. Subsequently, we used the four selected global XAI methods to explain the candidate models. Based on the explanation results, the most important features were selected and the results generated by the candidate models were explained using the four selected global XAI methods. Finally, the performance of global XAI methods was evaluated based on stability, robustness, and effectiveness. The specific details of these steps will be discussed in the rest of this Section.

6.1. Dataset

In 2017, Wei et al. built a large Android malware dataset, called AMD [88]. This dataset collects 24,650 malware samples from 2010 to 2016 and manually analyzed them, which were divided into 71 families and their 135 variants. AMD calibration method for malware family information is based on the return results of VirusTotal, that is, each sample is voted for using an antivirus engine. If the family label given by the result is consistent with more than half, the label is assigned to the corresponding sample. The categories of AMD are

shown in Table 6.1. Since AMD only contains malware datasets, to balance the data set and improve the reliability of the overall analysis, we also need benign data to improve the generalizability of the model. The benign dataset was downloaded from AndroZoo [3], which is a growing collection of Android applications from multiple sources. Currently, the platform provides 25,071,571 different APKs, and each APK has been tested by at least 10 different AntiVirus tools to ensure the reliability of its data. Androzoo contains both malware and benign datasets, aiming to provide and encourage researchers with new potential research topics.

6.1.1 Features and Data Preprocessing

In Android applications, APK is the installation package of the application, which contains files such as the manifest file (AndroidManifest.xml), the dex file (classes.dex) and other resources. The AndroidManifest.xml file in the APK defines various components of the application, such as Activity, Service, Broadcast Receiver and Content Provider, while classes.dex contains the compiled application code [27]. As we mentioned in Section 4, the methods used for static and dynamic analysis of Android applications are different. In order to perform static analysis on Android applications, the APK needs to be unzipped and decompiled to extract static features of the manifest file and dex file. For example, sensitive permissions, activity names, intents and other configuration information can be extracted from the manifest file, while sensitive API calls, control flow and data flow can be extracted from the dex file. In addition, APKs that failed to parse or were incomplete need to be removed to avoid their impact. The feature extraction process is completed by Androguard[1], a tool written in Python that can be used to process Android files. After screening, we obtained a total of 696 static features. In contrast, dynamic analysis of Android applications requires executing applications in a dedicated analysis environment and extracting dynamic features by monitoring runtime behaviors (such as system calls, Binder calls, network traffic, and file operations) [56].

Given that AMD only contains malware samples, we downloaded the same number of benign samples as AMD from AndroZoo at the same time. Since our experimental design covers a variety of detection models and explanation methods, using all the data will significantly increase the computational overhead. Therefore, we only extracted a total of 2,402 benign samples from two sources, anzhi and huawei. To avoid experimental bias and ensure the balance of the two categories in the experiment, we randomly selected 2,400 from each of the benign and malware samples, a total of 4,800 apks for analysis and experiments. We set bit 0 to represent the malware sample label and bit 1 to represent the benign sample label.

To split the training and test sets, we use ShuffleSplit from sklearn[2] to perform random permutation cross-validation on the dataset, where we set $n\_splits = 10$ to create 10 different training and test splits. For each split, 85% of the dataset is divided into the training set and the remaining 15% is divided into the test set to ensure that the benign and malware

[1] https://github.com/androguard/androguard
[2] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.ShuffleSplit.html

Type	Description	# of families
Trojan	It disguises itself to attract users to download, aiming to destroy, steal users important files, information, passwords, etc., or remotely control the computer of the infected user.	12
Trojan-SMS	Attacks targeting mobile text messaging are aimed to intercept or send malicious text messages.	15
Trojan-Banker	Targeting on online banking and digital financial transaction programs, used to steal infected users sensitive information such as identity, banking login credentials, etc., in order to commit fraud in the form of monetary theft, identity theft, etc.	5
Trojan-Clicker	By trying to connect to online resources, mimicking users clicking on designated URLs to increase advertising revenue.	2
Trojan-Spy	Monitor user activities, including capturing and recording keystrokes, websites visited by infected users, emails, tracking logins and other sensitive data.	4
Trojan-Dropper	Designed to drop additional malicious programs on already infected mobile or computers.	6
Backdoor	By bypassing normal authentication procedures, it can remotely access resources within unauthorized applications on the system, or damage or delete hard disk data.	11
Hacker Tool	Launch attacks against local or remote computers and hide attackers presence on the system.	2
Adware	Advertisement-supported software, malicious adware makes illegal profits by popping up ads on mobile devices or computers to trick users into clicking or installing them.	8
Ransom	The user is held ransom by encrypting files to prevent access.	6

Table 6.1: The details of AMD malware dataset categories and descriptions.

two classes are evenly distributed in all splits. ShuffleSplit randomly shuffles the data and each split is independent and non-repeated, which can better reflect the actual performance of the model.

6.1.2 Feature Selection

Based on evaluation metrics for local explanation methods 5.1, we first need to divide the features into relevant features and irrelevant features. For the purpose of distinguish relevant and irrelevant features in the dataset, we calculate the variance of the features and sort them. The features with the variance of 0 are considered as irrelevant features and the rest are relevant features. The larger the variance of a feature, the more important the feature is. Conversely, the smaller the variance of a feature, the less important the feature is. Features with a variance of 0 are the least important features and are therefore considered irrelevant features. Among all 696 features, some features have variance values very close to 0 but not 0, and the amount of information contained in these features are very small. In order to avoid noise interference from these features with very small variances, we select the top 200 features with the largest variance as relevant features, and the 200 features with a variance of 0 as irrelevant features. However, for the global explanation methods, this feature extraction processing is not necessary. The extraction of important features in the global interpretation method is based on the values obtained by the explanation methods, which will be discussed in detail in Section 6.3.2.

6.2. Models and Explanation Methods Selection

In Section 3 we have discussed several models and explanation methods that can be applied for Android malware detection tasks. The selection of explanation methods was driven by two key factors. The first was whether the explanation method could generate meaningful explanation results for our dataset and the selected classification models. That is, the explanation method needs to produce low-complexity explanation results that are easy to understand and analyze. Meaningless explanation results cannot provide actionable insights to analysts, thus undermining the trust and understanding of XAI technology. Second, we also considered the computational efficiency of the explanation method. XAI techniques usually consume additional computing resources to generate decision explanations for black-box models, but in actual applications, analysts cannot spend a lot of time waiting for the generation of explanation results. Therefore, such explanation methods are considered inefficient [13], [9].

After our tests, based on the explanation methods of generating counterfactuals, CEM, CF and ProtoCF fail to generate explanation results. Among them, the CEM method failed to generate "PP" and "PN" instances, and CF and ProtoCF failed to find valid counterfactual instances. This shows that these methods have certain limitations in the selection of models or under the current data distribution, and cannot provide feasible or explainable counterfactual results. The local method LORE needs to generate a large number of synthetic neighborhood samples around the original sample through a genetic algorithm, which is computationally intensive and time-consuming; the global interpretation method PI needs

to shuffle each feature value one by one to observe the changes in the model output; the global SHAP method needs to summarize the results of multiple local interpretations for global analysis, so these methods require a lot of computing resources. Taking the SVM model as an example, when running a complete explanation task, methods such as ALE and PD only take a few minutes to generate explanation results, while PI methods take at least 2 hours. The running time of each explanation method on the SVM model is shown in Table 6.2:

Method	Time(s)	Generate Meaningful Result	Description
ALE	149.15	Yes	Global
PD	292.21	Yes	Global
PDV	270.15	Yes	Global
Morris	132.47	Yes	Global
PI	8938.30	Yes	Global
SHAP Global	18730.79	Yes	Global
LIME	1.11	Yes	Local
SHAP	11.73	Yes	Local
Anchors	1.75	Yes	Local
EDC	0.59	Yes	Local
LORE	573.08	Yes	Local
CF	NA	No cf instances	Local
ProtoCF	NA	No cf instances	Local
CEM	NA	No 'PP'/'PN'	Local

Table 6.2: The running time of each explanation method on the SVM model and whether it can generate interpretable results.

Since our experiments focus on the interpretability of black-box models and use tabular data, the choice of interpreter, whether it is a local or global interpretation method, needs to be able to effectively capture and explain the underlying decision-making process in the black-box model and needs to be compatible with the tabular data format. There are already some relatively complete Python libraries for interpreting machine learning models, such as Alibi [49], interpretationML [45], [42] and OmniXAI [91] etc. However, most of the explanation methods we mentioned in these libraries do not currently support models built with TF/Keras. Although the explanation method based on generating counterfactual instances can support the model built by TF/Keras, it cannot generate interpretable results. Therefore, our study was limited to using classical machine learning models, SVM, RF, XGBoost and MLP. Since linear models are usually easy to interpret [56] and our research goal is mainly to target models with low interpretability, linear SVM is not in our selection.

Finally, our experiment chose to use LIME, Anchors, SHAP and EDC as local methods, and ALE, PD, PDV and Morris method as global explanation methods to evaluate four black box models: SVM, RF, XGBoost and MLP.

6.3. Evaluations

The purpose of our experiments is to evaluate the trustworthiness of the results generated by various explanation methods using the metrics we selected. The experiments are divided into two parts: local explanation method evaluation and global explanation method evaluation. Although the two explanation methods use different explainers, the underlying prediction model remains consistent in both parts. As described in the 3.1 section, the criterion for selecting classification models is high accuracy but poor interpretability, because models with high interpretability usually have more transparent and easy to understand decision-making processes, which may not fully challenge the ability of explanation methods in complex tasks. As shown in the figure 3.1, four different classification models are selected for our experiments: SVM, RF, XGBOOST and MLP.

For the performance evaluation of these black-box models, we use accuracy, precision, recall and F1 score to evaluate the performance of the models, which are also the most commonly used detection metrics in current machine learning-based Android malware detection methods. For this, we utilize the confusion matrix to provide insights into how the model performs in the classification task. This matrix describes the number of correct or incorrect classifications made by the model in the classification task. The four key metrics in the confusion matrix are: True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN). The confusion matrix for our classification task is presented in Table 6.3:

Truth	Prediction	
	Positive	Negative
Positive	TP	FN
Negative	FP	TN

Table 6.3: The description of a confusion matrix.

In our experiments, the models prediction results are based on whether samples labeled as malware can be correctly identified, so in the confusion matrix, we consider malware samples as positive and benign samples as negative. Thus, the four key metrics can be defined as:

- TP: labeled as malware, predict as malware;
- FP: labeled as benign, predict as malware;
- FN: labeled as malware, predict as benign;
- TN: labeled as benign, predict as benign.

Then the detection of model performance evaluation metrics can be defined as:

- Accuracy: $\frac{TP+TN}{TP+FP+FN+TN}$;
- Precision: $\frac{TP}{TP+FP}$;
- Recall: $\frac{TP}{TP+FN}$;
- F1 score: $\frac{2\times Precision\times Recall}{Precision+Recall}$.

Table 6.4 lists the parameter settings of the four classification models. These parameter configurations are the parameter combinations with the best comprehensive performance based on the above four performance evaluation metrics.

Model	Parameter
SVM	C=100, gamma=scale, kernel=rbf, probability=True
Random Forest	max_depth=None, n_estimators=200, n_jobs=-1, min_samples_split=5, min_samples_leaf=1
XGBoost	learning_rate=0.1, max_depth=3, n_estimators=300, subsample=1.0, n_jobs=-1
MLP	alpha=0.01, max_iter=200, activation=tanh, hidden_layer_sizes=(100,)

Table 6.4: The parameters setting of 4 classification models.

6.3.1 Local Explanation Methods

In this experiment, we selected 4 explanation methods, LIME, Anchors, SHAP and EDC methods for evaluation. The aim of this experiment is to evaluate the trustworthiness and effectiveness of these explanation methods in explaining the decision-making process of black box models. To evaluate these local explanation methods, we use two metrics, consistency rate and soundness rate as mentioned in Section 5, to verify the results obtained by the local explainers.

Consistency Rate (CR)

As discussed in Section 5.1.1, CR is used to measure the degree of changing in the explained features obtained when using different classification algorithms on the same feature set. For a given single instance, we want to know whether the results obtained by the explanation method are consistent under different classification models.

To achieve this goal, we need to run different classification models on the same interpreter. In order to improve classifier performance and reduce the number of features to reduce feature complexity, we only use relevant features for this experiments. First, we train different classification models in sequence and use a selected local explanation method to explain the same instance and generate a list of important features. The explanation features generated by each model are collected into a list, allowing us to evaluate the consistency of these features across different models. When the explanations of all models are obtained, the consistency rate of the explanation results obtained by the explanation method across different models can be calculated according to the CR formula. Repeat the above steps until all explainers obtain CR results to ensure a more comprehensive evaluation of different explanation methods. However, in fact, the explanation results are unstable, which means that even when the same model and explainer are used for multiple runs on the same instance, the explanation results obtained each time may not be exactly the same. This is because the distribution of instances generated by local explanation methods is not fixed. To

solve this problem, we run multiple times with 20 different random seeds, repeat the explanation process 20 times and take the average results, and use the average results of all runs as the final CR result. This repeated evaluation helps to mitigate the impact of randomness.

A higher CR value indicates that the explanation results produced by the explanation method for different models have a higher similarity and can generate more consistent explanation results. This means that the explanation method is relatively independent of a certain model and has good general applicability. Specifically, a higher CR value means that the explanation method can effectively generate data features that are important to different models. A lower consistency rate means that the explanation results produced by the explanation method for different models vary greatly, and the explanation method may depend on a certain model. The change in the explanation results also means that the explanation method may be more suitable for certain specific models and depends to a certain extent on the decision-making method of the specific model.

Soundness Rate (SR)

As mentioned in Section 5.1.2, SR is used to measure the degree of change in the explanatory features obtained after adding an irrelevant dataset to the original dataset. Therefore, in this experiment, we need to add the irrelevant features filtered out in the feature extraction step 6.1.1 into the original dataset. The purpose of this experiment is to analyze the impact of adding irrelevant features on the important features generated by the explanation methods. By comparing the explanation features generated before and after adding irrelevant features, we can evaluate the soundness of the model explanation under the influence of irrelevant data.

To measure the change in the explanation results after adding irrelevant features, we can compare them with the explanation results of the original features. Specifically, when irrelevant features are added, whether the explanation feature set is consistent with the explanation feature set obtained using the original features. In addition, we also need to consider whether the explanation feature set contains irrelevant features. The original feature set is trained with different classification models in turn and generates an explanation feature set with the specified interpreter. The explanation features generated by each model are collected into a list. In the second run, we add irrelevant features to the original features to generate a new feature set. We repeat the steps of the first run with the new feature set to obtain a new explanation feature set after adding irrelevant features and store it in a new list. After obtaining these two results, we can derive the soundness rate of the explanation method according to the SR formula. Then we run multiple times with 20 different random seeds, repeat the explanation process 20 times and take the average results, and use the average results of all runs as the final SR result.

A higher SR value indicates that the explanation method is not affected too much after adding irrelevant features, and the interpretation results are relatively stable and reliable. In other words, a higher SR indicates that the explanation method can ignore the noise caused by irrelevant features and focus more on the truly important features. On the contrary, a lower SR value means that the interpretation method is very sensitive to irrelevant

features and is easily affected by noise. Therefore, the conclusions drawn by this method have lower reliability. When irrelevant features appear in the interpretation feature set, it means that the interpretation method mistakenly identifies irrelevant features as relevant features, thereby drawing wrong conclusions, which means that the results obtained by the explanation method are unreliable and people will be confused by the results. Because ideally, the explanation method should be noise resistant and be able to successfully filter out irrelevant features, thereby providing more reliable interpretation results.

6.3.2 Global Explanation Methods

In the experiments of global explanation methods, we focus on analyzing the trustworthiness of the results generated by global explanation methods. We selected 4 global explanation methods, ALE, PD, PD Variance and Morris method we have discussed in Section 3.2, and evaluated them according to the three global explanation feature-based metrics, stability, robustness and effectiveness we formulated in Section 5.2.

Important Features Extraction

The important feature extraction method in the global explanation method is different from that in the local explanation method. The important features of the global explanation method are based on the generated results of the explanation method. The explanation results generated by the feature-based explanation methods mainly include feature names and their corresponding values. For example, the ALE method contains ale values, PD contains pd values, PD Variance contains feature importances, and the Morris method contains scores, which is also the basis for our ranking of important features. As described in Section 3.2.2 and Section 3.2.2, the pd values in the PD method are the marginal effects of the specific values of each feature on the prediction, and the ale values are the locally accumulated average difference of the features to the model predictions within the interval. Therefore, we can rank the feature importance by calculating the range of variation of the pd values and ale values of each feature to find out the important features that affect the model prediction. Taking the feature 'android.permission.CAMERA' as an example, the feature value is 0 and 1, so the pd value is: $[[0.49814314, 0.48507062], [0.50185686, 0.51492938]]$, the first array is the pd value when the feature is 0, and the second array is the pd value when the feature is 1. The first element in each array corresponds to the malware label we set, and the second element corresponds to the benign label. This shows that when the feature value is 0, the model predicts that the probability of malware is about 0.49814314, and the model predicts that the probability of benign is about 0.48507062. When the feature value is 1, the model predicts that the probability of malware is about 0.50185686, and the model predicts that the probability of benign is about 0.51492938. Thus, by calculating the difference between the maximum and minimum values in the array, that is, 0.51492938 - 0.48507062 $=$ 0.02985876, we can get the influence range of the feature on the prediction output under different values. The larger the range of variation, the more important the feature is. For the ale values of this feature, we can also get a array: $[[0.00653626, -0.00653626], [-0.00653626, 0.00653626]]$, the positive and negative values are due to the centralization of the ale method. This means that when the feature value is 0, the model predicts effect of malware class increases by 0.00653626 and the effect predicts as

benign decreases by 0.00653626. When the feature value is 1, the result is the opposite. We can also find important features by calculating the difference between the absolute values of the maximum and minimum values. This ale values shows that the average effect of this feature is zero, so the impact is weak. PD Variance method 3.2.2 includes feature importance values: $[[0.00924367, 0.00476579, 0.00029151, ...], [0.00924367, 0.00476579, 0.00029151, ...]]$, where the first array represents the importance score of each feature when the feature value is 0, and the second array represents the importance score of each feature when the feature value is 1. We can get the important features by calculating the importance difference between each feature value. In Morris method 3.2.2, scores measure the impact of the input variable on the model output when the feature value changes. For example, the scores generated by Morris method are $[0.012768212932900513, 0.0033002767857142844, 0.0036563035714285595, ...]$, each score represents the average sensitivity of an input feature. The higher the score, the more sensitive the feature is, that is, the greater the impact on model prediction.

Through the above method, we can sort the calculated importance of each feature and find the key features that have a greater impact on the prediction results. In the sorting process, we first evaluate the range of change of each feature on the predicted output under different values. Features with a larger range of change indicate that the feature can cause significant changes in the model under different values, otherwise it indicates that the feature has a weak influence. In addition, considering that our experiment is based on important features, and there are a large number of features with weak influence in the actual feature set, we need to choose an appropriate number of important features to avoid the interference of these features on subsequent interpretation or decision-making. Referring to the experience of Fan et al. [30] and Soulimani [78] in selecting the number of important features in local interpretation methods, Fan et al. selected a maximum of 20 features in the experiment, while Soulimani selected a maximum of 30 features. Given that the number of features we use is between the two, we finally decided to select the top 30 features with the highest importance after sorting as the most important features. This number can filter out features with weak or even zero influence, and can also ensure that key information is fully captured.

Stability (stb)

In the Section 5.2.1, we have discussed the definition of stability, the degree to which the explanations generated by the explanation method vary between runs. We believe that explanation methods should be stable, that is, the results they produce should remain consistent across multiple runs.

To verify this, we measure the stability of the explanation method by comparing the sets of important features generated across multiple runs with different random seeds. Specifically, we apply different random seeds to generate important feature sets in multiple experiments and compare the similarity between sets. If the important feature sets generated in multiple runs are highly consistent, it means that the method is stable. We first run the selected explanation method multiple times, and the explanation feature set generated in each run contains the top 30 important features, and the results are stored in a list. These sets are then compared in pairs to calculate the similarity between the sets. In our experiment, the number of multiple runs is 20, so the number of feature set pairs is $\frac{20\times(20-1)}{2} = 190$. Finally,

we average the similarity values obtained for these set pairs to obtain the final stability of the explanation method. Repeat the above steps until all explanation methods have obtained similarity values.

An explanation method with high stability means that after multiple runs, the interpretation feature sets maintain a high consistency rate, that is, the results of each run do not change significantly, indicating that the explanation method has high stability. On the contrary, low stability means that the interpretation feature sets generated between different runs are very different, indicating that the explanation method has poor stability and the generated interpretation results are not reliable.

Robustness (rob)

The robustness of the explanation method means that when a slight perturbation is applied to the input, the explanation results generated by the explanation method are basically consistent with the results generated using the original data, without significant changes. This shows that the method can identify the truly important features in the data without being overly sensitive to small input changes and generating misleading explanations.

To verify the robustness, for a given global explanation method, we first apply each classification model to the original data and obtain the corresponding explanation results. In our experiments, these explanation results are the top 30 important features identified by each model through the explanation method. After obtaining the important features, we randomly select 30 non-important features from the remaining features after excluding the important features for perturbation. The reason for selecting non-important features for experiments is to avoid destroying key information, otherwise it will be difficult to distinguish whether the change in the results is due to the lack of robustness of the explanation method itself or the destruction of key information. We use eigenvalue permutation as the perturbation method, that is, for the selected non-important features, we replace their value 0 with 1 with a certain probability, and for binary values, flip their value 1 to 0 with a certain probability. For integer values greater than 1, we randomly add or subtract 1 with a certain probability. We use this method to perturb all important features in turn and obtain a set of new features based on perturbation. After that, we apply the new feature set based on perturbation to generate a new perturbed explanation feature set, and compare this feature set with the original explanation feature set. Based on this result, we can deduce the robustness of the explanation method according to robustness formula discussed in Section 5.2.2. We run the experiment 20 times with different random seeds to reduce the impact of random variations. We store the robustness values obtained for each run in a list, and after all runs are completed, we average the values in the list to obtain the final robustness value of the explanation method. Repeat the above steps until the robustness of all global explanation methods is obtained.

We hope that the explanation method to be robust to small perturbations. Small perturbations mean that the input data has changed slightly, while the underlying important features remain largely unchanged. For small perturbations, the higher the robustness value, the less sensitive the explanation method is to small changes in the input, the noise can

be ignored, and it has a certain degree of noise resistance, indicating that the explanation method can stably identify important features in the data and will not produce misleading explanation results due to noise. Low robustness means that the explanation method is extremely sensitive to small noise, and small changes in the input will cause significant changes in important features, so the explanation results produced by the explanation method are unreliable.

