Universiteit Leiden
The Netherlands

# Opleiding informatica

Evaluating Android malware detection explanation

Oussama Soulimani

Supervisors:
Olga Gadyatskaya & Rui Li

BACHELOR THESIS
Leiden Institute of Advanced Computer Science (LIACS)

# 1 Abstract

Machine learning-based methods have become de-facto standard in Android malware detection. However, it is not sufficient to be able to correctly classify a sample: human analysts also need some details (explanation) why this sample has been categorized as malicious or benign. Several black-box explanation methods have been proposed in the literature. Still, to be useful, explanation methods need to satisfy certain properties that are expected by the analysts in the Android malware detection domain.

Based on the existing works in this area, in this thesis we propose two new properties that a good Android malware explanation method must satisfy, and two metrics to measure these properties. We have conducted experiments on a dataset of Android apps, using 10 classifiers and evaluating 5 explanation methods: Morris Sensitivity, LIME, Anchors, EDC and SHAP. Our experiments show that SHAP is the best explanation method according to both of the proposed metrics.

# Contents

## 2 Introduction

With the rapid growth of Android users from five hundred million in 2012 to around 6 billion in the year 2021, the interest of different threat actors in compromising the Android security has risen. With this, more malicious Android applications are being discovered in the wild. Subsequently, the number of sophisticated Android malware application has also increased. For this reason, researchers devoted their time to develop new malware analysis techniques. One of the recent approaches to analyze Android malware is the usage of classification models. These models group Android application samples into a Malware class or a Benign class by training on a dataset and leveraging a wide range of algorithms used to classify future application samples. Modern approaches allow also the explanation of such classifications using *Explainable AI*.
Explainable AI is considered to be a response to the lack of transparency nature of AI models.This means that the output of AI models is increasingly difficult to understand and explain. AI has many advantages but it is often poorly able to explain its results on its own due to its complexity. Such approaches answer questions including: Why did the classification model take a certain decision? What features are responsible for that decision? The purpose of explainable AI is to make decisions and results understandable and explainable for the largest group as possible. A great example of how understanding results of AI can be helpful is the Dutch childcare benefits scandal [43]. In this scandal AI models were used that resulted in "unlawful, discriminatory and improper" working methods. This could be prevented in an early stage if the decisions made by the AI models could be explained. Different stakeholder can benefit immensely from researchers by knowing the reasons behind the classification. Researchers can increase their understanding of the models and produce alternative models that performs better. Regulators can increase their trust and be more transparent about their decision. The consumer can also benefit from Explainable AI by getting access to reports and analyses of such models to understand the impact of the model and increase their trust in it.
Focusing more on Android applications, to analyse and detect malicious Android applications, some features about the application located in the Android manifest and the Dalvik executable files need to be extracted. A specific model is used to classify the Android application (Malware/Benign) based on the extracted features. Using an explanation approach, the responsible features for the classification can be identified. Figure 2 below illustrates the whole process, from collecting data to finally evaluating the explanation approach. In the light of the aforementioned explanation approaches, some important questions should be asked. How can we know that an explanation approach can be trusted? What are the limitations of these explanations? We hereby propose two evaluation metrics that provide a quantitative measure of the performance of different explanation approaches.

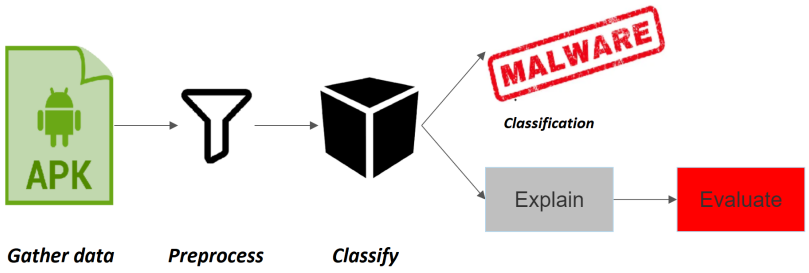


Figure 2: Evaluation process

# 3 Background

## 3.1 Classification algorithms

Some of the mentioned explanation approaches are black-box explanation. That is, they are able to simulate a given classification model which makes them able to explain any given classification model. A classification model groups objects into a set of categories in order to train on datasets and leverage a wide range of algorithms and classify future objects. In order to perform experiments on the explanation approach (section 6.1 and 6.2) different classifiers are being used. This section gives a short explanation on how the different classification model function.