Effectiveness (eff)

The effectiveness of an explanation method means that the results generated by the method are valid and trustworthy. This shows that the important features generated by the explanation method are an important basis for the models decision-making process, and when the important features change, the models prediction results will also change significantly. Effectiveness represents that the explanation method can accurately capture the decision basis of the model, so that users can trust its explanation results.

The purpose of this experiment is to evaluate whether the explanation results generated by the global explanation method are the decision basis of the model by comparing the changes in accuracy before and after the classification model deletes important features. To verify the effectiveness, we first select a classification model, train it on the original dataset, and record the original accuracy of the model. After obtaining the model prediction results, the specified explanation method is used to generate the explanation results of the model, that is, the top k most important features, which are regarded as important bases for the model to make predictions. These features account for about k of the total number of features. Among them, the selection of k is based on the percentage of the total number of features (for example, 10%, 20%, etc.).
The selection of k is to balance the importance and information deletion. Deleting too many features may cause too much information loss and affect model prediction, and it is impossible to determine whether the change in results is caused by the deletion of important features; deleting too few features may cause the model to be insensitive to a small amount of information loss due to the robustness issues of some classification models. At the same time, based on the number of k, we can also determine the approximate number of important features required for model decision-making.
After that, we delete the top k important features determined by the explanation method to form a new feature set. The new feature set no longer contains these top k features, and the remaining features are the same as the original features. We use the new feature set to predict the selected classification model again and compare the results with the original results. f the model metrics: : accuracy, precision, recall and F1 score after deleting the features are less than the original metrics, the deleted features are considered to be more important for model decision-making, and the validity is recorded as 1; if the experimental metrics do not change, the validity is recorded as 0.
However, even if the same data is used for repeated runs, the accuracy obtained each time cannot be exactly the same. This is because the training samples, initialization weights and other factors of the model during the training process are random, resulting in different initial states during model training and different results. Secondly, the accuracy change in the two experiments is too small to determine whether it is caused by the random fluctuation

of the model or the deletion of important features.
To solve this problem, we set a threshold, that is, when the model accuracy changes by more than 5%, we consider the change to prove effectiveness. Otherwise, the change can be ignored. The setting of the threshold can help us more accurately judge the actual impact of deleting important features on model decisions and avoid mistaking slight random fluctuations of the model for changes in feature importance. After obtaining the results, we can derive the effectiveness of the explanation method based on the effectiveness formula, that is, whether the explanation feature set is an important basis for model decision-making. At the same time, in order to avoid fluctuations in model prediction results, we use 20 different random seeds for multiple runs, and store the two results obtained from each run in two different lists, and then average the effectiveness obtained from each run to obtain the effectiveness of the explanation method for the classification model.
Repeat the above steps until all classification models are verified, and then we average the effectiveness of all models to obtain the final effectiveness of the explanation method. Then change the explanation method and repeat the above steps until the effectiveness of all explanation methods is obtained.

One potential problem is that if enough features are deleted, even if they include unimportant features, the performance of the model will deteriorate. This is because deleting a large number of features may reduce the amount of information available for the model to learn, or change the data distribution. To avoid interference from this situation, we added a verification step based on the original experiment, that is, deleting irrelevant features and retaining only relevant features for model prediction. If the important features in the explanatory feature set are the basis for the model decisions, then even if the model only uses the important features for prediction, the prediction results should be similar to those obtained using the original feature set.

A higher effectiveness value indicates that the interpretation results generated by the explanation method are the basis for model decision-making, so the results obtained by the explanation method can be considered valid. A lower validity indicates that the model does not rely on the important features generated by the explanation method, and the explanation method fails to effectively capture the real basis for model decision-making, resulting in a lower credibility of the interpretation results.

6.4. Hardware

All experiments were run on a Vibranium server at the Leiden Institute of Advanced Computer Science (LIACS) Data Science Lab. Vibranium has the following specifications:

CPU	48 Intel(R) Xeon(R) Silver 4214R CPU @ 2.40GHz
Storage	3TB
RAM	256GB
OS	CentOS Linux release 7.9.2009 (Core)
Architecture	x86_64
CPU op-mode(s)	32-bit, 64-bit
CPU cores	12
GPU	NVIDIA GeForce RTX 3090

6.5. Software

This section summarizes the implementation (libraries used) of each XAI method and the use of four machine learning methods. Table 6.5 summarizes the libraries used by each model, and Table 6.6 summarizes the libraries used by the explanation methods used in the experiments.

Model	Implementation
SVM	sklearn.svm
RF	sklearn.ensemble
XGBOOST	xgboost
MLP	sklearn.neural_network

Table 6.5: Packages used by four machine learning models.

Explainer	Implementation
LIME	lime
SHAP	shap (KernelExplainer)
Anchors	alibi
EDC	Github
ALE	alibi
PD	alibi
PD Variance	alibi
Morris	InterpretML

Table 6.6: Libraries used by each explanation methods. EDC does not yet provide a library package that can be installed directly, so we need to download the source code from Github manually[3] [4].

[3] https://github.com/oussama-soulimani/Evaluating-the-explanation-of-android-malware-detection/blob/master/fn_sedc.py

[4] https://github.com/oussama-soulimani/Evaluating-the-explanation-of-android-malware-detection/blob/master/sedc_algorithm.py

7. Experiment Results and Discussion

This section presents our experimental results, including the prediction results of the classification models and the results of the explanation methods. The results of the explanation methods are divided into local explanation results and global explanation results.

7.1. Local Explanation Results

In the local explanation experiment, we applied two metrics consistency rate (CR) and soundness rate (SR) to evaluate the performance of four local explanation methods LIME, SHAP, EDC and Anchors on four classification models SVM, RF, XGBoost and MLP. In this section, we discuss the results obtained based on the two metrics respectively.

Table 7.1 lists the parameters and descriptions used of each local explanation method in the experiment.

7.1.1 Consistency Rate (CR)

In the experiment of CR, we want to check whether the set of explanatory features generated by the explanation method is consistent across different classification models. Besides, the experiment also tested different parameter configurations of these explanation methods. By comparing the CR values under different parameters, we can intuitively find the impact of parameters on the consistency rate of explanation methods.

LIME CR Results

The CR results of the LIME method are shown in Table 7.2. According to the results in the table, the parameters with the highest consistency rate obtained by the LIME method are "distance_metric: manhattan, num_samples: 7000", and the result is 69.94%. The parameters with the lowest consistency rate are "distance_metric: euclidean, num_samples: 9000", and the result is 57.17%.

Figure 7.1 shows the changing trend of LIME under different parameters. From Figure 7.1 (left), we can see that the CR rises sharply between the 10th and 11th indexes. Combined with Table 7.2, we can find that this is due to the change of the distance parameter. From Figure 7.1 (right), it can be clearly find that the CR value of LIME changes significantly under different distance parameters and the number of neighborhood samples. When the Manhattan distance parameter is used, the highest CR result is 69.94%, and the worst CR is 67.12%. When the Euclidean distance is used, the highest CR of LIME is only

Parameter	Description	Method
distance_metric	The distance between the perturbed data point and the original data point. The closer the instance is to the explanation, the greater the weight assigned.	LIME
num_samples	The number of new perturbed samples generated after perturbing the sample point.	LIME
l1_reg	The selection of l1 regularization, auto indicates automatic selection, aic uses Akaike and bic uses Bayesian information criteria for regularization parameters selection	SHAP
nsamples	The number of subsets used to estimate the shap value.	SHAP
max_iter	The maximum number of iterations in the search procedure.	EDC
threshold_classifier	Threshold, when the score exceeds the threshold, the instance is predicted as positive, otherwise it is negative. Probs is the predict proba method of the model, which is used to predict the training data and return the probability that each sample belongs to each category.	EDC
batch_size	The batch size of sampling	Anchors
tau	the multi-armed bandit parameters for selecting candidate anchors at each iteration, i.e. finding the most promising anchor within a tolerance according to precision.	Anchors
threshold	The minimum anchor precision threshold, which is an anchor that maximizes the coverage under precision constraint.	Anchors

Table 7.1: The parameters of each local explanation methods.

61.02%, and the lowest CR is 57.17%. Therefore, when the Manhattan distance parameter is used, the CR value is much higher than the Euclidean distance. This shows that for our dataset, using Manhattan distance can obtain a higher consistency rate. In other words, the important features identified by LIME using Manhattan distance have a greater overlap between different classification models. This means that LIME can provide a more reliable explanation under this parameter.
Under both distance parameters, the number of neighborhood samples reaches a peak at 7000. When the number of samples exceeds 7000, the CR value begins to decrease, indicating that too many samples may introduce noise and make the local linear approximation effect worse, resulting in large differences in the important features identified between models. When the distance parameter is Euclidean distance and the number of neighborhood samples is 9000, LIME has the worst consistency rate of only 57.17%. This shows that the explanation feature results generated by different classification models are very different.

In summary, this experiment shows that for the dataset and classification model we selected, when Manhattan distance is used and the number of neighborhood samples is set to 7000, LIME can obtain a high and stable consistency rate. The high consistency rate indicates that the important features identified by the four models have a high degree of similarity. This result provides a valuable reference for further optimizing the parameter selection of LIME.

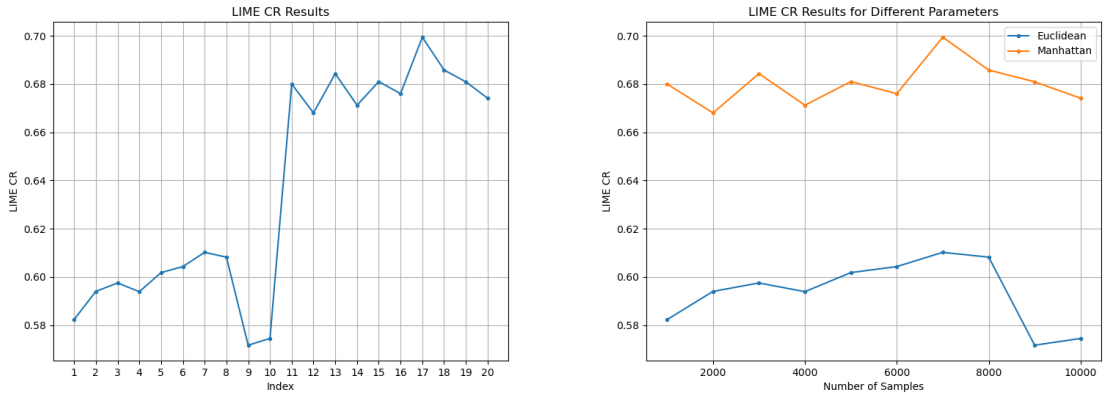


Figure 7.1: CR results of LIME under different parameter combinations. (Left) The horizontal axis shows the index corresponding to each parameter combination listed in Table 7.2, while the vertical axis displays the corresponding CR value. (Right) CR results plotted for each parameter combination, highlighting the effect of different settings on the CR results.

SHAP CR Results

The CR results of the SHAP method are shown in Table 7.3. According to the results in the table, we can find that the highest consistency rate result obtained by the SHAP method is 68.41%, and the lowest consistency rate result is 50.79%.

Index	Parameters	LIME CR Results
1	distance_metric: euclidean, num_samples: 1000	0.5823
2	distance_metric: euclidean, num_samples: 2000	0.5940
3	distance_metric: euclidean, num_samples: 3000	0.5975
4	distance_metric: euclidean, num_samples: 4000	0.5939
5	distance_metric: euclidean, num_samples: 5000	0.6018
6	distance_metric: euclidean, num_samples: 6000	0.6043
7	distance_metric: euclidean, num_samples: 7000	0.6102
8	distance_metric: euclidean, num_samples: 8000	0.6082
9	distance_metric: euclidean, num_samples: 9000	0.5717
10	distance_metric: euclidean, num_samples: 10000	0.5745
11	distance_metric: manhattan, num_samples: 1000	0.6800
12	distance_metric: manhattan, num_samples: 2000	0.6680
13	distance_metric: manhattan, num_samples: 3000	0.6843
14	distance_metric: manhattan, num_samples: 4000	0.6712
15	distance_metric: manhattan, num_samples: 5000	0.6810
16	distance_metric: manhattan, num_samples: 6000	0.6760
17	distance_metric: manhattan, num_samples: 7000	0.6994
18	distance_metric: manhattan, num_samples: 8000	0.6858
19	distance_metric: manhattan, num_samples: 9000	0.6809
20	distance_metric: manhattan, num_samples: 10000	0.6741

Table 7.2: CR results of the LIME method under different parameter combinations. The parameter distance_metric measures the distance between the perturbed data point and the original data point. The closer the instance is to the explanation, the greater the weight assigned. Parameter num_samples is the number of new perturbed samples generated after perturbing the sample point.

Figure 7.2 shows the changing trend of SHAP under different parameters. As can be seen from the figure (left), the nsamples parameter is the main parameter affecting the CR value of the SHAP method. Figure (right) further reveals the impact of nsamples on the CR value. Specifically, when nsamples is 2000, the CR of the three l1_regs is the lowest. As nsamples increases, the CR value generally shows an upward trend. When the parameter value reaches 30000, the CR under the three l1_regs reaches a peak and still shows an upward trend. This shows that increasing the number of samples can significantly improve the consistency of the explanation results between different models. Compared with the nsamples parameter, the l1_reg parameter has a relatively small effect on SHAP, and the curve does not change much under three different l1_regs. When l1_reg is auto and nsamples is 2000, SHAP has the lowest CR value, which is only 50.79%. When l1_reg is auto and nsamples is 30000, SHAP has the highest CR value, which is 68.41%.
It is worth mentioning that, for this experiment, although a high consistency rate was obtained when nsamples reached 30000, according to the trend of the curve, it has not yet reached the saturation value, which means that further increasing the number of nsamples is likely to achieve a higher consistency rate. However, whether further increasing the number of nsamples can bring significant improvements and whether the additional computational cost brought by the increase in the number of samples is reasonable are both worthy of further verification and discussion.

In summary, the experimental results show that the nsamples parameter plays a key role in improving the consistency rate of the SHAP method. When the number of subsets used to estimate the shap value is small, SHAP cannot fully capture the important features of each classification model, so the interpretation results between the models are quite different. Increasing the number of subsets can effectively improve this defect and thus improve the reliability of the interpretation results. The choice of regularization parameter has a relatively weak effect on the CR value, which indicates that the regularization selection strategy does not play a key decision-making role. In short, the experimental results emphasize that the nsamples parameter plays a key role in improving the consistency rate of the SHAP method and provide a valuable reference for the reasonable setting of the number of samples in the field of Android malware detection.

EDC CR Results

The CR results of the EDC method are shown in Table 7.4. According to the results in the table, the highest consistency rate obtained by the EDC method is 28.33%, and its parameter configuration is max iter=80, threshold classifier=np.percentile(probs, 95). The lowest consistency rate result is 25%, and the parameters are max iter=20, threshold classifier=np.percentile(probs, 25) and max iter=80, threshold classifier=np.percentile(probs, 25).

From Figure 7.3 (left) can be seen that the CR curve rises sharply at indexes 12 to 13. Combined with Table 7.4, this change in amplitude is caused by the increase in the threshold setting to the 95th percentile of the predicted probability. Although the ups and downs of the curve can be seen from the figure, the CR value range is distributed between 0.25 and 0.28, and the overall range of change is small. From Figure 7.3 (right), it can be clearly

Index	Parameters	SHAP CR Results
1	l1_reg: auto, nsamples: auto	0.5549
2	l1_reg: auto, nsamples: 2000	0.5079
3	l1_reg: auto, nsamples: 3000	0.5548
4	l1_reg: auto, nsamples: 5000	0.5857
5	l1_reg: auto, nsamples: 8000	0.6060
6	l1_reg: auto, nsamples: 12000	0.6428
7	l1_reg: auto, nsamples: 20000	0.6725
8	l1_reg: auto, nsamples: 30000	0.6841
9	l1_reg: aic, nsamples: auto	0.5350
10	l1_reg: aic, nsamples: 2000	0.5152
11	l1_reg: aic, nsamples: 3000	0.5553
12	l1_reg: aic, nsamples: 5000	0.6183
13	l1_reg: aic, nsamples: 8000	0.6124
14	l1_reg: aic, nsamples: 12000	0.6398
15	l1_reg: aic, nsamples: 20000	0.6583
16	l1_reg: aic, nsamples: 30000	0.6799
17	l1_reg: bic, nsamples: auto	0.5323
18	l1_reg: bic, nsamples: 2000	0.5123
19	l1_reg: bic, nsamples: 3000	0.5467
20	l1_reg: bic, nsamples: 5000	0.5920
21	l1_reg: bic, nsamples: 8000	0.6049
22	l1_reg: bic, nsamples: 12000	0.6552
23	l1_reg: bic, nsamples: 20000	0.6534
24	l1_reg: bic, nsamples: 30000	0.6778

Table 7.3: CR results of the SHAP method under different parameter combinations. The parameter l1_reg measures the selection of l_1 regularization, auto indicates automatic selection, aic uses Akaike and bic uses Bayesian information criteria for regularization parameters selection. The parameter nsamples is the number of subsets used to estimate the shap value.

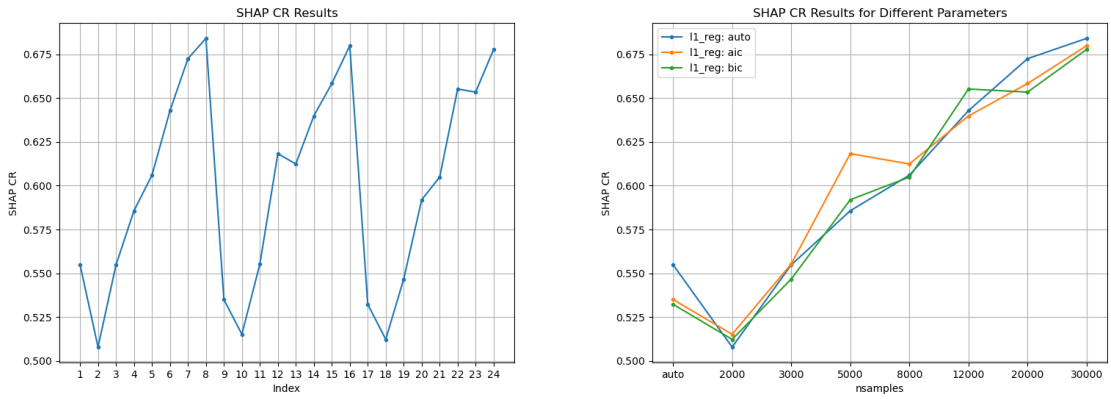


Figure 7.2: CR results of SHAP under different parameter combinations. (Left) The horizontal axis shows the index corresponding to each parameter combination listed in Table 7.3, while the vertical axis displays the corresponding CR value. (Right) CR results plotted for each parameter combination, highlighting the effect of different settings on the CR results.

seen that for the 95% threshold, its CR value is much higher than the other thresholds, and reaches a peak when the maximum number of iterations is 80. It can be clearly seen from the figure 7.3 (right) that for the 95% threshold (i.e., the 95th percentile of the predicted probability), its CR value is much higher than other thresholds and reaches a peak value when the maximum number of iterations is 80. For the 50% threshold, the change range of its CR value curve is relatively consistent with the 95% threshold, but the overall CR value is lower than the 95% threshold and higher than the 25% threshold and 75% threshold. The CR value curve of the 25% threshold shows regular changes, but the overall CR value is low. The CR value curve of the 75% threshold does not form an obvious upward or downward trend under each number of iterations, but the trend is relatively gentle. For the maximum number of iterations, there is no obvious effect on the CR value of EDC. From this result, it can be found that the parameter setting of the threshold lower than 95% does not have much effect on the consistency rate of the EDC interpretation results, but the overall consistency rate result of this method is low, which means that the important features obtained by this method using different classification models have very few overlapping parts.

In summary, the CR value of the EDC method is generally low, which means that using different classification models may obtain almost completely different interpretation results. By experimenting with different parameter configurations, it can be found that this method is relatively less affected by parameter changes, and only when the threshold percentile is 95% can the consistency rate be slightly improved. Some limitations of EDC method will be discussed in detail in Section 8.1.

Anchors CR Results

The CR results of Anchors method are shown in Table 7.5. According to the results in the table, we can find that the highest consistency rate result obtained by Anchors is 33.25%,

Index	Parameters	EDC CR Results
1	max_iter: 20, threshold_classifier: np.percentile(probs, 25)	0.25
2	max_iter: 50, threshold_classifier: np.percentile(probs, 25)	0.2625
3	max_iter: 80, threshold_classifier: np.percentile(probs, 25)	0.25
4	max_iter: 100, threshold_classifier: np.percentile(probs, 25)	0.2625
5	max_iter: 20, threshold_classifier: np.percentile(probs, 50)	0.2625
6	max_iter: 50, threshold_classifier: np.percentile(probs, 50)	0.2625
7	max_iter: 80, threshold_classifier: np.percentile(probs, 50)	0.2688
8	max_iter: 100, threshold_classifier: np.percentile(probs, 50)	0.2625
9	max_iter: 20, threshold_classifier: np.percentile(probs, 75)	0.2625
10	max_iter: 50, threshold_classifier: np.percentile(probs, 75)	0.2583
11	max_iter: 80, threshold_classifier: np.percentile(probs, 75)	0.2604
12	max_iter: 100, threshold_classifier: np.percentile(probs, 75)	0.2583
13	max_iter: 20, threshold_classifier: np.percentile(probs, 95)	0.2792
14	max_iter: 50, threshold_classifier: np.percentile(probs, 95)	0.2792
15	max_iter: 80, threshold_classifier: np.percentile(probs, 95)	0.2833
16	max_iter: 100, threshold_classifier: np.percentile(probs, 95)	0.2792

Table 7.4: CR results of the EDC method under different parameter combinations. The parameter max_iter measures the maximum number of iterations in the search procedure, and parameter threshold_classifier is the threshold, when the score exceeds the threshold, the instance is predicted as positive, otherwise it is negative. Probs is the predict_proba method of the model, which is used to predict the training data and return the probability that each sample belongs to each category.

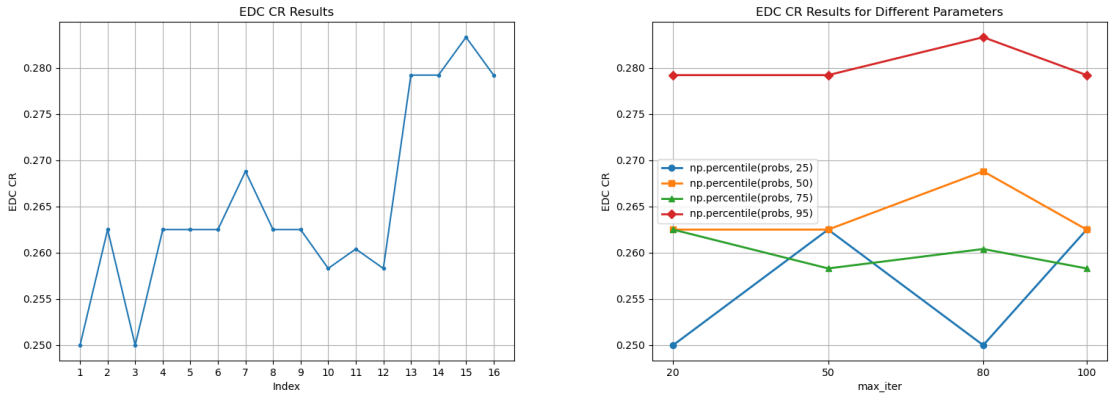


Figure 7.3: CR results of EDC under different parameter combinations. (Left) The horizontal axis shows the index corresponding to each parameter combination listed in Table 7.3, while the vertical axis displays the corresponding CR value. (Right) CR results plotted for each parameter combination, highlighting the effect of different settings on the CR results.

and the lowest consistency rate result is 30.68%.