### 3.1.1 Decision Tree:

A decisiom tree [35] is an undirected, acyclic and connected graph. The set of nodes can be divided into three categories:

- Root nodes: Access to the tree starts from this node.
- Internal nodes: Nodes that have descendants which are in turn nodes.
- Leaf nodes: last nodes of the tree that do not have any descendants.

Decision trees are a category of trees. They use a hierarchical representation of the data structure in the form of sequences of decisions to finally predict the class. Each given record which must be assigned to a class is described by a set of features which are tested in the nodes of the tree. Testing is done in the internal nodes and decisions are made in the leaf nodes.
The robustness of decision trees is relative to other other tree based models not very good [8] The variance of class is relatively high. However, it is a good building block for better performing classification models below.

### 3.1.2 Random Forest:

Random Forest was proposed by Leo Breiman in [6]. Random forest is an ensemble learning method for classification and regression, where different models

are combined to form a specific prediction. This classification model is an alternative to the basic decision tree classifier. The issue with decision trees is that it can result to overfitting on the training data. By changing a record on the dataset, it is possible that the model changes the decision tree completely because decision trees are in general highly sensitive to changes within the training data.
The random forest classifier overcomes this issue by using multiple trees. This classification model is based on bootstrap aggregation which creates a random sample with replacement from the dataset. Some records may occur more than once in each sample. Then, a decision tree is trained on each sample independently. For each random sample, a random subset of features is generated to be used for training a specific decision tree. The number of subsets of the original dataset (i.e. the number of decision trees) is specified beforehand.
At this point a number of decision trees are trained and each of them generate a specific classification given a specific record. As shown in figure 3, the results of the decision trees are aggregated using the majority vote and used as the final classification of the specific record.

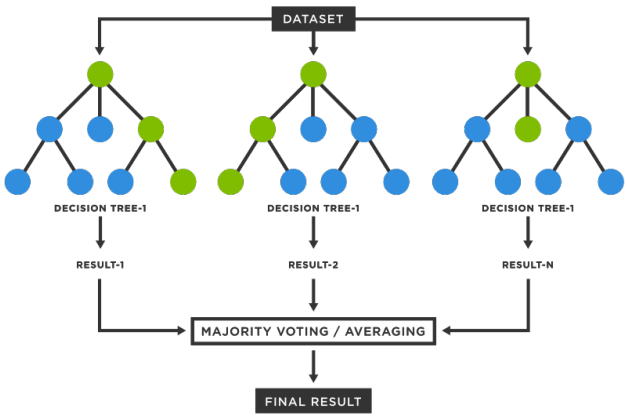


Figure 3: Random Forest Classifier https://www.tibco.com/sites/tibco/files/media_entity/2021-05/random-forest-diagram.svg

### 3.1.3 AdaBoost:

AdaBoost was proposed by Freund and Schapire in [15] AdaBoost is an ensemble learning method which calls a given weak or base learning algorithm repeatedly in an iterative manner. On each iteration a different model is being generated and the bias is being decreased. In order to achieve this, a decision stump is created for each feature. Then the Gini Index will be calculated for each decision stump. The sample that results in a bad split (i.e. the tree with

the lowest Gini Index) is given priority by increasing their weight and is being forwarded as input to the following model. This process is repeated until the specified number of trees is reached.

#### 3.1.4 Support vector machine (SVM):

Support vector machine was proposed by Siegelmann and Vapnik in [4]. The support vector machine is a binary classifier based on finding the optimal hyperplane. This hyperplane ultimately divides the dataset into 2 classes. In one-dimensional space, the hyperplane is a point, in two-dimensional space, the hyperplane is a line and in three-dimensional space, the hyperplane is a plane that divides the the record into two parts. The closest data points to the hyperplane (the support vector) determine the position of the hyperplane. If the support vectors are removed, the hyperplane will also change.
In order to create the best possible classification, the margin (i.e. distance between the two support vectors) must be maximized. In practice, datasets are more complex and the hyperplane cannot be easily determined. To bypass this problem, the dataset is transformed to be a three-dimensional space using a kernel function that allows the use of three-dimensional data instead of a two dimensional. Different types of kernels can be used such as polynomial kernels, radial basis function kernels, neural network Gaussian process and much more. From this perspective it can be easier to determine the hyperplane because some points might be deeper than other data points. The higher the dimension, the easier the classification. This is called "the kernel trick".