Figure 7.4 shows the changing trend of Anchors method under different parameter combinations. It can be seen from the figure that the CR value of Anchors does not change much under different parameter combinations, and the fluctuation range is concentrated between 0.307 and 0.333. In this range, the peak value of Anchors is more obvious. As can be seen from the Figure 7.4 (right), under the same threshold, different batch_size and tau combinations have different change trends. When batch_size is 32, tau is 0.15 and batch_size is 64, tau is 0.1, the threshold is 0.9, which is the highest point, 0.95 is the lowest point, and then it shows an upward trend, and the overall change range is more obvious. When batch_size is 64 and tau is 0.15, the curve shows a slow upward trend with the increase of threshold. When batch_size is 32 and tau is 0.1, the curve also shows a trend of first decreasing and then increasing, but the highest CR value is reached when the threshold is 0.99 and the overall change trend is relatively gentle. This observation shows that the CR value of the Anchors method is not affected by a single parameter, but depends on different parameter combinations. However, the results show that the important features selected by the Anchors method in different classification models are significantly different, which indicates that the explanation results generated by this method depend on the selected classification model and lack consistency across models. Therefore, when using Anchors method, it is necessary to consider the generalization ability of the explanation results between different models.

In summary, the best CR value of Anchors is 33.25% when batch size is 64, tau is 0.1, and threshold is 0.9. The worst CR value of 30.68% is obtained when batch size is 32, tau is 0.15, and threshold is 0.95. The CR value of the Anchors method depends on the interaction between different parameter combinations. Selecting appropriate parameters can effectively improve the CR value of Anchors, that is, appropriate parameters can make the interpretation method produce relatively similar interpretation results for different models. Therefore, to obtain a higher consistency rate, it is necessary to jointly tune the parameters and balance the influence of the parameters, so as to optimize the overall consistency performance of the interpretation method. In addition, some limitations of Anchors will be discussed in detail in Section 8.1.

7.1.2 Soundness Rate (SR)

In the experiment of SR, we want to check the impact of adding irrelevant features to the original feature set on the explanation results. In this experiment, we also tested different parameter combinations of the explanation method to explore the impact of parameters on the soundness of the explanation method.

LIME SR Results

Table 7.6 shows the SR results of the LIME method. According to the SR results, when the distance parameter is euclidean and the number of new perturbation samples generated after perturbing the sample points is 10000, LIME has the highest SR value of 96.08%.

Index	Parameters	Anchor CR Results
1	batch_size: 32, tau: 0.1, threshold: 0.9	0.3208
2	batch_size: 32, tau: 0.1, threshold: 0.95	0.3185
3	batch_size: 32, tau: 0.1, threshold: 0.99	0.3248
4	batch_size: 32, tau: 0.15, threshold: 0.9	0.3252
5	batch_size: 32, tau: 0.15, threshold: 0.95	0.3068
6	batch_size: 32, tau: 0.15, threshold: 0.99	0.3097
7	batch_size: 64, tau: 0.1, threshold: 0.9	0.3325
8	batch_size: 64, tau: 0.1, threshold: 0.95	0.3142
9	batch_size: 64, tau: 0.1, threshold: 0.99	0.3231
10	batch_size: 64, tau: 0.15, threshold: 0.9	0.3095
11	batch_size: 64, tau: 0.15, threshold: 0.95	0.3117
12	batch_size: 64, tau: 0.15, threshold: 0.99	0.3187

Table 7.5: CR results of the Anchors method under different parameter combinations. The parameter batch_size measures the batch size of sampling, tau measures the multi-armed bandit parameters for selecting candidate anchors at each iteration, i.e. finding the most promising anchor within a tolerance according to precision. And threshold is the minimum anchor precision threshold, which is an anchor that maximizes the coverage under precision constraint.

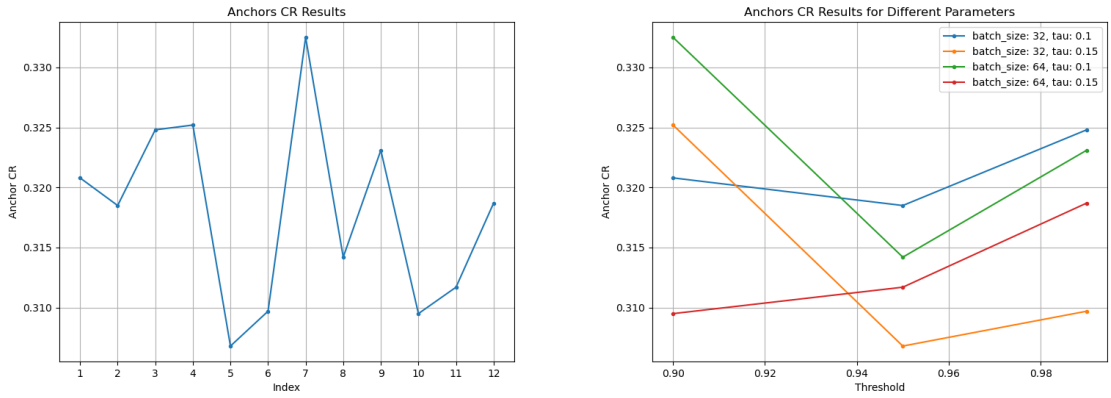


Figure 7.4: CR results of Anchors under different parameter combinations. (Left) The horizontal axis shows the index corresponding to each parameter combination listed in Table 7.5, while the vertical axis displays the corresponding CR value. (Right) CR results plotted for each parameter combination, highlighting the effect of different settings on the CR results.

When the distance metric is manhattan and the number of new perturbation samples generated after perturbing the sample points is 5000, LIME has the lowest SR value of 89.04%.

Figure 7.5 shows the changing trend of LIME SR value under different parameters. As can be seen from the figure (left), the SR value curve drops sharply at indexes 10 to 11. Combined with the results in Table 7.6, it can be seen that this change is caused by the change in the distance metric parameter. This observation shows that the change in the distance parameter alone will have a significant impact on the SR result. The figure (right) better explains the impact of the distance parameter on the SR value. The SR using Euclidean distance parameter is significantly higher than that using Manhattan distance parameter. This shows that after adding irrelevant features, Euclidean distance has stronger anti-interference ability to irrelevant features, so that the interpretation results can be basically consistent. The SR of Manhattan distance is relatively low, which indicates that this distance parameter is relatively sensitive to irrelevant features. However, from the overall results, it can be seen that no matter which distance parameter is used, a relatively high SR is achieved, which shows that the LIME method is relatively insensitive to irrelevant features as a whole and has good anti-noise ability.
Compared with the distance parameter, the number of samples has a relatively weaker effect on the SR results. As can be seen from the figure, as the number of samples increases, when LIME uses the Euclidean distance, the curve fluctuates slightly but shows an overall upward trend. When the Manhattan distance is used, the curve is relatively flat when the number of samples increases, indicating that the sensitivity to irrelevant features does not change much with the number of samples.

In summary, the SR value of LIME is mainly affected by the distance parameter. When using the Euclidean distance, appropriately increasing the number of samples may reduce the sensitivity of the interpretation method to irrelevant features and improve the reliability of the interpretation results. For the Manhattan distance, the number of samples reaches a peak at 4000, which means that further increasing the number of samples has no effect on improving SR. Overall, the LIME method achieves relatively high SR values under both distance parameters, which shows that after adding irrelevant features, the interpretation results of this method can still remain consistent and have stronger anti-interference ability against irrelevant features.

SHAP SR Results

Table 7.7 shows the SR results of the SHAP method. From the SR results, when the l1 regularization parameter is selected as bic and the number of samples is 30,000, the SR value of SHAP is the highest, which is 97.41%; when the l1 regularization parameter is selected as bic and the number of samples is 2,000, the SR value of SHAP is the lowest, which is 89.86%.

Combining Table 7.7 and Figure 7.6 (left), we can see that the peaks and troughs in SHAP SR curve are caused by the change in the number of parameter samples. As the number of samples increases, the SR curves under the three regularization parameters generally show an upward trend. Figure (right) better reveals this phenomenon. When the

Index	Parameter	LIME SR Results
1	distance_metric: euclidean, num_samples: 1000	0.9297
2	distance_metric: euclidean, num_samples: 2000	0.9494
3	distance_metric: euclidean, num_samples: 3000	0.9514
4	distance_metric: euclidean, num_samples: 4000	0.9567
5	distance_metric: euclidean, num_samples: 5000	0.9573
6	distance_metric: euclidean, num_samples: 6000	0.9505
7	distance_metric: euclidean, num_samples: 7000	0.9557
8	distance_metric: euclidean, num_samples: 8000	0.9554
9	distance_metric: euclidean, num_samples: 9000	0.9562
10	distance_metric: euclidean, num_samples: 10000	0.9608
11	distance_metric: manhattan, num_samples: 1000	0.8962
12	distance_metric: manhattan, num_samples: 2000	0.8953
13	distance_metric: manhattan, num_samples: 3000	0.8964
14	distance_metric: manhattan, num_samples: 4000	0.9010
15	distance_metric: manhattan, num_samples: 5000	0.8904
16	distance_metric: manhattan, num_samples: 6000	0.8936
17	distance_metric: manhattan, num_samples: 7000	0.8957
18	distance_metric: manhattan, num_samples: 8000	0.9006
19	distance_metric: manhattan, num_samples: 9000	0.8998
20	distance_metric: manhattan, num_samples: 10000	0.8971

Table 7.6: SR results of the LIME method under different parameter combinations.

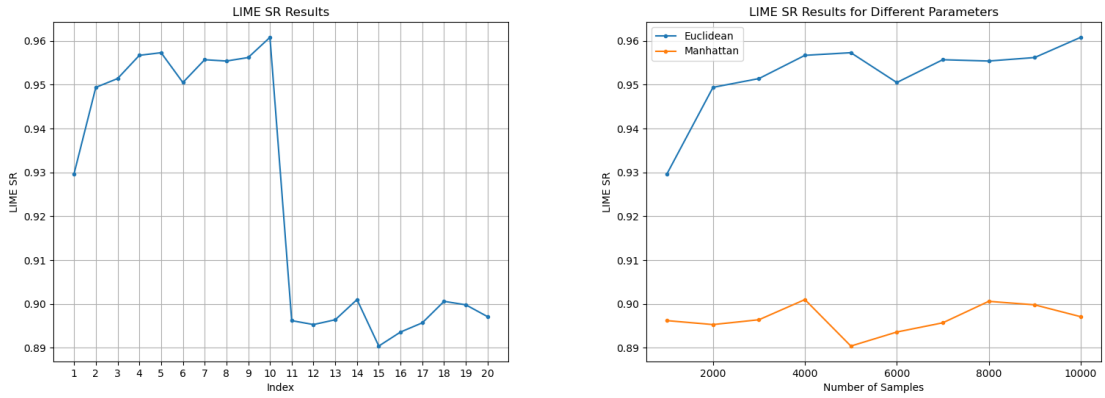


Figure 7.5: SR results of LIME under different parameter combinations. (Left) The horizontal axis shows the index corresponding to each parameter combination listed in Table 7.6, while the vertical axis displays the corresponding SR value. (Right) SR results plotted for each parameter combination, highlighting the effect of different settings on the SR results.

number of samples is 2000, the SR curves under the three regularization parameters are at the lowest point, and gradually rise with the increase of nsamples. Among them, when the regularization parameter is auto, the curve reaches the highest point when the number of samples is 20000, and then decreases. When the regularization parameter is aic, the number of samples decreases slightly in the range of 5000 8000, and then increases. When the regularization parameter is bic, it rises rapidly after the number of samples is greater than 2000, slows down when the number of samples is between 12000 and 20000, and then rises rapidly again. This shows that a larger number of samples can effectively avoid the impact of irrelevant features in the data set and provide a more stable explanation. For the auto regularization parameter selection, the best performance is achieved when the number of samples reaches 20,000, while aic and bic require a larger number of samples to obtain a higher SR.
From the perspective of the overall SR value, the SHAP method can achieve a relatively high SR value even with a smaller number of nsamples, which shows that this method can effectively avoid the interference caused by irrelevant features and provide effective and reliable interpretation results.

In summary, compared with the choice of regularization parameters, the number of samples is more important for the SR of the SHAP method explanation results. In general, the SHAP method can provide reliable explanations when dealing with data sets containing irrelevant features. According to the experimental results, in the field of Android malware detection, using bic as a regularization option and using a larger sample size can effectively improve the SR of the SHAP method and enhance its anti-interference ability.

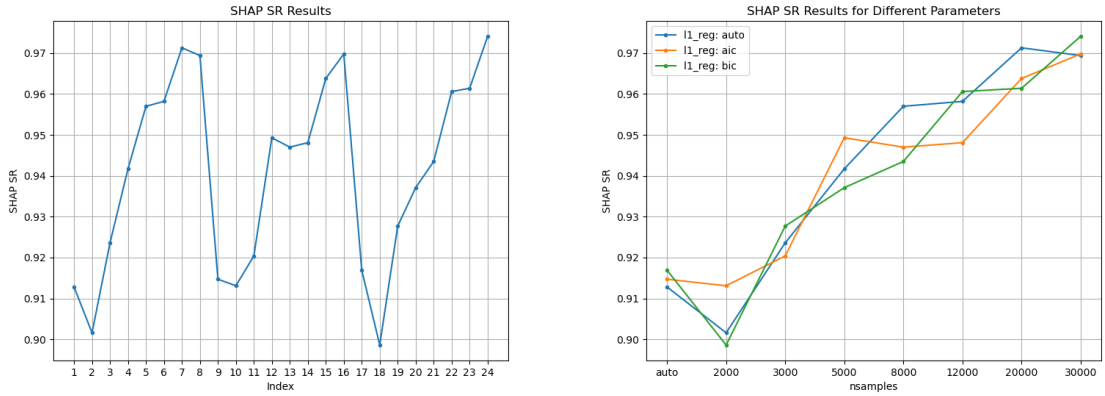


Figure 7.6: SR results of SHAP under different parameter combinations. (Left) The horizontal axis shows the index corresponding to each parameter combination listed in Table 7.7, while the vertical axis displays the corresponding SR value. (Right) SR results plotted for each parameter combination, highlighting the effect of different settings on the SR results.

Index	Parameters	SHAP SR Results
1	l1_reg: auto, nsamples: auto	0.9128
2	l1_reg: auto, nsamples: 2000	0.9016
3	l1_reg: auto, nsamples: 3000	0.9236
4	l1_reg: auto, nsamples: 5000	0.9417
5	l1_reg: auto, nsamples: 8000	0.9570
6	l1_reg: auto, nsamples: 12000	0.9582
7	l1_reg: auto, nsamples: 20000	0.9713
8	l1_reg: auto, nsamples: 30000	0.9694
9	l1_reg: aic, nsamples: auto	0.9147
10	l1_reg: aic, nsamples: 2000	0.9131
11	l1_reg: aic, nsamples: 3000	0.9204
12	l1_reg: aic, nsamples: 5000	0.9493
13	l1_reg: aic, nsamples: 8000	0.9470
14	l1_reg: aic, nsamples: 12000	0.9481
15	l1_reg: aic, nsamples: 20000	0.9638
16	l1_reg: aic, nsamples: 30000	0.9698
17	l1_reg: bic, nsamples: auto	0.9169
18	l1_reg: bic, nsamples: 2000	0.8986
19	l1_reg: bic, nsamples: 3000	0.9277
20	l1_reg: bic, nsamples: 5000	0.9371
21	l1_reg: bic, nsamples: 8000	0.9435
22	l1_reg: bic, nsamples: 12000	0.9606
23	l1_reg: bic, nsamples: 20000	0.9614
24	l1_reg: bic, nsamples: 30000	0.9741

Table 7.7: SR results of the SHAP method under different parameter combinations.

EDC SR Results

Table 7.8 shows the SR results of the EDC method. From the SR results, when the max_iter is 100 and the threshold is 50th percentile , the SR value of EDC is the highest, which is 96.25%; when the max_iter is 50 and the threshold is 95th percentile, the SR value of EDC is the lowest, which is 85.31%.

Figure 7.7 shows the changing trend of EDC SR value under different parameters. As can be seen from the figure, when the threshold is 50 percentile, the SR of EDC is much greater than the other threshold percentiles. When the threshold is at the 25 percentile, the SR curve of EDC decreases significantly in the range of the maximum number of iterations from 20 to 80, and rises rapidly after exceeding 80. When the threshold is at the 75th and 95th percentiles, the curves almost completely overlap, decrease in the range of the maximum number of iterations from 20 to 50, and then increase. When the threshold is at the 50th percentile, the curve also slowly decreases in the range of the maximum number of iterations from 20 to 50, and rises when the number of iterations increases. It can be seen from the observation that the SR of EDC is affected by the maximum number of iterations and the threshold. When the number of iterations exceeds 80, the SR curves under different thresholds show an upward trend. This shows that as the number of iterations increases, EDC can fully resist the interference caused by irrelevant features and provide stable inter-

pretation results. However, as the threshold increases, the SR of EDC gradually decreases, which shows that a high threshold may filter out potential effective information and increase the impact of irrelevant features. However, this defect can be gradually compensated as the number of iterations increases.

In summary, the SR of EDC needs to balance the selection of the maximum number of iterations and the threshold. A lower threshold may not fully tap the effectiveness of features in fewer iterations, resulting in low SR. As the number of iterations increases, the impact of EDC on irrelevant features gradually weakens, and it can provide more stable interpretation results.

Index	Parameters	EDC SR Results
1	max_iter: 20, threshold_classifier: np.percentile(probs, 25)	0.9094
2	max_iter: 50, threshold_classifier: np.percentile(probs, 25)	0.8859
3	max_iter: 80, threshold_classifier: np.percentile(probs, 25)	0.8609
4	max_iter: 100, threshold_classifier: np.percentile(probs, 25)	0.9313
5	max_iter: 20, threshold_classifier: np.percentile(probs, 50)	0.9438
6	max_iter: 50, threshold_classifier: np.percentile(probs, 50)	0.9406
7	max_iter: 80, threshold_classifier: np.percentile(probs, 50)	0.9531
8	max_iter: 100, threshold_classifier: np.percentile(probs, 50)	0.9625
9	max_iter: 20, threshold_classifier: np.percentile(probs, 75)	0.8631
10	max_iter: 50, threshold_classifier: np.percentile(probs, 75)	0.8531
11	max_iter: 80, threshold_classifier: np.percentile(probs, 75)	0.8545
12	max_iter: 100, threshold_classifier: np.percentile(probs, 75)	0.8809
13	max_iter: 20, threshold_classifier: np.percentile(probs, 95)	0.8631
14	max_iter: 50, threshold_classifier: np.percentile(probs, 95)	0.8531
15	max_iter: 80, threshold_classifier: np.percentile(probs, 95)	0.8548
16	max_iter: 100, threshold_classifier: np.percentile(probs, 95)	0.8809

Table 7.8: SR results of the EDC method under different parameter combinations.

Anchors SR Results

The SR results of Anchors method are shown in Table 7.9. According to the results in the table, we can find that the highest consistency rate result obtained by Anchors is 83.64%, and the lowest consistency rate result is 76.79%.

Combining the figure 7.8 and the table 7.9, it can be seen that the SR of Anchors is greatly affected by the threshold. As the threshold increases, the SR shows a downward trend under different combinations of tau and batch size. This shows that in the process of generating interpretation rules by Anchors, although higher rules can improve the confidence of interpretation rules, they will also make the rules more stringent. Therefore, when irrelevant features appear in the data, too high a threshold will limit the range of Anchors in capturing effective features in the data. Even a small disturbance may make the rules that originally meet the threshold no longer valid, resulting in a decrease in the stability of the interpretation results, thereby reducing its overall anti-interference ability.

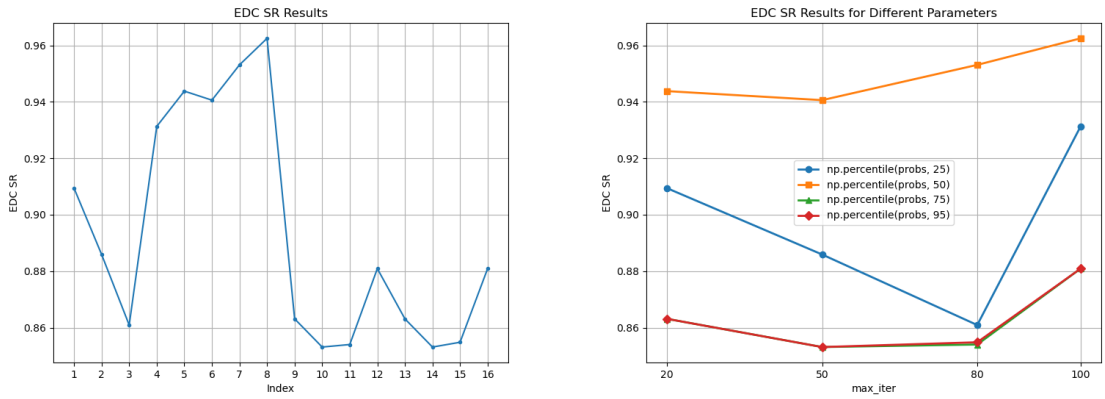


Figure 7.7: SR results of EDC under different parameter combinations. (Left) The horizontal axis shows the index corresponding to each parameter combination listed in Table 7.8, while the vertical axis displays the corresponding SR value. (Right) SR results plotted for each parameter combination, highlighting the effect of different settings on the SR results.

Tau has a greater impact on SR than batch size. As can be seen from the figure, a higher tau actually obtains a lower SR. This shows that a lower tau value leads to an increase in the accuracy tolerance range, so that some candidates with lower accuracy are selected, making it more difficult to resist the interference caused by irrelevant features.

In summary, the SR of the Anchors method is mainly affected by the threshold and tau. Higher threshold and tau will reduce the ability of Anchors to resist interference. In our experiments, the parameter combination that achieves the highest SR is batch size=32, tau=0.1 and threshold=0.9. The parameter combination with the lowest SR is batch size=32, tau=0.15 and threshold=0.99.

Index	Parameters	Anchor SR Results
1	batch_size: 32, tau: 0.1, threshold: 0.9	0.8364
2	batch_size: 32, tau: 0.1, threshold: 0.95	0.7965
3	batch_size: 32, tau: 0.1, threshold: 0.99	0.7896
4	batch_size: 32, tau: 0.15, threshold: 0.9	0.7927
5	batch_size: 32, tau: 0.15, threshold: 0.95	0.7696
6	batch_size: 32, tau: 0.15, threshold: 0.99	0.7602
7	batch_size: 64, tau: 0.1, threshold: 0.9	0.8164
8	batch_size: 64, tau: 0.1, threshold: 0.95	0.8027
9	batch_size: 64, tau: 0.1, threshold: 0.99	0.8018
10	batch_size: 64, tau: 0.15, threshold: 0.9	0.8085
11	batch_size: 64, tau: 0.15, threshold: 0.95	0.7921
12	batch_size: 64, tau: 0.15, threshold: 0.99	0.7679

Table 7.9: SR results of the Anchors method under different parameter combinations.

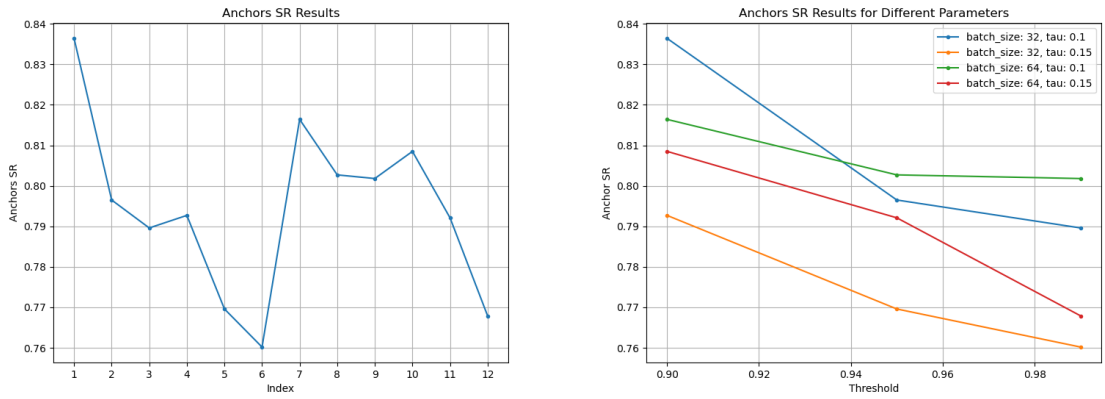


Figure 7.8: SR results of Anchors under different parameter combinations. (Left) The horizontal axis shows the index corresponding to each parameter combination listed in Table 7.9, while the vertical axis displays the corresponding SR value. (Right) SR results plotted for each parameter combination, highlighting the effect of different settings on the SR results.

7.1.3 Discussion of Local Explanation Methods

Figure 7.9 shows the CR and SR results of four different local explanation methods. According to the results of the final consistency rate (CR), EDC has the lowest CR, only 28.33%, indicating that the changes in the explanation features obtained by using different classification algorithms on the same feature set using the EDC method are more obvious. The second is Anchors, with a CR of only 33.25%. This result shows that the explanation features generated by the EDC and Anchors methods depend on the classification model used. The performance of the SHAP and LIME methods is better than that of EDC and Anchors, among which SHAP has a CR of 68.41%, and LIME has the highest CR of 69.94%. This shows that the changes in the explanation features obtained by these two explanation methods using different classification algorithms on the same feature set are relatively small, and have a higher consistency rate among the four explanation methods. This shows that the two explanation methods, SHAP and LIME, are relatively independent of the classification model used. From the consistency rate results, the performance of the explanation methods is LIME >SHAP >Anchors >EDC.

Regarding to the results of the final soundness rate (SR), SHAP has the highest SR value of 97.41%, which means that adding irrelevant features had no significant effect on the explanation results and could accurately capture the real important features. Anchors method performed the worst, with an SR of only 83.64%, which means that after adding irrelevant features to the original feature set, the explanation features obtained by Anchors changed greatly and were easily disturbed by noise. The SR of EDC reaches 96.25%, second only to SHAP. The SR of LIME reaches 96.08%, slightly lower than EDC. Therefore, from the SR results, SHAP, EDC and LIME all have high SR values, which means that after adding irrelevant features, these three explanation methods can still accurately identify the important features that really affect the model decision and have a certain ability to resist noise. In addition, the four explanation methods will not mistake irrelevant features for relevant

features and add them to the input. From the soundness rate results, the performance of the explanation methods is SHAP >EDC >LIME >Anchors.

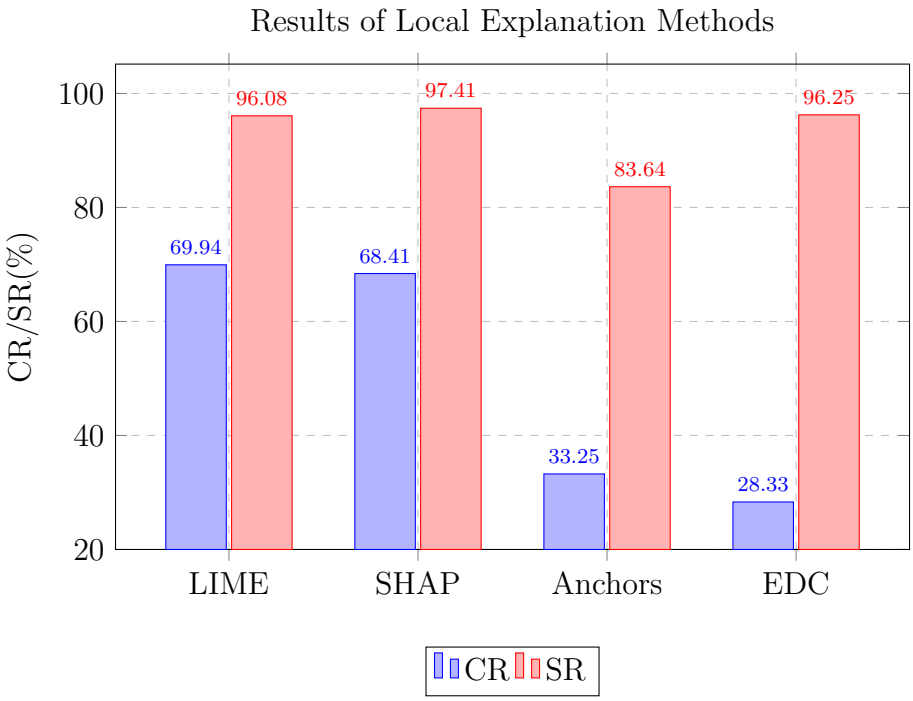


Figure 7.9: Performance of the chosen local explanation methods

Overall, combined with the evaluation metrics we selected for local explanation methods, the LIME method has the highest CR of 69.94%. SHAP is second with a CR of 68.41%. EDC has the lowest CR of only 28.33%. Anchors has a slightly higher CR than EDC, at 33.25%. For SR, SHAP is 97.41%, the highest among the four explanation methods, followed by EDC with SR of 96.25%. LIME ranks third with SR of 96.08%. The SR of Anchors is much lower than the other three methods, only 83.64%. It can be seen that LIME has the highest CR and SHAP achieves the highest SR. EDC has the lowest CR and Anchors has the lowest SR.

In Android malware detection, a high consistency rate means that even if different detection models are used, the explanation method can still output similar explanation results. Specifically, in practical applications, if the explanation method is too dependent on the detection model, it will cause the explanation method to obtain completely different explanation results under different models, making it difficult for analysts to accurately identify key features or behavior patterns related to malicious behavior, thereby reducing interpretability and credibility. The soundness rate indicates the anti-interference ability of the explanation method. That is, in the malware detection task, apk samples may contain a large number of redundant or irrelevant features. If the explanation method is too sensitive to these irrelevant features, it may cause the interpretation results to deviate from the essence of malicious behavior and mistakenly identify irrelevant features as key features, increasing the probability of misjudgment. Therefore, a higher soundness rate means that the explanation method can still accurately identify the real malicious behavior after adding irrelevant features and generate a stable explanation.

7.2. Global Explanation Results

In the global explanation experiment, we formulated three metrics to evaluate the performance of 4 feature-based global explanation methods, ALE, PD, PD Variance and Morris method on 4 classification models, SVM, RF, XGBoost and MLP. In this section, we discuss the results obtained based on the three metrics respectively.

Table 7.10 lists the parameters and descriptions used of each global explanation method in the experiment.

Parameter	Description	Method
low_resolution_threshold	Determine the feature values are used as grid points instead of quantiles to avoid jumps in the ALE plot that would mask the true effect.	ALE
min_bin_points	Determines how many bins each feature range is divided into, smaller and larger values result in less accurate local estimates because more of the feature range needs to be averaged.	ALE
grid_resolution	Enabled when the number of unique values of the target feature is greater than the grid resolution value. The number of equidistant points used to divide the range of each target feature.	PD, PDV
conf_level	Confidence interval level.	Morris
num_levels	The number of grid levels.	Morris
num_resamples	measures the resample numbers used to compute the confidence.	Morris

Table 7.10: The parameters of each global explanation methods.

7.2.1 Stability

In stability experiments, we want to check whether the set of explained features generated by an explanation method remains consistent across multiple runs.

ALE Stb Results

Table 7.12 shows the stability comparison of the explanation results of the four black-box models using the ALE method under different parameters. Among them, the stability of ALE based on XGB reaches 100%, which means that XGB is not affected by the parameter configuration of ALE and can always produce completely consistent interpretation results in multiple runs. ALE also achieved high stability on SVM, and the stability reached 100% when low_resolution_threshold was 5 and the number of minimum bins was 8, 32, and 50, and when low_resolution_threshold was 10 and the number of bins was 8 and 32. However, the stability performance of ALE on RF and MLP is relatively poor. RF achieves

a maximum stability of 82.19% and MLP achieves a maximum stability of 86.61% when low_resolution_threshold is equal to 5 and min_bin_points are 32 and 50, respectively.

When we focus on the parameter configuration of ALE, we can find that when threshold value is greater than or equal to 25, the number of bins has no effect on stability. ALE obtains the same stability results on the four models, and its stability is slightly higher than that in the interval of 15 to 20 for low_resolution_threshold. The number of bins has an effect when low_resolution_threshold is 5 and 10, and a relatively large number of bins is required for lower thresholds. However, in the low threshold range, ALE has the best stability among the models.

Overall, ALE method shows good stability on the XGB and SVM models, which means that the results generated by ALE on these two models remain unchanged over multiple runs. For RF and MLP, parameter configuration will affect their results. Choosing a high threshold or a low threshold but a relatively large number of bins can obtain better stability results, and the latter is better than the former. In addition, the stability fluctuations of RF and MLP may also be related to the randomness introduced by their models in addition to the influence of parameters. When using the ALE method in practice, the appropriate parameter combination should be selected according to the model type to avoid unstable interpretation due to too small bins or thresholds.

PD Stb Results

Table 7.12 shows the stability comparison of the interpretation results of the four black-box models using the PD method under different parameters. It is obvious from the results in the table that the stability of the PD method is not affected by the parameters, and each model obtains the same stability under different parameters. Both SVM and XGB achieved 100% stability, which means that after multiple runs, the explanations generated each time are exactly the same. RF and MLP achieved stability scores of 89.07% and 86.79%, respectively, which indicates that important features in the explanations changed slightly in one or more runs, affecting the overall stability. RF and MLP obtained stability scores of 89.07% and 86.79%, respectively, this indicates that important features in the explanation have changed slightly in one or more runs, affecting the overall stability. However, as described above, the stability of the PD method is independent of its parameters, so the change in important features in the explanation may be caused by the model itself. For example, MLP produces slight uncertainty during forward reasoning, or RF has small changes in the explanation results due to different sampling in each run.

From the overall results of PD stability, the method shows perfect stability on SVM and XGB, and also maintains relatively high stability on RF and MLP. This shows that PD can still produce relatively consistent interpretation results after multiple runs. The parameter independence of the PD method has great advantages over some interpretation methods that are sensitive to parameter configuration and is easier to use. Its higher stability can also improve the reliability of the interpretation results.

Parameters	SVM	RF	XGB	MLP
low_resolution_threshold: 5, min_bin_points: 1	98.53%	79.33%	100.00%	77.60%
low_resolution_threshold: 5, min_bin_points: 4	98.53%	78.67%	100.00%	78.88%
low_resolution_threshold: 5, min_bin_points: 8	100.00%	79.93%	100.00%	78.82%
low_resolution_threshold: 5, min_bin_points: 16	98.82%	80.33%	100.00%	80.72%
low_resolution_threshold: 5, min_bin_points: 32	100.00%	82.19%	100.00%	83.19%
low_resolution_threshold: 5, min_bin_points: 50	100.00%	81.91%	100.00%	86.61%
low_resolution_threshold: 10, min_bin_points: 1	97.81%	77.49%	100.00%	76.95%
low_resolution_threshold: 10, min_bin_points: 4	97.81%	77.04%	100.00%	77.47%
low_resolution_threshold: 10, min_bin_points: 8	97.21%	78.35%	100.00%	78.21%
low_resolution_threshold: 10, min_bin_points: 16	98.68%	77.88%	100.00%	79.21%
low_resolution_threshold: 10, min_bin_points: 32	100.00%	79.84%	100.00%	78.63%
low_resolution_threshold: 10, min_bin_points: 50	98.68%	78.05%	100.00%	80.70%
low_resolution_threshold: 15, min_bin_points: 1	97.07%	78.26%	100.00%	77.44%
low_resolution_threshold: 15, min_bin_points: 4	97.07%	77.67%	100.00%	77.30%
low_resolution_threshold: 15, min_bin_points: 8	97.07%	79.16%	100.00%	77.53%
low_resolution_threshold: 15, min_bin_points: 16	98.35%	78.11%	100.00%	77.54%
low_resolution_threshold: 15, min_bin_points: 32	97.74%	79.89%	100.00%	78.28%
low_resolution_threshold: 15, min_bin_points: 50	97.81%	79.40%	100.00%	78.91%
low_resolution_threshold: 20, min_bin_points: 1	97.07%	78.44%	100.00%	77.65%
low_resolution_threshold: 20, min_bin_points: 4	97.07%	78.02%	100.00%	77.63%
low_resolution_threshold: 20, min_bin_points: 8	97.07%	78.39%	100.00%	77.30%
low_resolution_threshold: 20, min_bin_points: 16	98.35%	77.84%	100.00%	77.81%
low_resolution_threshold: 20, min_bin_points: 32	97.74%	79.63%	100.00%	78.67%
low_resolution_threshold: 20, min_bin_points: 50	97.81%	78.61%	100.00%	77.91%
low_resolution_threshold: 25, min_bin_points: 1	97.74%	78.44%	100.00%	78.02%
low_resolution_threshold: 25, min_bin_points: 4	97.74%	78.02%	100.00%	78.00%
low_resolution_threshold: 25, min_bin_points: 8	97.74%	78.39%	100.00%	77.67%
low_resolution_threshold: 25, min_bin_points: 16	97.74%	77.84%	100.00%	78.05%
low_resolution_threshold: 25, min_bin_points: 32	97.74%	79.63%	100.00%	78.82%
low_resolution_threshold: 25, min_bin_points: 50	98.68%	78.61%	100.00%	78.60%
low_resolution_threshold: 30, min_bin_points: 1	97.74%	78.44%	100.00%	78.02%
low_resolution_threshold: 30, min_bin_points: 4	97.74%	78.02%	100.00%	78.00%
low_resolution_threshold: 30, min_bin_points: 8	97.74%	78.39%	100.00%	77.67%
low_resolution_threshold: 30, min_bin_points: 16	97.74%	77.84%	100.00%	78.05%
low_resolution_threshold: 30, min_bin_points: 32	97.74%	79.63%	100.00%	78.82%
low_resolution_threshold: 30, min_bin_points: 50	98.68%	78.61%	100.00%	78.60%

Table 7.11: ALE stability for different parameter combinations results based on SVM, RF, XGB, MLP. Min_bin_points determines how many bins each feature range is divided into, smaller and larger values result in less accurate local estimates because more of the feature range needs to be averaged. Low_resolution_threshold determine the feature values are used as grid points instead of quantiles to avoid jumps in the ALE plot that would mask the true effect.

Parameters	SVM	RF	XGB	MLP
grid_resolution: 10	100%	86.79%	100%	89.07%
grid_resolution: 20	100%	86.79%	100%	89.07%
grid_resolution: 30	100%	86.79%	100%	89.07%
grid_resolution: 50	100%	86.79%	100%	89.07%
grid_resolution: 100	100%	86.79%	100%	89.07%
grid_resolution: 150	100%	86.79%	100%	89.07%
grid_resolution: 200	100%	86.79%	100%	89.07%
grid_resolution: 250	100%	86.79%	100%	89.07%
grid_resolution: 300	100%	86.79%	100%	89.07%

Table 7.12: PD stability for different parameter combinations results based on SVM, RF, XGB, MLP. Grid_resolution is enabled when the number of unique values of the target feature is greater than the grid_resolution value. The number of equidistant points used to divide the range of each target feature.

PDV Stb Results

Table 7.13 shows the stability comparison of the interpretation results of the four black-box models using the PDV method under different parameters. It is clear from the table that XGB-based PDV achieved 100% stability, which means that XGB is parameter-independent and the important features in its interpretation results remain exactly the same in each run. However, it is worth mentioning the stability results of the other three models. The stability of PDV on SVM and RF always remains in the range of 17% to 22% under different parameter configurations, and the stability of MLP only fluctuates in the range of 17.2% to 17.9%. This means that the important features extracted by the PDV method based on these three models are almost completely different after multiple runs, that is, each run will produce different important features.

From the above research results, it can be seen that the PDV method maintains good stability only when using the XGB model, while for SVM, RF and MLP, the stability of the PDV method is extremely poor. At a stability of 17.2% to 22%, it is meaningless to discuss whether this instability is caused by the parameter configuration of the interpretation method or by the uncertainty of the model itself. The almost completely different interpretation results brought by each run greatly reduce the credibility of the interpretation method, because the results it produces will confuse analysts and cannot be trusted. Therefore, we can conclude that the PDV method is not suitable for the domain of Android malware detection, but its performance on XGB needs to be further verified in combination with other metrics mentioned later in this Section.

Morris Stb Results

Table 7.14 shows the stability comparison of the interpretation results of the four black-box models using the Morris method under different parameters. From the results, it can be seen that the Morris method has achieved high stability on SVM, XGB and MLP, among which the stability on XGB is the highest, reaching 96.98%, while SVM and MLP have achieved

Parameters	SVM	RF	XGB	MLP
grid_resolution: 10	21.67%	21.98%	100%	17.89%
grid_resolution: 20	19.12%	22.11%	100%	17.40%
grid_resolution: 30	19.42%	21.77%	100%	17.49%
grid_resolution: 50	19.42%	19.70%	100%	17.49%
grid_resolution: 100	19.51%	21.12%	100%	17.25%
grid_resolution: 150	19.56%	21.91%	100%	17.35%
grid_resolution: 200	19.56%	20.46%	100%	17.35%
grid_resolution: 250	19.51%	21.02%	100%	17.19%
grid_resolution: 300	19.51%	20.25%	100%	17.19%

Table 7.13: PDV stability for different parameter combinations results based on SVM, RF, XGB, MLP. The measurement of grid_resolution is the same as PD.

the highest stability of 93.65% and 91.49% respectively. This indicates that the importance features generated by the Morris method on SVM, XGB, and MLP vary relatively little between multiple runs, and can generate more consistent interpretation results. However, stability of Morris on RF is poor, with the highest being only 71.79%, this shows that the interpretation results generated by Morris on RF show obvious differences between multiple runs.

According to different parameter configurations, the results did not show a clear upward or downward trend after increasing the confidence interval level, so this parameter has almost no substantial effect on the stability of the interpretation results. This shows that the fluctuation of stability is jointly affected by the number of grid levels and the number of resamples used to calculate confidence, or due to the uncertainty of the model itself. However, from the overall results, the stability of each model for different parameter configurations does not change much. For SVM, the change range is between 91.63% and 93.65%; the change range of RF is between 69.61% and 71.79%; the overall change range of XGB is between 95.89% and 97.18%, and the stability of MLP is between 90.32% and 91.49%, and its maximum change range is only about 2%.

In summary, the interpretation results generated by the Morris method on SVM, XGB, and MLP are highly stable, with the best performance on XGB. However, the stability on RF is poor, indicating that the interpretation results fluctuate greatly between multiple runs. The Morris method is relatively insensitive to changes in parameter configuration. Choosing the appropriate number of grid levels and num_resamples can slightly improve stability to a certain extent, but it will not bring about a huge change.

7.2.2 Robustness

In this part of the experiment, we want to examine the effect of perturbations of features on the generated explanations. That is, we want to check whether the explanations generated after applying perturbations to the input are consistent with the explanations generated using the original features.

Parameters	SVM	RF	XGB	MLP
conf_level: 0.95, num_levels: 2, num_resamples: 50	92.88%	70.46%	96.12%	90.68%
conf_level: 0.95, num_levels: 2, num_resamples: 100	92.16%	71.23%	96.79%	90.89%
conf_level: 0.95, num_levels: 2, num_resamples: 150	91.88%	70.51%	96.07%	90.89%
conf_level: 0.95, num_levels: 4, num_resamples: 50	93.05%	71.30%	96.68%	91.49%
conf_level: 0.95, num_levels: 4, num_resamples: 100	92.61%	70.04%	96.32%	90.81%
conf_level: 0.95, num_levels: 4, num_resamples: 150	92.32%	70.23%	96.47%	90.65%
conf_level: 0.95, num_levels: 8, num_resamples: 50	92.47%	70.86%	96.72%	90.91%
conf_level: 0.95, num_levels: 8, num_resamples: 100	93.23%	70.68%	95.91%	90.47%
conf_level: 0.95, num_levels: 8, num_resamples: 150	92.79%	70.32%	96.63%	91.25%
conf_level: 0.99, num_levels: 2, num_resamples: 50	92.81%	71.30%	96.79%	90.32%
conf_level: 0.99, num_levels: 2, num_resamples: 100	92.93%	71.28%	97.05%	91.26%
conf_level: 0.99, num_levels: 2, num_resamples: 150	93.65%	71.77%	96.11%	91.42%
conf_level: 0.99, num_levels: 4, num_resamples: 50	92.60%	70.16%	96.98%	90.56%
conf_level: 0.99, num_levels: 4, num_resamples: 100	93.26%	70.96%	96.12%	90.91%
conf_level: 0.99, num_levels: 4, num_resamples: 150	91.68%	69.61%	95.89%	90.51%
conf_level: 0.99, num_levels: 8, num_resamples: 50	92.25%	70.53%	95.96%	91.09%
conf_level: 0.99, num_levels: 8, num_resamples: 100	92.18%	71.79%	97.18%	91.00%
conf_level: 0.99, num_levels: 8, num_resamples: 150	91.63%	71.05%	97.00%	91.02%

Table 7.14: Morris stability for different parameter combinations results based on SVM, RF, XGB, MLP. Conf_level is confidence interval level, num_levels is the number of grid levels and num_resamples measures the resample numbers used to compute the confidence.

ALE Rob Results

Table 7.15 shows the robustness comparison of the explanation results of the four black-box models using the ALE method under different parameters. According to the results, the best performance is XGB with low_resolution_threshold=5, min_bin_points=50, with a robustness of 98.83%, and the lowest robustness is 97.33% when threshold=25, min_bin_points=16. The second best is SVM, with a robustness of 98.68% when threshold=30, min_bin_points=50, and the lowest is 91.50% when threshold=15, min_bin_points=1. Then comes RF, which reaches the best robustness of 93.67% when threshold=10, min_bin_points=4, and the lowest is 90.67% when threshold=5, min_bin_points=1. The lowest is MLP, reaching 85.50% when threshold=5, min_bin_points=32, and only 74.50% when threshold=20, min_bin_points=4. It can be seen that ALE shows good robustness on XGB, SVM and RF, reaching more than 90% even in the worst performance. In contrast, the robustness of MLP is relatively low, but it also reaches 85.5%. This means that the ALE method has good anti-interference ability when non-important features are slightly disturbed.

XGB and RF have small fluctuations under different parameter configurations, which shows that the robustness of ALE on these two models is relatively less dependent on parameter settings. However, the robustness values of MLP and SVM under different parameters show obvious differences. Especially for MLP, its fluctuation is 11%. The difference between the best and worst robustness of SVM is over 7%. This shows that its robustness is greatly affected by parameters.

In summary, the ALE method shows high robustness on different models. For XGB and RF, different parameter configurations have relatively weak effects on their robustness. For SVM and MLP, reasonable parameter configurations can greatly improve their robustness. From the overall results, the ALE method has good noise resistance and can avoid completely different results due to slight changes in feature values. Therefore, ALE can provide reliable explanation results.

PD Rob Results

Table 7.16 shows the robustness comparison of the interpretation results of the four black-box models using the PD method under different parameters. From Table 7.16, we can find that under different grid_resolution parameters, the robustness of the PD method of the four models is at a high level. The robustness of PD on each model is from high to low: SVM, with a highest of 98.67%, XGB, with a highest robustness of 91.67%, followed by RF with 91.33%, and finally MLP with 91.17%.

The robustness of PD fluctuates under different parameter values. Among them, the best robustness of PD on SVM is when grid_resolution is 100, and the lowest robustness is when grid_resolution is 50, with a robustness of 96.83% and a change of 1.84%. It can be seen that SVM is relatively insensitive to changes in grid_resolution. The fluctuation range of RF is between 86.17% and 91.17%. It can be seen that the difference in robustness changes when the parameter value changes is relatively obvious. The highest and lowest robust values of grid_resolution are 300 and 20, respectively. XGB performs best at 100 resolution, which is 91.67%, and the lowest at 250 resolution, which is 87.33%, with a change of

Parameters	SVM	RF	XGB	MLP
low_resolution_threshold: 5, min_bin_points: 1	95.00%	90.67%	98.00%	80.17%
low_resolution_threshold: 5, min_bin_points: 4	93.33%	92.50%	98.00%	82.23%
low_resolution_threshold: 5, min_bin_points: 8	94.83%	92.17%	98.67%	83.50%
low_resolution_threshold: 5, min_bin_points: 16	93.00%	92.17%	97.00%	83.50%
low_resolution_threshold: 5, min_bin_points: 32	96.33%	92.83%	94.83%	85.50%
low_resolution_threshold: 5, min_bin_points: 50	94.83%	92.67%	98.83%	84.50%
low_resolution_threshold: 10, min_bin_points: 1	92.50%	91.00%	97.83%	80.00%
low_resolution_threshold: 10, min_bin_points: 4	93.17%	93.67%	97.33%	80.33%
low_resolution_threshold: 10, min_bin_points: 8	93.17%	90.83%	97.00%	81.17%
low_resolution_threshold: 10, min_bin_points: 16	93.67%	91.67%	96.33%	81.83%
low_resolution_threshold: 10, min_bin_points: 32	96.50%	93.17%	97.33%	84.00%
low_resolution_threshold: 10, min_bin_points: 50	96.50%	90.83%	98.17%	83.33%
low_resolution_threshold: 15, min_bin_points: 1	91.50%	92.17%	98.50%	81.33%
low_resolution_threshold: 15, min_bin_points: 4	92.67%	91.33%	98.17%	81.67%
low_resolution_threshold: 15, min_bin_points: 8	92.33%	92.17%	97.50%	80.17%
low_resolution_threshold: 15, min_bin_points: 16	93.17%	93.33%	97.00%	82.67%
low_resolution_threshold: 15, min_bin_points: 32	92.33%	91.83%	98.17%	83.50%
low_resolution_threshold: 15, min_bin_points: 50	93.67%	92.67%	97.50%	80.00%
low_resolution_threshold: 20, min_bin_points: 1	93.17%	93.00%	98.00%	78.83%
low_resolution_threshold: 20, min_bin_points: 4	91.33%	93.00%	98.83%	74.50%
low_resolution_threshold: 20, min_bin_points: 8	92.50%	93.33%	96.17%	82.83%
low_resolution_threshold: 20, min_bin_points: 16	92.67%	91.00%	97.17%	84.17%
low_resolution_threshold: 20, min_bin_points: 32	92.83%	92.33%	97.67%	80.00%
low_resolution_threshold: 20, min_bin_points: 50	93.67%	92.00%	98.17%	81.17%
low_resolution_threshold: 25, min_bin_points: 1	91.00%	92.17%	98.50%	81.00%
low_resolution_threshold: 25, min_bin_points: 4	93.50%	93.33%	97.50%	81.83%
low_resolution_threshold: 25, min_bin_points: 8	92.17%	92.83%	97.83%	81.00%
low_resolution_threshold: 25, min_bin_points: 16	90.67%	92.67%	97.33%	78.67%
low_resolution_threshold: 25, min_bin_points: 32	92.50%	91.83%	97.83%	80.33%
low_resolution_threshold: 25, min_bin_points: 50	93.17%	92.00%	97.83%	78.50%
low_resolution_threshold: 30, min_bin_points: 1	93.33%	92.17%	97.67%	78.83%
low_resolution_threshold: 30, min_bin_points: 4	91.83%	92.67%	97.50%	81.33%
low_resolution_threshold: 30, min_bin_points: 8	93.00%	92.50%	98.00%	81.67%
low_resolution_threshold: 30, min_bin_points: 16	92.33%	92.33%	97.33%	81.00%
low_resolution_threshold: 30, min_bin_points: 32	91.83%	92.67%	97.17%	82.83%
low_resolution_threshold: 30, min_bin_points: 50	98.68%	92.61%	97.33%	82.67%

Table 7.15: ALE robustness for different parameter combinations results based on SVM, RF, XGB, MLP.

4.34%, and its sensitivity to parameter changes is slightly lower than RF. The change range of MLP is between 87.17% and 91.00%, indicating that parameter changes play a certain role in robustness.

From the above observations, we can find that PD has the highest robustness on SVM, and SVM is relatively less affected by parameter changes. PD has the worst robustness on MLP, which is 91.17%. However, overall, the PD method has shown high robustness on all four models, indicating that the method has good anti-interference ability. In addition, we also found that different grid_resolution values have different effects on the models. For example, when grid_resolution is 100, PD has the highest robustness on SVM and XGB, but for RF and MLP, the value is 20 and 200. This shows that choosing the right grid_resolution for different models can improve the robustness of the explanation method to a certain extent.

Parameters	SVM	RF	XGB	MLP
grid_resolution: 10	97.17%	89.17%	90.83%	89.83%
grid_resolution: 20	97.67%	91.33%	91.00%	88.33%
grid_resolution: 30	97.50%	89.00%	89.17%	87.83%
grid_resolution: 50	96.83%	87.00%	89.00%	87.67%
grid_resolution: 100	98.67%	88.67%	91.67%	90.67%
grid_resolution: 150	97.67%	87.83%	91.17%	87.17%
grid_resolution: 200	97.00%	87.50%	90.83%	91.17%
grid_resolution: 250	98.33%	86.17%	87.33%	88.00%
grid_resolution: 300	97.67%	88.67%	89.33%	91.00%

Table 7.16: PD robustness for different parameter combinations results based on SVM, RF, XGB, MLP.

PDV Rob Results

Table 7.17 shows the robustness comparison of the interpretation results of the four black-box models using the PDV method under different parameters. From the results in the table, we can see that the robustness of the PDV method is extremely low, and it performance under different parameters is extremely poorly in all four models. This shows that although we have selected a small number of features that are not the most important for perturbation, the explanation method itself lacks robustness and is therefore extremely sensitive to small changes. This result also shows that the interpretation results of the PDV method are very susceptible to even the slightest changes. Therefore, the interpretation results generated by this method are difficult to trust. In other words, combined with the previous experimental results on stability, it can be concluded that the PDV method is not suitable for Android malware detection because the interpretation results it generates are extremely unstable and can be easily affected by slight changes, thereby increasing the risk of misleading, making it difficult for security analysts to identify the authenticity of the interpretation results.

Parameters	SVM	RF	XGB	MLP
grid_resolution: 10	21.33%	22.17%	30.50%	19.17%
grid_resolution: 20	19.00%	21.17%	32.00%	18.50%
grid_resolution: 30	16.50%	22.50%	24.00%	17.67%
grid_resolution: 50	17.33%	20.67%	29.00%	21.00%
grid_resolution: 100	18.33%	21.17%	30.33%	17.17%
grid_resolution: 150	17.33%	22.67%	32.83%	19.83%
grid_resolution: 200	19.17%	22.83%	29.00%	16.83%
grid_resolution: 250	19.17%	22.67%	23.33%	19.83%
grid_resolution: 300	16.83%	20.67%	39.00%	20.17%

Table 7.17: PDV robustness for different parameter combinations results based on SVM, RF, XGB, MLP.

Morris Rob Results

Table 7.18 shows the robustness comparison of the interpretation results of the four black-box models using the Morris method under different parameters. From the results in Table 7.18, it can be seen that the robustness of Morris method on the four models is generally high, all above 92%. From the overall average level, Morris has the highest robustness on RF, which is 96.00%; MLP is second, which is 95.67%. Morris has a robustness of 95.17% on XGB, and the last is SVM, 93.17%. The results show that when a small number of eigenvalues are perturbed, the Morris method can remain undisturbed and still identify the truly important features, with good noise resistance.

According to the robustness of each model under different parameters, the robustness of the Morris method is not so sensitive to parameter changes, and each model fluctuates within a very small range. This shows that the parameters of the Morris method have a relatively small impact on its robustness, and there is no need to over-adjust the parameters. The best robustness parameter settings for each model are: conf level: 0.99, num levels: 8, num resamples: 50 for RF; conf level: 0.95, num levels: 2, num resamples: 100 for MLP; conf level: 0.95, num levels: 4, num resamples: 100 for XGB and conf level: 0.95, num levels: 8, num resamples: 50 and conf level: 0.99, num levels: 4, num resamples: 50 for SVM.

In summary, Morris has shown high robustness on different models and can avoid interference caused by very small eigenvalue perturbations. Secondly, Morris's robustness is less dependent on parameters and can achieve high results without a large number of parameter adjustments. However, using different parameter configurations for different models can slightly improve its robustness to a certain extent.

7.2.3 Effectiveness

The effectiveness suggests that the explanatory features generated by the explanation method should be the key to the model decision-making basis. Therefore, we first test the model prediction results with the original features, and then compare them with the prediction of model results after removing the explanatory features generated by the expla-

Parameters	SVM	RF	XGB	MLP
conf_level: 0.95, num_levels: 2, num_resamples: 50	92.67%	95.33%	94.67%	95.33%
conf_level: 0.95, num_levels: 2, num_resamples: 100	92.67%	95.83%	94.83%	95.67%
conf_level: 0.95, num_levels: 2, num_resamples: 150	93.00%	95.00%	94.33%	95.17%
conf_level: 0.95, num_levels: 4, num_resamples: 50	92.33%	94.67%	94.33%	95.50%
conf_level: 0.95, num_levels: 4, num_resamples: 100	93.00%	93.67%	95.17%	94.67%
conf_level: 0.95, num_levels: 4, num_resamples: 150	92.67%	95.17%	94.50%	95.50%
conf_level: 0.95, num_levels: 8, num_resamples: 50	93.17%	95.33%	93.50%	94.83%
conf_level: 0.95, num_levels: 8, num_resamples: 100	92.83%	94.17%	94.50%	94.83%
conf_level: 0.95, num_levels: 8, num_resamples: 150	92.83%	95.00%	94.33%	95.50%
conf_level: 0.99, num_levels: 2, num_resamples: 50	92.83%	94.83%	94.67%	95.50%
conf_level: 0.99, num_levels: 2, num_resamples: 100	92.67%	94.67%	94.33%	94.83%
conf_level: 0.99, num_levels: 2, num_resamples: 150	92.33%	95.17%	94.17%	95.33%
conf_level: 0.99, num_levels: 4, num_resamples: 50	93.17%	95.00%	94.50%	95.50%
conf_level: 0.99, num_levels: 4, num_resamples: 100	92.83%	93.83%	94.83%	94.83%
conf_level: 0.99, num_levels: 4, num_resamples: 150	92.83%	94.50%	94.17%	95.50%
conf_level: 0.99, num_levels: 8, num_resamples: 50	92.67%	96.00%	94.50%	95.17%
conf_level: 0.99, num_levels: 8, num_resamples: 100	92.67%	94.17%	95.00%	95.50%
conf_level: 0.99, num_levels: 8, num_resamples: 150	92.50%	95.17%	94.67%	95.67%

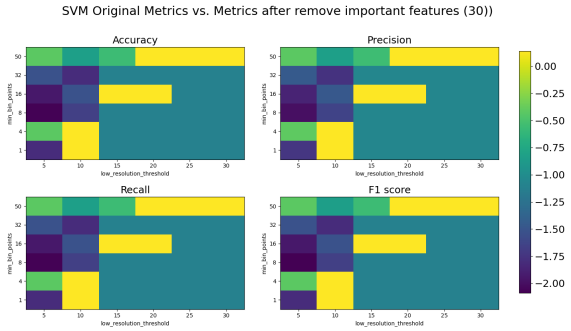
Table 7.18: Morris robustness for different parameter combinations results based on SVM, RF, XGB, MLP.

nation methods.

ALE Eff Results

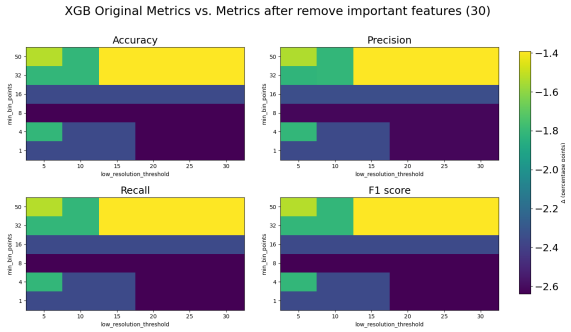
Figure 7.10 shows the original performance metrics of accuracy, precision, recall and F1 score of the four black-box models SVM, RF, XGBoost and MLP, and their changes after removing the 30 important features identified by the ALE method under different parameter combinations.

From the results in the figure, we can see that after deleting 30 important features, the performance metrics of each model are almost the same as those of the model using the original features, especially RF and MLP. The performance metrics of the four models did not drop below the threshold (5%), indicating that the number of important features that affect the model decision is more than 30. Therefore, deleting only 30 important features is not enough to affect the final result, and the model can still obtain sufficient effective information from other features.



(a) SVM

(b) RF



(c) XGBoost

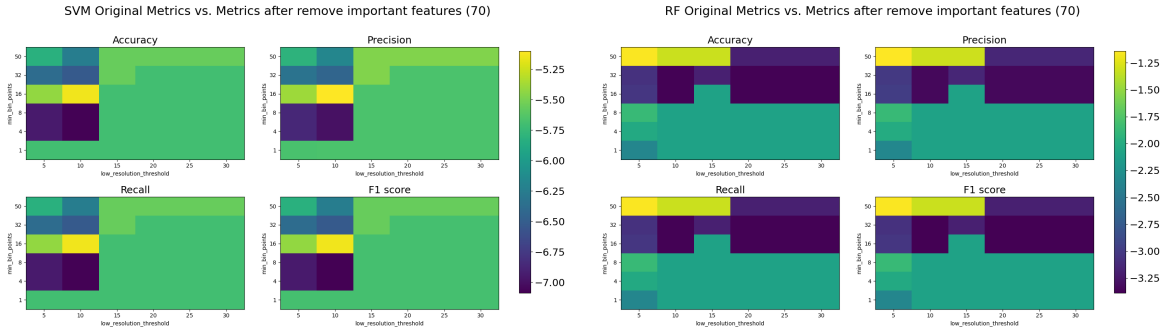
(d) MLP

Figure 7.10: Effect of removing 30 important features identified by ALE on four performance metrics (accuracy, precision, recall and F1 score) of four black-box models at different combinations of low_resolution_threshold (x-axis) and min_bin_points (y-axis). The colored bars in each subplot represent the percentage point change of each metric before and after removal, darker color indicate larger changes, and lighter color indicate smaller changes.

We increase the number of features to be removed to 70, which is about 10% of the total number of features. Figure 7.11 shows the original performance metrics of accuracy, precision, recall and F1 score of the four black-box models SVM, RF, XGBoost and MLP, and their changes after removing the 70 important features identified by the ALE method under different parameter combinations.

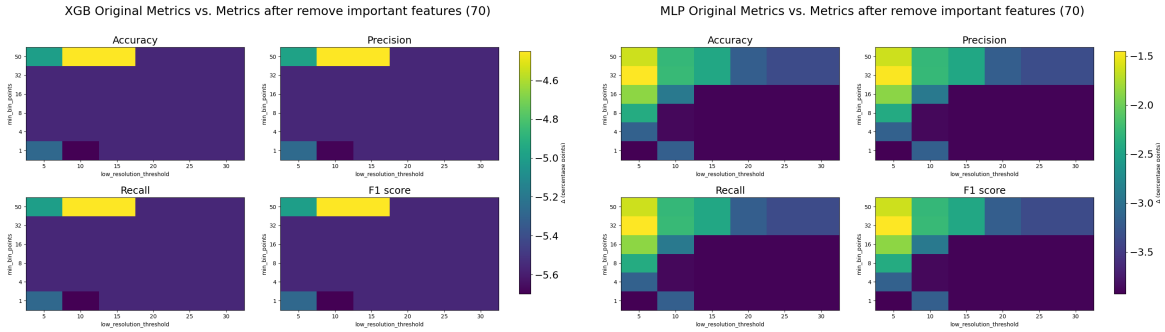
According to the results in Figure 7.11, after deleting the 70 features identified by the ALE method under different parameters, the performance of the SVM method dropped beyond the threshold we specified, followed by XGBoost, and most of the results also showed a significant drop. For RF and MLP, their performance dropped, but did not exceed the threshold. This shows that for SVM and XGBoost, 70 important features have already affected the decision-making process of the model. For RF and MLP, although this number has affected the decision-making of the model to a certain extent, it is obviously not enough.

To further verify the effectiveness of the explanation method, we continue to increase the number of deleted important features to 100. The result of remove 100 features as shown in Figure 7.12.



(a) SVM

(b) RF



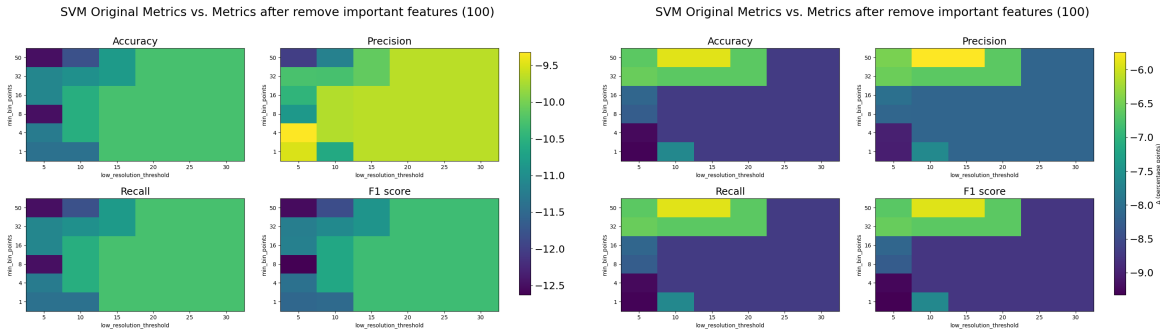
(c) XGBoost

(d) MLP

Figure 7.11: Effect of removing 70 important features identified by ALE on four performance metrics (accuracy, precision, recall and F1 score) of four black-box models at different combinations of low_resolution_threshold (x-axis) and min_bin_points (y-axis). The colored bars in each subplot represent the percentage point change of each metric before and after removal, darker color indicate larger changes, and lighter color indicate smaller changes.

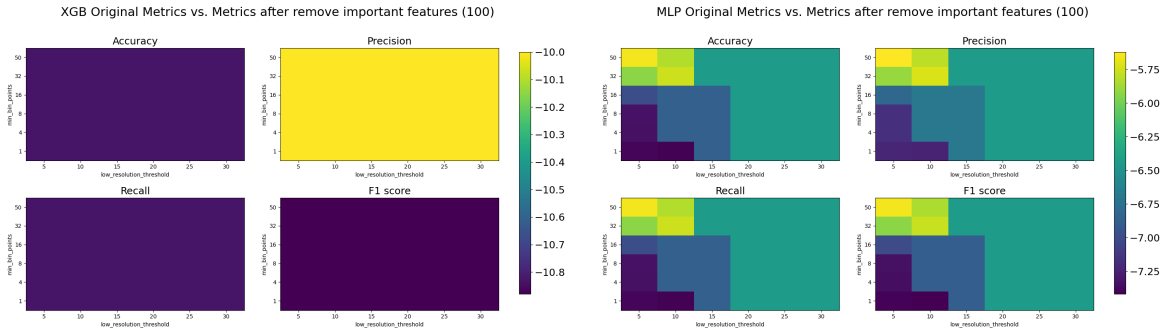
From the results in Figure 7.12, after removing 100 important features, the performance metrics of SVM, RF, XGBoost and MLP all dropped by more than the 5% threshold. Among them, by deleting the important features generated by the ALE method under different parameters, the drop of SVM is 9.5% - 12.5%, the drop of RF is 6% - 9%, the drop of XGBoost is about 10% to 11%, and the drop of MLP is from 5.75% to 7.25%. In addition, we also found that when the important features generated by ALE under low thresholds and fewer bins were removed, the performance of SVM, RF and MLP changed the most. Besides, SVM also showed sensitivity to low thresholds and high bins, while XGBoost was not affected by ALE parameters.

However, as we discussed in the 6.3.2 section, a potential problem is that if we remove enough features, even if they are unimportant features, the performance of the model may deteriorate due to the reduction in the amount of information available for the model to learn or the change in the data distribution. To make the effectiveness results more reliable, we added a verification step, that is, if the features we removed are important features that truly affect the basis for the model decision, then even if only these 100 important features are used, the performance of the model should be close to the original performance. Given the above results on the impact of ALE of different parameters on model performance, we only need to verify the parameter combination that causes the largest drop in model performance. That is, the thresholds are 5, the number of bins is 1, 4, 8, 50, and threshold is 10, bins are 1. The verification results are shown in Table 7.19.



(a) SVM

(b) RF



(c) XGBoost

(d) MLP

Figure 7.12: Effect of removing 100 important features identified by ALE on four performance metrics (accuracy, precision, recall and F1 score) of four black-box models at different combinations of low_resolution_threshold (x-axis) and min_bin_points (y-axis). The colored bars in each subplot represent the percentage point change of each metric before and after removal, darker color indicate larger changes, and lighter color indicate smaller changes.

Table 7.19 shows the comparison of model performance when only 100 important features are used and when the original features are used. It can be seen that the accuracy, recall, and F1 scores of XGB using the original features are all 98.89%, and the precision is 98.90%, while the accuracy, recall, and F1 scores of XGB using only 100 important features are all 98.61%, and the precision is 98.62%. It can also be seen from the results that the results of XGB are not affected by the ALE parameters.

The accuracy, recall, and F1 scores of MLP using the original features are all 98.53%, and the precision is 98.54%. The accuracy, precision, recall, and F1 scores of MLP using only 100 important features are all 98.30%. The important features generated by ALE at a threshold of 5 and a bin number of 4 are closest to the original data.

The accuracy, precision, recall, and F1 score of the SVM using the original features are all 97.64%. Except for the important features generated by ALE with a threshold of 5 and a bin number of 50, which has a performance index of 97.5%, the performance of the important features under the other parameters is exactly the same as the original data.

The accuracy, precision, recall, and F1 scores of RF using the original features are all

98.46%. The accuracy, precision, recall, and F1 scores of the 100 important features generated by ALE at a threshold of 5 and a bin number of 50 are closest to the original data, all of which are 98.45%.

From these results, it can be seen that the performance of the model using only 100 important features, although not completely reaching the performance of the model using the original features, is very close to the results. This shows that the important features generated by the ALE method do have the largest marginal contribution to the decision process of the model, so the impact is the largest when these features are removed. However, since the interactions between minor features can still provide a small amount of information, reconstructing the decision boundary using only these important features may result in a slight decrease in performance. In other words, the 100 important features generated by the ALE method capture most of the key information needed for model prediction, thus proving the effectiveness of the method. Judging from the performance of the four models, ALE is most effective on SVM.

	Orig (Acc, Prec, Recall, F1)	After Remove (Acc, Prec, Recall, F1)	Parameter
SVM	97.64%, 97.64%, 97.64%, 97.64%	97.64%, 97.64%, 97.64%, 97.64%	t=5, b=1
		97.64%, 97.64%, 97.64%, 97.64%	t=5, b=4
		97.64%, 97.64%, 97.64%, 97.64%	t=5, b=8
		97.50%, 97.50%, 97.50%, 97.50%	t=5, b=50
		97.64%, 97.64%, 97.64%, 97.64%	t=10, b=1
RF	98.46%, 98.46%, 98.46%, 98.46%	98.35%, 98.36%, 98.35%, 98.35%	t=5, b=1
		98.36%, 98.36%, 98.36%, 98.36%	t=5, b=4
		98.33%, 98.33%, 98.33%, 98.33%	t=5, b=8
		98.45%, 98.45%, 98.45%, 98.45%	t=5, b=50
		98.44%, 98.44%, 98.44%, 98.44%	t=10, b=1
XGB	98.89%, 98.90%, 98.89%, 98.89%	98.61%, 98.62%, 98.61%, 98.61%	t=5, b=1
		98.61%, 98.62%, 98.61%, 98.61%	t=5, b=4
		98.61%, 98.62%, 98.61%, 98.61%	t=5, b=8
		98.61%, 98.62%, 98.61%, 98.61%	t=5, b=50
		98.61%, 98.62%, 98.61%, 98.61%	t=10, b=1
MLP	98.52%, 98.53%, 98.52%, 98.52%	98.24%, 98.25%, 98.24%, 98.24%	t=5, b=1
		98.30%, 98.30%, 98.30%, 98.30%	t=5, b=4
		98.24%, 98.25%, 98.24%, 98.24%	t=5, b=8
		98.28%, 98.28%, 98.28%, 98.28%	t=5, b=50
		98.29%, 98.30%, 98.29%, 98.29%	t=10, b=1

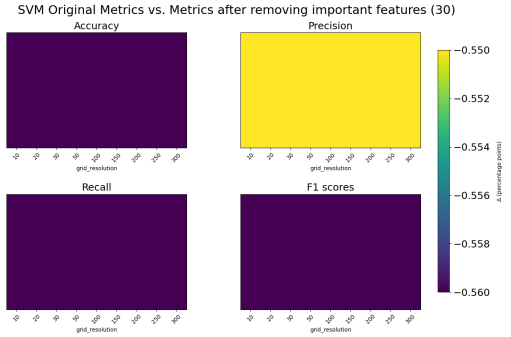
Table 7.19: Performance results of four black-box models using only 100 features generated by ALE. T refers to the low_resolution_threshold and b refers to the min_bin_points.

PD Eff Results

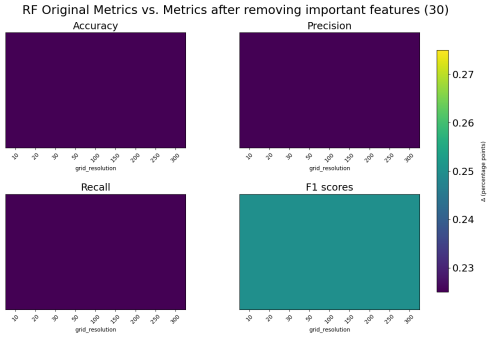
Figure 7.13 shows the original performance metrics of accuracy, precision, recall and F1 score of the four black-box models SVM, RF, XGBoost and MLP, and their changes after

removing the 30 important features identified by the PD method under different parameter combinations.

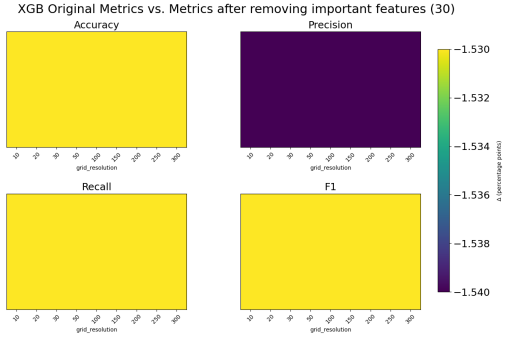
It can be seen from the results in the figure 7.13 that after deleting 30 important features, the performance metrics of each model did not change significantly, and none of them fell below the threshold (5%). This shows that the number of important features that affect model decisions is far more than 30. Therefore, deleting only 30 important features is not enough to affect the final result, and the model can still obtain enough effective information from other features. Secondly, after deleting the features generated by the PD method under different parameters, the performance metrics of the model decreased at the same rate under different parameters, indicating that the effectiveness of PD is not affected by the parameter changes of the explanation method.



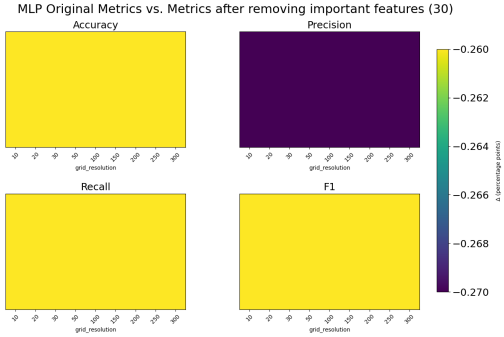
(a) SVM



(b) RF



(c) XGBoost



(d) MLP

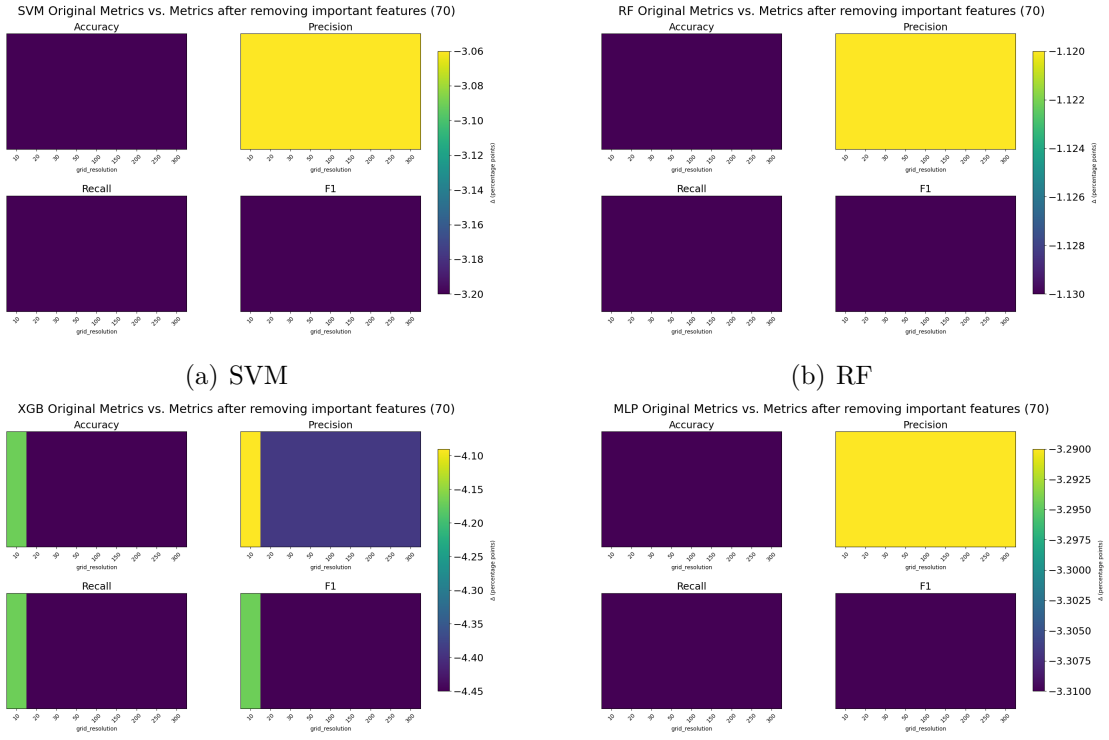
Figure 7.13: Effect of removing 30 important features identified by PD on four performance metrics (accuracy, precision, recall and F1 score) of four black-box models at different parameters of grid_resolution(x-axis). The colored bars in each subplot represent the percentage point change of each metric before and after removal, darker color indicate larger changes, and lighter color indicate smaller changes.

Since removing 30 features is not enough to affect the decision-making process of the model, we increase the number of removed features to 70. Figure 7.14 shows the change in the performance of each model after removing 70 important features compared to the original performance metrics.

From the results in the figure, we can find that after deleting 70 important features, the performance of SVM and MLP decreased by about 3%, while the performance of XGB

decreased by about 4%. This shows that the deleted features have affected the decision-making process of the model to a certain extent. Although it has not reached the threshold, it has shown a significant downward trend compared with deleting 30 features. However, the performance of RF has only decreased by 1%. Although there is also a significant downward trend compared with deleting 30 features, 70 features are still not enough to affect the decision-making process of the model.

Therefore, we increase the number of important features to be deleted to 100 to further observe the effectiveness of the explanation method.



(a) SVM

(b) RF

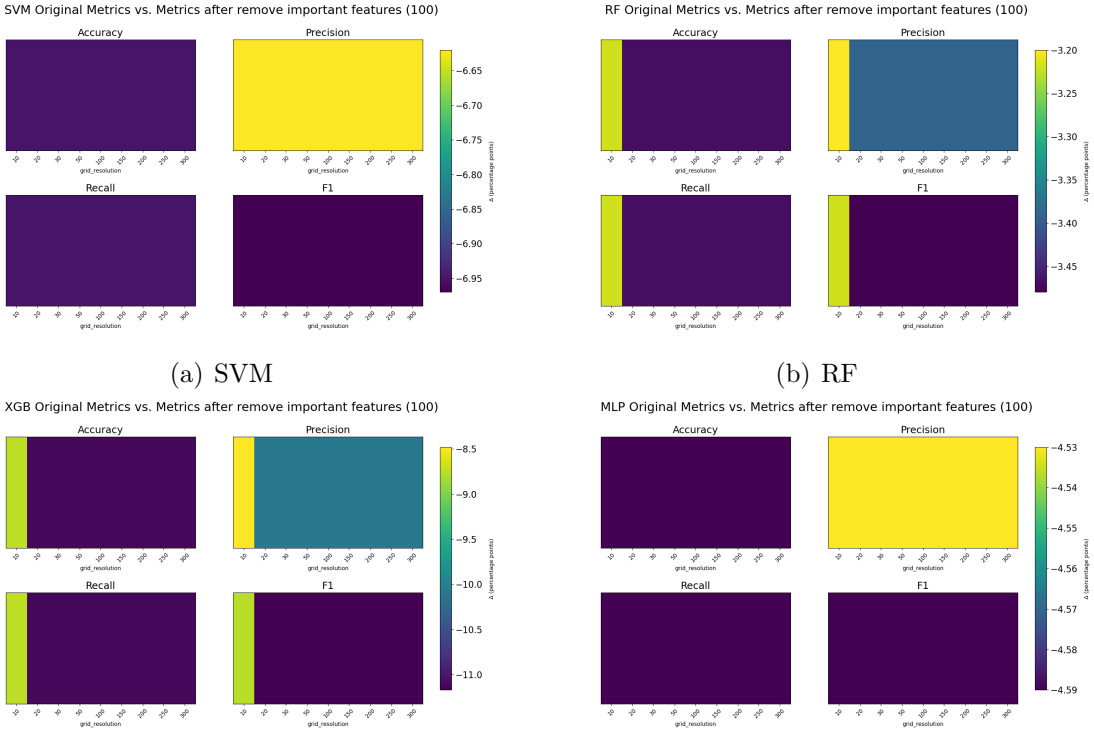
(c) XGBoost

(d) MLP

Figure 7.14: Effect of removing 70 important features identified by PD on four performance metrics (accuracy, precision, recall and F1 score) of four black-box models at different combinations of grid_resolution. The colored bars in each subplot represent the percentage point change of each metric before and after removal, darker color indicate larger changes, and lighter color indicate smaller changes.

The performance comparison between the model removed with 100 features generated based on the PD method deleted and the model with the original features is shown in Figure 7.15. It can be seen that after removing 100 important features, the performance of SVM and XGBoost both dropped by more than 5%. It is worth mentioning that although the performance of RF and MLP also dropped to a large extent, it did not exceed the threshold. Among them, the maximum performance drop of RF was about 3.5%, while that of MLP was about 4.6%. Given that the drop of MLP and RF was already close to the threshold range, we increased the number of important features deleted to 120. After deleting 120 features, the performance drop of RF and MLP exceeded the threshold. In other words, for the PD method, about 120 features are needed to be sufficient to affect

the model prediction results.
Then, based on this finding, we verified the effectiveness of the PD method. From 7.15, we can see that when grid_resolution is greater than 10, RF and XGBoot decrease more. SVM and MLP are not affected by the PD parameter. Therefore, we select grid_resolution of 20, 100 and 250 for verification. The results are shown in Table 7.20.



(a) SVM (b) RF

(c) XGBoost (d) MLP

Figure 7.15: Effect of removing 100 important features identified by PD on four performance metrics (accuracy, precision, recall and F1 score) of four black-box models at different combinations of grid_resolution. The colored bars in each subplot represent the percentage point change of each metric before and after removal, darker color indicate larger changes, and lighter color indicate smaller changes.

According to the results in Table 7.20, the model performance is relatively insensitive to the parameters of the PD method. Among them, XGB only uses 120 important features to obtain the same results as the original features. The performance of SVM, RF and MLP is slightly reduced when only 120 features are used, but it is not much different from the original results. Therefore, it can be seen that the explanatory features generated by the PD method are indeed an important basis for model decision-making, and their number accounts for about 17% of the total number of features. And PD has the best effect on XGB.

PDV Eff Results

The performance comparison of the models after deleting the 30 important features and removing 70 important features selected according to the PDV method is shown in Figure 7.16 and Figure 7.17, respectively.

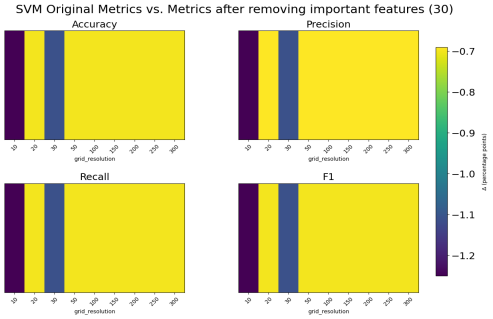
	Orig (Acc, Prec, Recall, F1)	After Remove (Acc, Prec, Recall, F1)	Parameter
SVM	97.64%, 97.64%, 97.64%, 97.64%	97.36%, 97.36%, 97.36%, 97.36%	grid=20
		97.36%, 97.36%, 97.36%, 97.36%	grid=100
		97.36%, 97.36%, 97.36%, 97.36%	grid=250
RF	98.46%, 98.46%, 98.46%, 98.46%	98.42%, 98.42%, 98.42%, 98.42%	grid=20
		98.42%, 98.42%, 98.42%, 98.42%	grid=100
		98.42%, 98.42%, 98.42%, 98.42%	grid=250
XGB	98.89%, 98.90%, 98.89%, 98.89%	98.89%, 98.90%, 98.89%, 98.90%	grid=20
		98.89%, 98.90%, 98.89%, 98.90%	grid=100
		98.89%, 98.90%, 98.89%, 98.90%	grid=250
MLP	98.52%, 98.53%, 98.52%, 98.52%	98.14%, 98.14%, 98.14%, 98.14%	grid=20
		98.14%, 98.14%, 98.14%, 98.14%	grid=100
		98.14%, 98.14%, 98.14%, 98.14%	grid=250

Table 7.20: Performance results of four black-box models using only 100 features generated by PD.

From the results of the model performance degradation in these two figures, whether deleting 30 or 70 important features, the model performance is almost unaffected. Only the performance of XGBoost dropped by more than 2%, and when the grid resolution was 10, the performance of SVM dropped by 2%. Therefore, according to the current results, it can be seen that the important features identified by the PDV method are unsufficient to affect the model decision. However, considering the performance of SVM and XGBoost, we still increase the number of important features deleted to 100 to observe the changes in the performance of each model.

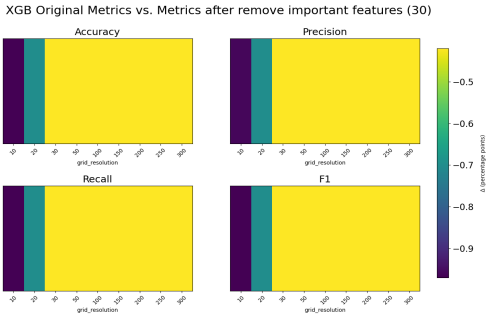
The result after removing 100 important features is shown in the Figure 7.18. It can be seen that the performance of MLP and RF after removing 100 features has almost no significant change compared to the original results. SVM only dropped by about 2% after removing PDV with a grid resolution of 10. This shows that the 100 features generated by PDV are not important features, so removing these features has almost no effect on the model decision. However, XGBoost dropped by more than 5%, which means that for XGBoost, the features generated by PDV may be able to affect its decision process. Given that the PDV effectiveness on other models is 0, we only validated the results of XGBoost to ensure its effectiveness. As can be seen from Figure 7.18, when the grid resolution of PDV is low, the performance of XGB drops significantly, so we validated the results generated by PDV when the grid resolution is 10, 20, and 30.
The verification results of XGB are shown in the Table 7.21.

It can be seen that the accuracy, precision, recall and F1 scores obtained by XGB using only 100 important features are all over 98%. Although the overall performance is good, there is still a gap with the original results. This shows that the explanation results generated by PDV can affect the decision of the model to a certain extent, but the results may contain

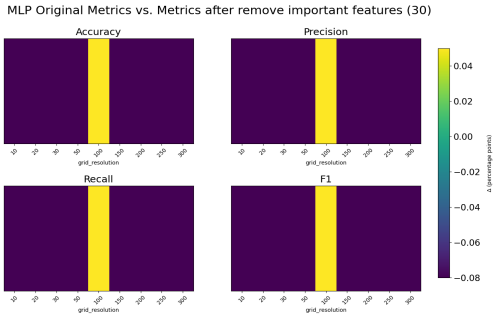


(a) SVM

(b) RF

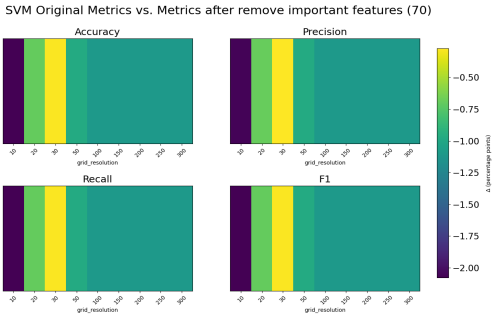


(c) XGBoost

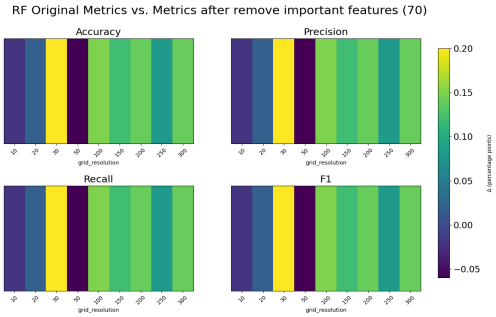


(d) MLP

Figure 7.16: Effect of removing 30 important features identified by PD on four performance metrics (accuracy, precision, recall and F1 score) of four black-box models at different parameters of grid_resolution(x-axis). The colored bars in each subplot represent the percentage point change of each metric before and after removal, darker color indicate larger changes, and lighter color indicate smaller changes.



(a) SVM



(b) RF

(c) XGBoost

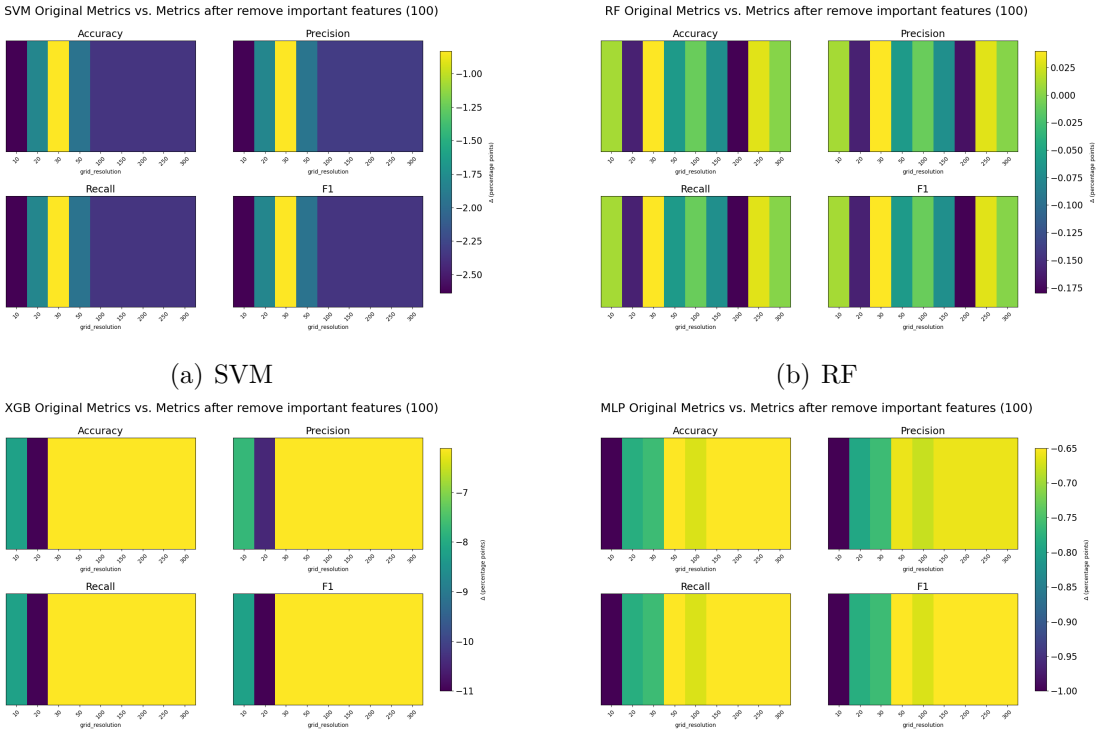
(d) MLP

Figure 7.17: Effect of removing 70 important features identified by PDV on four performance metrics (accuracy, precision, recall and F1 score) of four black-box models at different combinations of grid_resolution. The colored bars in each subplot represent the percentage point change of each metric before and after removal, darker color indicate larger changes, and lighter color indicate smaller changes.

	Orig (Acc, Prec, Recall, F1)	After Remove (Acc, Prec, Recall, F1)
grid=10	98.89%, 98.90%,98.89%, 98.89%	98.06%, 98.07%, 98.06%, 98.06%
grid=20	98.89%, 98.90%, 98.89%, 98.89%	98.19%, 98.20%, 98.19%, 98.19%
grid=30	98.89%, 98.90%, 98.89%, 98.89%	98.19%, 98.20%, 98.19%, 98.19%

Table 7.21: Performance results of XGBoost using only 100 features generated by PDV.

less important or unimportant features, which will have a certain impact on the prediction of the model. However, from the results, PDV is effective on XGBoost.



(a) SVM

(b) RF

(c) XGBoost

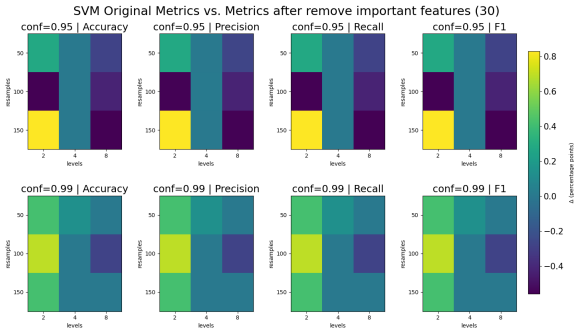
(d) MLP

Figure 7.18: Effect of removing 100 important features identified by PDV on four performance metrics (accuracy, precision, recall and F1 score) of four black-box models at different combinations of grid_resolution. The colored bars in each subplot represent the percentage point change of each metric before and after removal, darker color indicate larger changes, and lighter color indicate smaller changes.

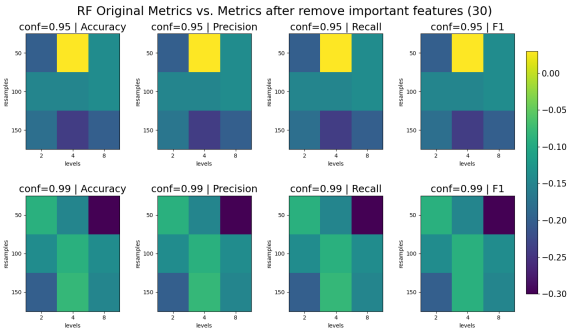
Morris Eff Results

The performance comparison of the models after deleting the 30 important features selected according to the Morris method is shown in Figure 7.19. From the results, we can see that removing only 30 features has almost no change in the performance of each model. This shows that 30 features are not enough to affect the prediction results of the model. Therefore, we increase this number to 70. The result is shown in Figure 7.20.

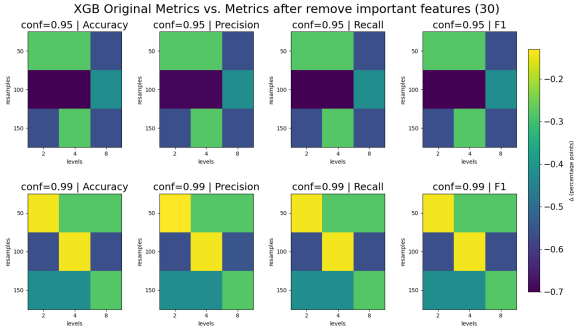
After deleting 70 features, the maximum decrease of XGBoost is close to 5%, and the maximum decrease of SVM and MLP is about 2.6%. However, the decrease of RF is still small, only about 1.5%. Based on this result, we continue to increase the number of deleted features to 100 to observe the changes in the performance of each model.



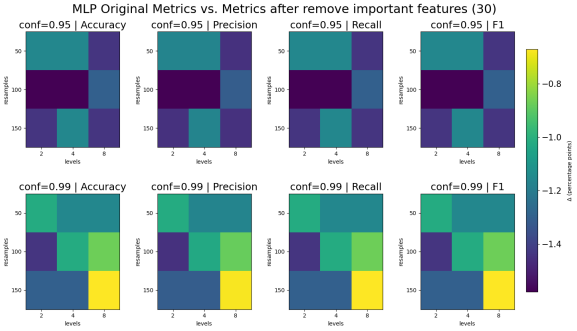
(a) SVM



(b) RF



(c) XGBoost



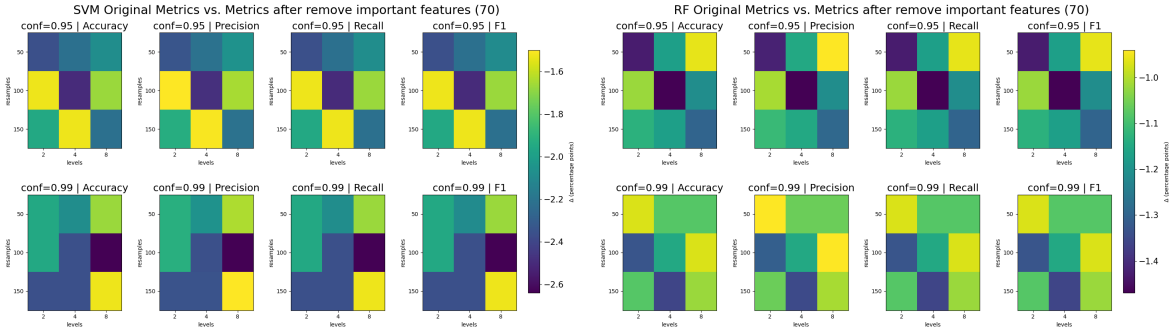
(d) MLP

Figure 7.19: Effect of removing 30 important features identified by Morris on four performance metrics (accuracy, precision, recall and F1 score) of four black-box models at different combinations of conf_level, num_levels and num_resamples. The colored bars in each subplot represent the percentage point change of each metric before and after removal, darker color indicate larger changes, and lighter color indicate smaller changes.

The comparison between the original result and the result after deleting 100 features is shown in Figure 7.21. It can be seen that after deleting 100 important features, the performance of XGB drops by more than 10%, and it is not related to the Morris parameter. The drop of RF is between 7% and 8%. Whether the performance drop of SVM and MLP exceeds the threshold is related to the parameter combination of the Morris method for generating important features. Not all parameter combinations have a performance drop of more than the threshold. Therefore, we increase the number of deleted features to 120. After deleting 120 features, the performance drop of SVM and MLP is about 9%.

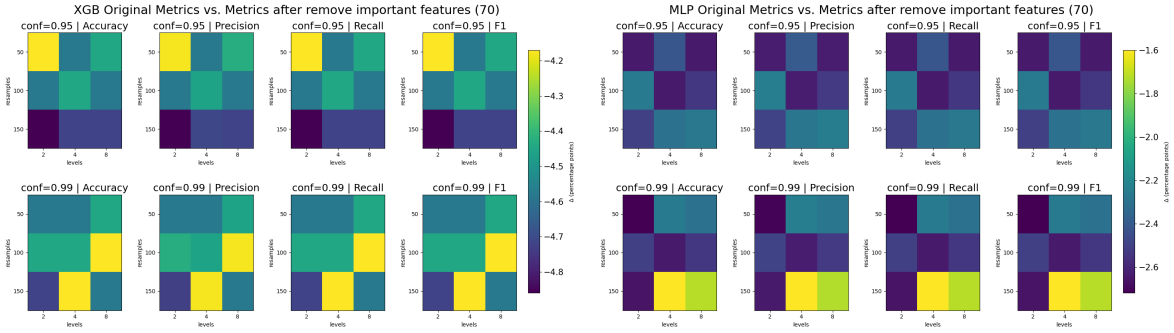
Then we verify the effectiveness of Morris, that is, we only use the 120 important features generated by Morris to verify the performance of the model. The results as shown in Table 7.22. From the results, we can see that when the model only uses 120 important features, the results of XGB are the same as those using the original features. SVM, RF and MLP are also close to the performance achieved by using the original features. Therefore, it can be shown that the important features generated by Morris are the basis for influencing

the model decision, so the explanation results generated by the Morris method are effective.



(a) SVM

(b) RF



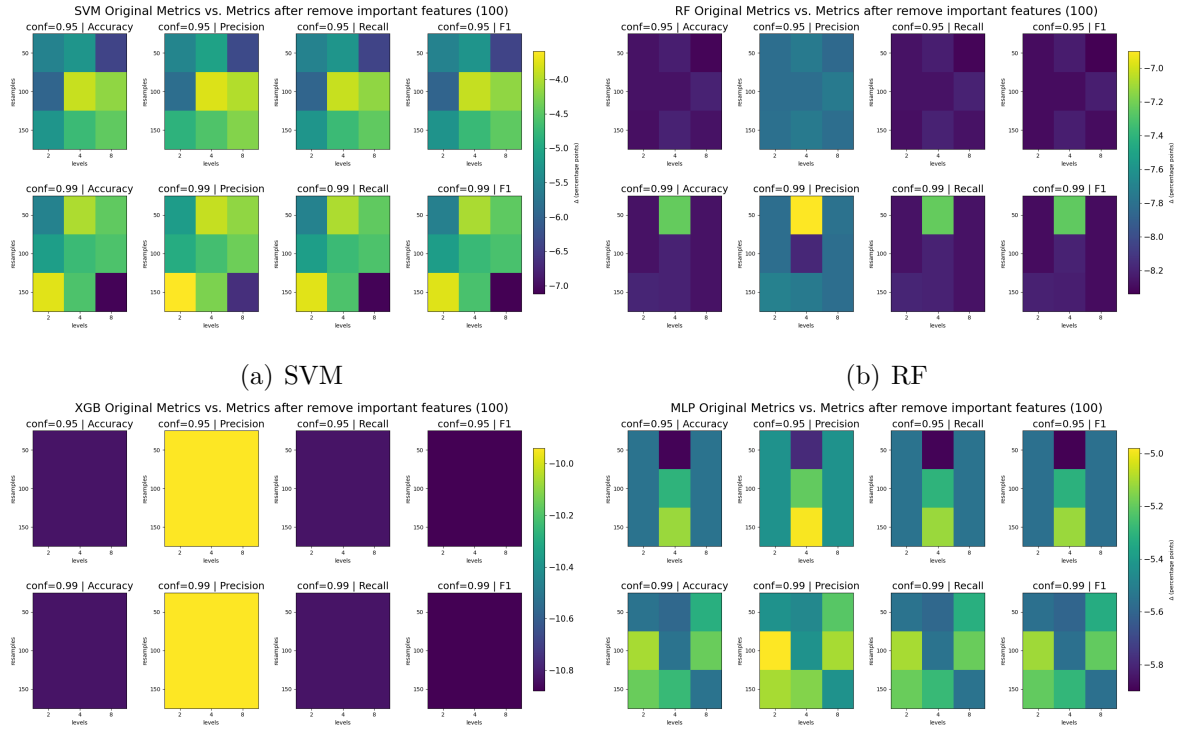
(c) XGBoost

(d) MLP

Figure 7.20: Effect of removing 70 important features identified by Morris on four performance metrics (accuracy, precision, recall and F1 score) of four black-box models at different combinations of conf_level, num_levels and num_resamples. The colored bars in each subplot represent the percentage point change of each metric before and after removal, darker color indicate larger changes, and lighter color indicate smaller changes.

	Orig (Acc, Prec, Recall, F1)	After Remove (Acc, Prec, Recall, F1)	Parameter
SVM	97.64%, 97.64%, 97.64%, 97.64%	97.50%, 97.50%, 97.50%, 97.36%	c=0.95, l=4, s=100
		97.36%, 97.36%, 97.36%, 97.36%	c=0.99, l=2, s=150
		97.36%, 97.36%, 97.36%, 97.36%	c=0.99, l=4, s=150
RF	98.46%, 98.46%, 98.46%, 98.46%	98.33%, 98.33%, 98.33%, 98.33%	c=0.95, l=4, s=100
		98.44%, 98.45%, 98.44%, 98.44%	c=0.99, l=2, s=150
		98.37%, 98.37%, 98.37%, 98.37%	c=0.99, l=4, s=150
XGB	98.89%, 98.90%, 98.89%, 98.89%	98.89%, 98.90%, 98.89%, 98.90%	c=0.95, l=4, s=100
		98.89%, 98.90%, 98.89%, 98.90%	c=0.99, l=2, s=150
		98.89%, 98.90%, 98.89%, 98.90%	c=0.99, l=4, s=150
MLP	98.52%, 98.53%, 98.52%, 98.52%	98.31%, 98.32%, 98.31%, 98.31%	c=0.95, l=4, s=100
		98.20%, 98.21%, 98.20%, 98.20%	c=0.99, l=2, s=150
		98.13%, 98.14%, 98.13%, 98.13%	c=0.99, l=4, s=150

Table 7.22: Performance results of four black-box models using only 100 features generated by Morris. C refers to the conf_level, l refers to the num_levels and s refers to num_resamples.



(a) SVM

(b) RF

(c) XGBoost

(d) MLP

Figure 7.21: Effect of removing 100 important features identified by Morris on four performance metrics (accuracy, precision, recall and F1 score) of four black-box models at different combinations of conf_level, num_levels and num_resamples. The colored bars in each subplot represent the percentage point change of each metric before and after removal, darker color indicate larger changes, and lighter color indicate smaller changes.

7.2.4 Discussion of Global Explanation Methods

Table 7.23 shows the stability (Stb), robustness (Rob) and effectiveness (Eff) performance of various global explanation methods on different models.
For stability, the ALE method has 100% stability on SVM and XGBoost, which means that the interpretation results do not change with multiple runs. It also has good stability, with 82.19% and 86.61% on RF and MLP, respectively. The PD method also has 100% stability on SVM and XGBoost, and 86.79% and 89.01% on RF and MLP, respectively, which is also good stability, slightly higher than ALE. Morris has the highest stability of 96.98% on XGBoost, followed by SVM and MLP, with 93.65% and 91.49%, respectively. However, Morris has a stability of only 71.79% on RF, which means that for RF, Morris generates significant differences in the results between multiple runs. PDV only achieves 100% stability on XGBoost, but has very low stability on SVM, RF, and MLP. This means that the interpretation results generated by the PDV method using these three models are almost different each time between multiple runs.

In terms of robustness, ALE performs well on XGBoost and SVM, with 98.83% and 98.68% respectively. RF is second with a robustness of 93.67%. This shows that the ALE method is insensitive to slight disturbances and has a certain anti-interference ability. The robustness of ALE on MLP is 85.50%, indicating that MLP is more sensitive when slight disturbances occur, so the generated explanation results will change with the disturbance of the eigenvalues. The robustness of the PD method exceeds 90% on all four models, with a robustness of 98.67% on SVM and 91.33% on RF. XGBoost and MLP have the same robustness of 91.17%. This shows that the PD method has low sensitivity to slight disturbances, and a small change in eigenvalues does not affect its explanation results. The robustness of the Morris method is best on RF and MLP, with 96.00% and 95.67% respectively, followed by XGBoost, which is 95.17%. The robustness on SVM is lower than that of other models, at 93.17%. However, from the overall results, the Morris method has stronger anti-interference ability than ALE and PD. The robustness values of the PDV method on the four models are very low, so this method is not robust, and a slight disturbance will cause a huge change in the interpretation results.

In terms of effectiveness, it can be seen that ALE, PD, and Morris have achieved full effectiveness on all four models, which means that the important features generated by these three explanation methods are indeed the features that affect the decision-making basis of the model, indicating that the results generated by these three explanation methods are meaningful and effective. However, the PDV method can only generate effective explanation results on XGBoost, but it is not effective for SVM, RF, and MLP. Therefore, combining the stability and robustness results, the PDV method is not suitable for Android malware detection because the method cannot generate reliable explanation results.

Figure 7.22 shows the average performance of the four global explanation methods on the four models. Regarding the stability of the explanation methods, PD >ALE >Morris >PDV; for robustness, Morris >PD >ALE >PDV. For effectiveness, although ALE, PD and Morris all reached 100%, according to previous experimental results, the ALE method only needs 100 features to make the model performance close to the original performance, while PD and Morris require 120. Therefore, in terms of effectiveness, ALE >PD = Morris >PDV. In

		SVM	RF	XGBoost	MLP
ALE	Stb	100%	82.19%	100%	86.61%
	Rob	98.68%	93.67%	98.83%	85.50%
	Eff	100%	100%	100%	100%
PD	Stb	100%	86.79%	100%	89.07%
	Rob	98.67%	91.33%	91.17%	91.17%
	Eff	100%	100%	100%	100%
PDV	Stb	21.67%	21.98%	100%	17.89%
	Rob	21.33%	22.67%	39.00%	21.00%
	Eff	0	0	100%	0
Morris	Stb	93.65%	71.79%	96.98%	91.49%
	Rob	93.17%	96.00%	95.17%	95.67%
	Eff	100%	100%	100%	100%

Table 7.23: Performance of the chosen global explanation methods.

addition, according to the previous experimental results, it can be seen that the parameters of the PD method have relatively little effect on the interpretation results. Overall, PD achieves the highest stability of 93.97%, Morris method has the highest robustness of 95% and ALE obtained the hightest effectiveness of 100%.

In Android malware detection, high stability means that the explanation method can output stably and provide similar explanation results when the detection model and input features do not change significantly, which is helpful for analysts to track some malicious behaviors in the long term. Android malicious applications usually confuse their true intentions through subtle changes or disguises. If the explanation method is too sensitive to small changes, it may result in completely different explanation results when the features change slightly, thus misleading the analysis. Therefore, high robustness indicates that the explanation method is insensitive to such changes and has a certain noise resistance to identify the important features that really affect the judgment of the detection model. Effectiveness focuses on whether the malicious behavior that the explanation method focuses on has a real impact on the model decision. Therefore, effectiveness can also be used to determine whether the detection method can be applied to the field of Android malware detection. High effectiveness demonstrates that the explanation method focuses on the key features that really affect the model judgment of whether it is malicious behavior, and through the evaluation of effectiveness, the number of most important features for the model decision can also be found, providing analysts with a more targeted explanation basis.

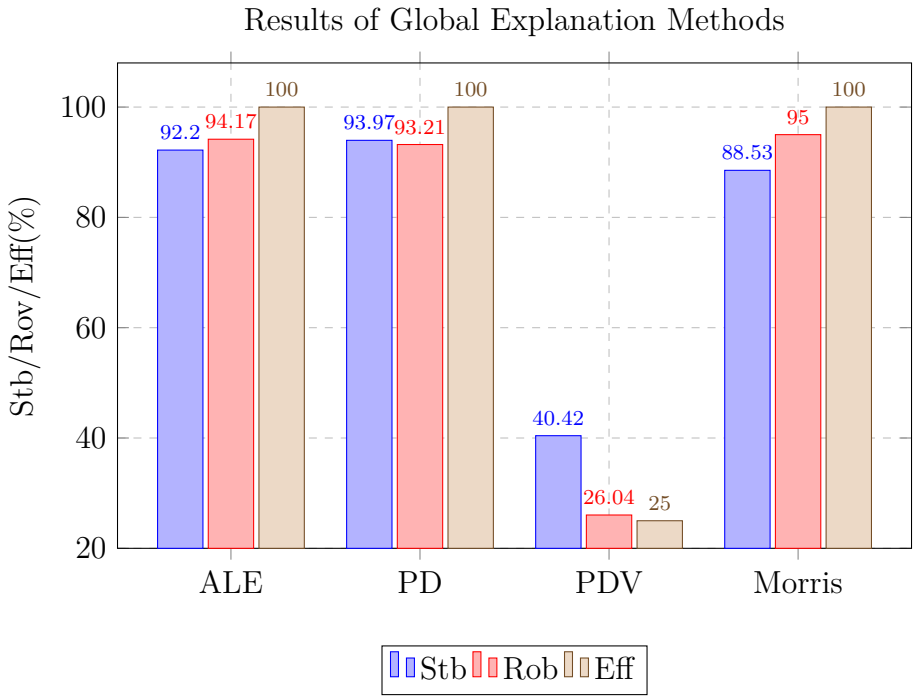


Figure 7.22: The average performance of the chosen Global explanation methods

8. Conclusion

This study aims to evaluate the quality of XAI methods for Android malware detection. To achieve this goal, we used four classification models with high accuracy but low interpretability, and evaluated 4 local explanation methods and 4 global explanation methods. The results we obtained can provide an empirical basis for evaluating the interpretability effects generated by different XAI methods in the Android malware detection task.

For local explanation methods, we extended the previous work of Soulaimani. Specifically, we improved some limitations raised in previous experiments. For example, we applied a larger dataset, improved the method of randomly selecting irrelevant features, and replaced some white-box machine learning explanation methods with black-box explanation methods, giving priority to those that can better explain complex model decisions. Local explanation methods target a single apk, and two evaluation metrics are based on the [78] consistency rate and soundness rate previously proposed by Soulaimani. The consistency rate effectively evaluates whether the method can generate relatively consistent explanation results in different models by comparing the consistency of the explanation feature sets generated between different models, reflecting the reliability of the explanation method. The soundness rate metric effectively evaluates whether the explanation method can ignore irrelevant features and accurately identify important features that really affect model decisions by comparing the consistency of the explanation feature set with the original result after adding the irrelevant feature set.
Based on the previous experimental results of Soulaimani and our experimental results, these evaluation metrics are applicable to feature-based local explanation methods. According to our experimental results, we demonstrate the impact of local explanation methods parameters on different tasks in Android malware detection. For example, the LIME method can achieve higher consistency rates using Manhattan as the distance parameter, while using Euclidean distance can achieve higher soundness rates, etc. Among these local explanation methods, SHAP performs the best in terms of soundness and LIME obtains the highest consistency rate. Furthermore, we found that explanation approaches based on generating counterfactuals are not applicable to the domain of Android malware detection because such methods cannot generate counterfactual instances and thus cannot provide interpretable results. The consistency rates of EDC and Anchors methods are low, only 28.33% and 33.25%. This shows that the malicious behaviors identified by the explanation methods on different detection models are quite different, which may confuse analysts with the changing interpretation results. In addition, the SR of Anchors is also lower than that of the other three explanation methods, at 83.64%. In practical applications, the key features identified are unstable due to their sensitivity to irrelevant features. Therefore, the EDC and Anchors methods may have certain limitations in actual detection, especially when using integrated models or in environments with more data noise. The reliability and practicality of their explanation results need to be further verified.

For global explanation methods, we developed three quality assessment metrics for explanation methods based on important features: stability, robustness, and effectiveness,

and evaluated them with four global explanation methods. Stability is used to evaluate the similarity between the explanation feature sets generated by the explanation method in multiple runs. Robustness is used to evaluate the sensitivity of the explanation method under slight perturbations of the input. Effectiveness is used to evaluate whether the explanation method can identify the features that are truly important for the model decision. Our metric is applicable to global explanation methods based on important features.
Experimental results show that PD achieves the best stability of 93.97%, Morris performs the best on robustness of 95%, and ALE has the highest effectiveness of 100%. In addition, we also prove that PDV is not suitable for Android malware detection. Besides, we also show that XGBoost not only leads SVM, RF, and MLP in classification performance, but also outperforms other models in interpretability evaluation and is less sensitive to parameter changes of each explanation method.

Our results cover both local and global explanation methods, which can provide more comprehensive and effective insights for malware analysts to choose appropriate explanation methods. We also focus on parameter tuning strategies to reveal the impact of explanation method parameters on explanation results, providing a reference for analysts to flexibly configure parameters according to specific task goals. For instance, the grid resolution parameter in the PD method has no effect on its stability on various black box models, but it effects the robustness of the method on different models. In addition, we also provide valuable suggestions for analysts on how to balance explanation quality and computational overhead in practice. For example, we found that the computational resources required by LORE in the local explanation method are much higher than those of LIME, SHAP and other methods. Although LORE has certain explanation capabilities, its operating efficiency is not practical when processing large-scale samples. The running time of PI and SHAP in the global explanation method is also much longer than that of other global explanation methods. Thus, it will bring a large computational burden in multi-model or multi-run explanation tasks. In the actual application deployment of Android malware detection, analysts need to select a lighter and higher-quality explanation method under the constraints of limited computing resources to balance efficiency and accuracy. We focus on the real key features so that analysts can quickly locate the decision-making basis of malicious behavior, thereby accelerating response efficiency and conducting more comprehensive and effective analysis.
In summary, this study not only provides a comprehensive, transparent, and operational framework for the selection of explanation methods in Android malware detection, but also provides operational, quantifiable evaluation criteria and practical paths for the application of XAI technology in the security field.

8.1. Limitations

In the experiments of local explanation methods, for the Anchors method, we only tested batch size, tau, and threshold parameters. In fact, the Anchors method also has other parameters, such as delta, beam size, etc. However, the parameter tuning of all Anchors consumes a lot of computing resources, so more attempts cannot be made. According to the current experimental results, the CR and SR results of Anchors are low, but we cannot

be sure whether the performance of Anchors will be improved after trying more parameter combinations. Another is the EDC method, which aims to understand the decision process of the document classifier. Although this method obtains a higher SR, the CR value is very low. Therefore, whether this method can be applied to the field of Android malware detection needs further verification.

The limitation of the global explanation method in the experiment lies in the experimental part of metric stability. In the experiment, we tested the stability of the four explanation methods in 20 runs, but failed to verify whether their stability would gradually decrease with the increase of the number of runs. A more cautious approach is to compare the experiments of multiple runs and gradually increase the number of runs (for example, from 20 to 200 or even more) to observe whether the stability changes significantly. However, due to hardware limitations, we were unable to run more comparison experiments, which could take several months. Second, our choice of the number of important features was based on some previous experience in selecting the number of features in experiments to evaluate local explanation methods. A more cautious approach would be to try different numbers of important features for validation to obtain more reliable results.

Another limitation is that in our experiments, all malware data comes from the AMD dataset and all benign data comes from AndroZoo. This means that the extracted important features are related to these two datasets, which affects the generalizability of the research results.

8.2. Future work

In future work, we can try to solve the problems mentioned in the limitations. In addition, we can evaluate the selected explanation methods and metrics on more malware datasets to obtain more comprehensive and reliable analysis and results. Moreover, neural networks, such as DNN, CNN, etc., have excellent performance in the field of Android malware, but some current XAI technologies do not support models built with TF/Keras. Therefore, in the future, the scope of application of some traditional XAI methods can be expanded to enable them to be applied to more neural networks, thereby providing equally reliable explainability support for more complex and less explainable deep learning, thereby further improving the coverage and credibility of Android malware detection systems. Furthermore, although there are many open source libraries for post-hoc XAI technologies, there are no libraries for XAI technologies that can be used in the direction of Android malware detection. Therefore, in future work, we can integrate XAI technologies that can be used in this field and develop an open source library for this field.

9. References

[1] Adel Abusitta, Miles Q Li, and Benjamin CM Fung. Survey on explainable ai: Techniques, challenges and open issues. *Expert Systems with Applications*, 255:124710, 2024.

[2] S Agatonovic-Kustrin and Rosemary Beresford. Basic concepts of artificial neural network (ann) modeling and its application in pharmaceutical research. *Journal of pharmaceutical and biomedical analysis*, 22(5):717–727, 2000.

[3] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. Androzoo: Collecting millions of android apps for the research community. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 468–471, New York, NY, USA, 2016. ACM.

[4] Ebtesam J Alqahtani, Rachid Zagrouba, and Abdullah Almuhaideb. A survey on android malware detection techniques using machine learning algorithms. In *2019 Sixth International Conference on Software Defined Systems (SDS)*, pages 110–117. IEEE, 2019.

[5] David Alvarez-Melis and Tommi S Jaakkola. On the robustness of interpretability methods. *arXiv preprint arXiv:1806.08049*, 2018.

[6] Namrata Govind Ambekar, N Nandini Devi, Surmila Thokchom, and Yogita. Tablstm-net: enhancing android malware classification through integrated attention and explainable ai. *Microsystem Technologies*, pages 1–19, 2024.

[7] Daniel W. Apley and Jingyu Zhu. Visualizing the effects of predictor variables in black box supervised learning models, 2019.

[8] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *Ndss*, volume 14, pages 23–26, 2014.

[9] Osvaldo Arreche, Tanish R Guntur, Jack W Roberts, and Mustafa Abdallah. E-xai: Evaluating black-box explainable ai frameworks for network intrusion detection. *IEEE Access*, 12:23954–23988, 2024.

[10] Alejandro Barredo Arrieta, Natalia Díaz Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. Explainable artificial intelligence (XAI): concepts, taxonomies, opportunities and challenges toward responsible AI. *CoRR*, abs/1910.10045, 2019.

[11] Ananya Aswathy, TR Amal, PG Swathy, MOHAMMAD SHOJAFAR, and P Vinod. Sysdroid: a dynamic ml-based android malware analyzer using system call traces. *Cluster computing*, 23(4):2789–2808, 2020.

[12] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.

[13] Elshan Baghirov. A comprehensive investigation into robust malware detection with explainable ai. *Cyber Security and Applications*, 3:100072, 2025.

[14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[15] Halit Bakır. A new method for tuning the cnn pre-trained models as a feature extractor for malware detection. *Pattern Analysis and Applications*, 28(1):26, 2025.

[16] Vivek Singh Bawa and Vinay Kumar. Linearized sigmoidal activation: A novel activation function with tractable non-linear characteristics to boost representation capability. *Expert Systems with Applications*, 120:346–356, 2019.

[17] Asa Ben-Hur, David Horn, Hava T Siegelmann, and Vladimir Vapnik. Support vector clustering. *Journal of machine learning research*, 2(Dec):125–137, 2001.

[18] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[19] Hamid Bostani and Veelasha Moonsamy. Evadedroid: A practical evasion attack on machine learning for black-box android malware detection. *Computers & Security*, 139:103676, 2024.

[20] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[21] Fabrizio Cara, Michele Scalas, Giorgio Giacinto, and Davide Maiorca. On the feasibility of adversarial sample creation using the android system api. *Information*, 11(9):433, 2020.

[22] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754, 2016.

[23] Xiao Chen, Chaoran Li, Derui Wang, Sheng Wen, Jun Zhang, Surya Nepal, Yang Xiang, and Kui Ren. Android hiv: A study of repackaging malware for evading machine-learning detection. *IEEE Transactions on Information Forensics and Security*, 15:987–1001, 2019.

[24] Molnar Christoph. *Interpretable machine learning: A guide for making black box models explainable*. Leanpub, 2020.

[25] Ambra Demontis, Marco Melis, Battista Biggio, Davide Maiorca, Daniel Arp, Konrad Rieck, Igino Corona, Giorgio Giacinto, and Fabio Roli. Yes, machine learning can be more secure! a case study on android malware detection. *IEEE transactions on dependable and secure computing*, 16(4):711–724, 2017.

[26] Adit Deshpande. A beginner's guide to understanding convolutional neural networks. *Retrieved March*, 31(2017), 2016.

[27] Android Developers. Application fundamentals. https://developer.android.com/guide/components/fundamentals. March, 2025.

[28] Amit Dhurandhar, Pin-Yu Chen, Ronny Luss, Chun-Chen Tu, Paishun Ting, Karthikeyan Shanmugam, and Payel Das. Explanations based on the missing: Towards contrastive explanations with pertinent negatives. *Advances in neural information processing systems*, 31, 2018.

[29] Konstantin Eckle and Johannes Schmidt-Hieber. A comparison of deep networks with relu activation function and linear spline-type methods. *Neural Networks*, 110:232–242, 2019.

[30] Ming Fan, Wenying Wei, Xiaofei Xie, Yang Liu, Xiaohong Guan, and Ting Liu. Can we trust your explanations? sanity checks for interpreters in android malware analysis. *IEEE Transactions on Information Forensics and Security*, 16:838–853, 2020.

[31] Jiayin Feng, Limin Shen, Zhen Chen, Yuying Wang, and Hui Li. A two-layer deep learning method for android malware detection using network traffic. *Ieee Access*, 8:125786–125796, 2020.

[32] Aaron Fisher, Cynthia Rudin, and Francesca Dominici. All models are wrong, but many are useful: Learning a variable's importance by studying an entire class of prediction models simultaneously. *Journal of Machine Learning Research*, 20(177):1–81, 2019.

[33] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[34] Antonio Galli, Valerio La Gatta, Vincenzo Moscato, Marco Postiglione, and Giancarlo Sperlì. Explainability in ai-based behavioral malware detection systems. *Computers & Security*, 141:103842, 2024.

[35] Daniel Gibert, Carles Mateu, and Jordi Planes. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications*, 153:102526, 2020.

[36] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[37] Brandon M Greenwell, Bradley C Boehmke, and Andrew J McCarthy. A simple and effective model-based variable importance measure. *arXiv preprint arXiv:1805.04755*, 2018.

[38] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Dino Pedreschi, Franco Turini, and Fosca Giannotti. Local rule-based explanations of black box decision systems. *arXiv preprint arXiv:1805.10820*, 2018.

[39] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM computing surveys (CSUR)*, 51(5):1–42, 2018.

[40] Wenbo Guo, Dongliang Mu, Jun Xu, Purui Su, Gang Wang, and Xinyu Xing. Lemna: Explaining deep learning based security applications. In *proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 364–379, 2018.

[41] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. The elements of statistical learning: data mining, inference, and prediction, 2017.

[42] Jon Herman and Will Usher. SALib: An open-source python library for sensitivity analysis. *The Journal of Open Source Software*, 2(9), jan 2017.

[43] S Hochreiter. Long short-term memory. *Neural Computation MIT-Press*, 1997.

[44] John H Holland. Genetic algorithms. *Scientific american*, 267(1):66–73, 1992.

[45] Takuya Iwanaga, William Usher, and Jonathan Herman. Toward SALib 2.0: Advancing the accessibility and interpretability of global sensitivity analyses. *Socio-Environmental Systems Modelling*, 4:18155, May 2022.

[46] E Venkata Pawan Kalyan, A Purushottam Adarsh, S Sai Likith Reddy, and Pn Renjith. Detection of malware using cnn. In *2022 second international conference on computer science, engineering and applications (ICCSEA)*, pages 1–6. IEEE, 2022.

[47] Emilie Kaufmann and Shivaram Kalyanakrishnan. Information complexity in bandit subset selection. In *Conference on Learning Theory*, pages 228–251. PMLR, 2013.

[48] Eoin M Kenny, Courtney Ford, Molly Quinn, and Mark T Keane. Explaining black-box classifiers using post-hoc explanations-by-example: The effect of explanations and error-rates in xai user studies. *Artificial Intelligence*, 294:103459, 2021.

[49] Janis Klaise, Arnaud Van Looveren, Giovanni Vacanti, and Alexandru Coca. Alibi explain: Algorithms for explaining machine learning models. *Journal of Machine Learning Research*, 22(181):1–7, 2021.

[50] Janis Klaise, Arnaud Van Looveren, Giovanni Vacanti, and Alexandru Coca. Alibi explain: Algorithms for explaining machine learning models. *The Journal of Machine Learning Research*, 22(1):8194–8200, 2021.

[51] Bojan Kolosnjaji, Ambra Demontis, Battista Biggio, Davide Maiorca, Giorgio Giacinto, Claudia Eckert, and Fabio Roli. Adversarial malware binaries: Evading deep learning for malware detection in executables. In *2018 26th European signal processing conference (EUSIPCO)*, pages 533–537. IEEE, 2018.

[52] Ron Korine and Danny Hendler. Daemon: dataset/platform-agnostic explainable malware classification using multi-stage feature mining. *IEEe Access*, 9:78382–78399, 2021.

[53] Vasileios Kouliaridis and Georgios Kambourakis. A comprehensive survey on machine learning techniques for android malware detection. *Information*, 12(5):185, 2021.

[54] Maithili Kulkarni and Mark Stamp. Xai and android malware models. *arXiv preprint arXiv:2411.16817*, 2024.

[55] Dan Li, Lichao Zhao, Qingfeng Cheng, Ning Lu, and Wenbo Shi. Opcode sequence analysis of android malware by a convolutional neural network. *Concurrency and Computation: Practice and Experience*, 32(18):e5308, 2020.

[56] Rui Li and Olga Gadyatskaya. Evaluating rule-based global xai malware detection methods. In *International Conference on Network and System Security*, pages 3–22. Springer, 2023.

[57] Kaijun Liu, Shengwei Xu, Guoai Xu, Miao Zhang, Dawei Sun, and Haifeng Liu. A review of android malware detection approaches based on machine learning. *IEEE access*, 8:124579–124607, 2020.

[58] Harshit Kumar Lohani, S Dhanalakshmi, and V Hemalatha. Performance analysis of extreme learning machine variants with varying intermediate nodes and different activation functions. In *Cognitive Informatics and Soft Computing: Proceeding of CISC 2017*, pages 613–623. Springer, 2019.

[59] Songhao Lou, Shaoyin Cheng, Jingjing Huang, and Fan Jiang. Tfdroid: Android malware detection by topics and sensitive data flows using machine learning techniques. In *2019 IEEE 2Nd international conference on information and computer technologies (ICICT)*, pages 30–36. IEEE, 2019.

[60] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

[61] David Martens and Foster Provost. Explaining data-driven document classifications. *MIS quarterly*, 38(1):73–100, 2014.

[62] Max D Morris. Factorial sampling plans for preliminary computational experiments. *Technometrics*, 33(2):161–174, 1991.

[63] Jakub Nalepa and Michal Kawulok. Selecting training sets for support vector machines: a review. *Artificial Intelligence Review*, 52(2):857–900, 2019.

[64] Modupe Odusami, Olusola Abayomi-Alli, Sanjay Misra, Olamilekan Shobayo, Robertas Damasevicius, and Rytis Maskeliunas. Android malware detection: A survey. In *Applied Informatics: First International Conference, ICAI 2018, Bogotá, Colombia, November 1-3, 2018, Proceedings 1*, pages 255–266. Springer, 2018.

[65] K O'Shea. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[66] Harshil Patel. Explained: Global, local and cohort explainability. Censiusblog. https://censius.ai/blogs/global-local-cohort-explainability. November, 2024.

[67] Xingzhi Qian, Xinran Zheng, Yiling He, Shuo Yang, and Lorenzo Cavallaro. Lamd: Context-driven android malware detection and classification with llms. *arXiv preprint arXiv:2502.13055*, 2025.

[68] Sharayu Rane. The balance: Accuracy vs. interpretability. Towards Data Science. https://towardsdatascience.com/the-balance-accuracy-vs-interpretability-1b3861408062. November, 2024.

[69] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[70] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[71] Janosh Riebesell and Stefan Bringuier. Collection of standalone tikz images, 2020. 10.5281/zenodo.7486911 - https://github.com/janosh/tikz.

[72] Lior Rokach and Oded Maimon. Decision trees. *Data mining and knowledge discovery handbook*, pages 165–192, 2005.

[73] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[74] Alvin E Roth. Introduction to the shapley value. *The Shapley value*, 1, 1988.

[75] Andrea Saltelli, Stefano Tarantola, Francesca Campolongo, Marco Ratto, et al. *Sensitivity analysis in practice: a guide to assessing scientific models*, volume 1. Wiley Online Library, 2004.

[76] Lloyd S Shapley et al. A value for n-person games. 1953.

[77] Santosh K Smmarwar, Govind P Gupta, and Sanjay Kumar. Xai-amd-dl: An explainable ai approach for android malware detection system using deep learning. In *2023 IEEE World Conference on Applied Intelligence and Computing (AIC)*, pages 423–428. IEEE, 2023.

[78] Oussama Souliman. Evaluating android malware detection explanation, 2019.

[79] Kimberly Tam, Ali Feizollah, Nor Badrul Anuar, Rosli Salleh, and Lorenzo Cavallaro. The evolution of android malware and android analysis techniques. *ACM Computing Surveys (CSUR)*, 49(4):1–41, 2017.

[80] Hind Taud and Jean-Franccois Mas. Multilayer perceptron (mlp). *Geomatic approaches for modeling land change scenarios*, pages 451–455, 2018.

[81] Suman R Tiwari and Ravi U Shukla. An android malware detection technique based on optimized permissions and api. In *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*, pages 258–263. IEEE, 2018.

[82] Arnaud Van Looveren and Janis Klaise. Interpretable counterfactual explanations guided by prototypes. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 650–665. Springer, 2021.

[83] Sandra Wachter, Brent Mittelstadt, and Chris Russell. Counterfactual explanations without opening the black box: Automated decisions and the gdpr. *Harv. JL & Tech.*, 31:841, 2017.

[84] Shanshan Wang, Zhenxiang Chen, Lei Zhang, Qiben Yan, Bo Yang, Lizhi Peng, and Zhongtian Jia. Trafficav: An effective and explainable detection of mobile malware behavior using network traffic. In *2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS)*, pages 1–6. IEEE, 2016.

[85] Weilun Wang, Goutam Chakraborty, and Basabi Chakraborty. Predicting the risk of chronic kidney disease (ckd) using machine learning algorithm. *Applied Sciences*, 11:202, 12 2020.

[86] Yingying Wang, Yibin Li, Yong Song, and Xuewen Rong. The influence of the activation function in a convolution neural network model of facial expression recognition. *Applied Sciences*, 10(5):1897, 2020.

[87] Alexander Warnecke, Daniel Arp, Christian Wressnegger, and Konrad Rieck. Evaluating explanation methods for deep learning in security. In *2020 IEEE european symposium on security and privacy (EuroS&P)*, pages 158–174. IEEE, 2020.

[88] Fengguo Wei, Yuping Li, Sankardas Roy, Xinming Ou, and Wu Zhou. Deep ground truth analysis of current android malware. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 14th International Conference, DIMVA 2017, Bonn, Germany, July 6-7, 2017, Proceedings 14*, pages 252–276. Springer, 2017.

[89] Bozhi Wu, Sen Chen, Cuiyun Gao, Lingling Fan, Yang Liu, Weiping Wen, and Michael R Lyu. Why an android app is classified as malware: Toward malware classification interpretation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 30(2):1–29, 2021.

[90] Feiyu Xu, Hans Uszkoreit, Yangzhou Du, Wei Fan, Dongyan Zhao, and Jun Zhu. Explainable ai: A brief survey on history, research areas, approaches and challenges. In *Natural language processing and Chinese computing: 8th cCF international conference, NLPCC 2019, dunhuang, China, October 9–14, 2019, proceedings, part II 8*, pages 563–574. Springer, 2019.

[91] Wenzhuo Yang, Hung Le, Silvio Savarese, and Steven Hoi. Omnixai: A library for explainable ai. 2022.

[92] Yubin Yang, Zongtao Wei, Yong Xu, Haiwu He, and Wei Wang. Droidward: an effective dynamic analysis method for vetting android applications. *Cluster Computing*, 21:265–275, 2018.

[93] Chih-Kuan Yeh, Cheng-Yu Hsieh, Arun Suggala, David I Inouye, and Pradeep K Ravikumar. On the (in) fidelity and sensitivity of explanations. *Advances in neural information processing systems*, 32, 2019.