#### 3.1.5 Multi-layer Perceptron (MLP):

Multi-layer Perceptron was proposed by Frank Rosenblatt in [36]. The multi-layer perceptron is a type of neural network organized in multiple layers in which information flows from the input layer to the output layer. This is called a *feed-forward* network. Each layers consists of a number of neurons. The neurons of the last the last layer (i.e. the output) produces the target classification. The first version, the perceptron was a single-layer network and had a single output to which all the perceptrons were connected. It was incapable of solving nonlinear problems. This limitation was overcome by a supervised learning technique called backpropagation.
Neurons in one layer of the multi-layer backpropagation perceptron are connected to all neurons in adjacent layers.
A coefficient that affects these connections changes the effect of the information on the destination neurons. Consequently, the weight of each connection is the most important part of the network. Creating a multi-layer perceptron includes determining the optimal weights which are applicable in every interneuronal connection to solve a classification problem. This determination is performed using a backpropagation algorithm through a method called chain rule.

#### 3.1.6 K-Nearest Neighbors:

K-Nearest Neighbors was proposed by Evelyn Fix and J. L. Hodges, Jr in [13]. k-nearest neighbors is a non-parametric classification/regression method. The input consists of k closest training samples in the dataset. A record is classified using multiple votes of its neighbors, where the object is placed in the most occurring class among its k neighbors. If k=1, the record gets the same classification as its nearest neighbor.
The training records are vectors in a multidimensional space where each training records includes its classification target. In the classification, k is a predefined constant by the user and the unlabeled vector in question is classified based on the most common label among its k nearest neighbors. For continuous data, the Euclidean distance [10] can be used. Alternatively, for discrete variables such as bitstrings, the Hamming distance [5] can be used. To improve the accuracy of the classification, the distance between the data records can be normalized in case the features have different measuring units.
Another aspect that needs to be taken into consideration is the possibility of the class distribution being skewed. In this case, a new record can be dominated by the most occurring class which have a higher probability of occurring in the k nearest neighbors. To overcome this issue, the classification of record can be done while taking the distance between the new record and the k nearest neighbors into account.

#### 3.1.7 Gaussian Naive Bayes:

Naive Bayes [23] classifiers are a family of probabilistic classification algorithms based on the Bayes theorem. This classifier assumes strong independence between the features i.e. it assumes that the existence of a feature for a specific class is independent of the existence of other features. Gaussian Naive Bayes is an extension of Naive Bayes that calculates the mean and standard deviation for the training data. In this case it specifically assumes normal/Gaussian distribution.
The advantage of the Naive Bayes classifier is that it requires relatively small training dataset to estimate the standard deviation and the variance.

#### 3.1.8 Gradient Boosting:

The basic idea of gradient boosting [16] is similar to bagging. Rather than using a single model, several ones are used that are then aggregated to produce a single result. To build the models, boosting works sequentially. It starts by building a first model that is evaluated. From this evaluation, each model is weighted according to the performance of the prediction. The objective is to give a greater weight to models for which the value was poorly predicted for the construction of the following model. Correcting the weights over time makes it easier to predict difficult values. Gradient boosting uses the gradient of the loss function to calculate the weights of the individual models during the construction of each new model. Gradient boosting machines use in general

classification and regression trees (CART). The algorithm is customisable as it is possible to use different parameters and different functions.

#### 3.1.9 Gaussian process:

For most machine learning models an input is provided and a single classification is produced. Gaussian processes [37] instead produce a full distribution over classes to reflect the uncertainty in the prediction. Suppose a two dimensional dataset is provided. The goal is to produce a predictive distribution given a specific record. This means that for a given record, there a certain probability that class of that record lays between two classifications. In order to produce this predictive distribution, linear regression can be used. Subsequently, the remaining noise variance can be estimated. However, in this case, the uncertainty on linear model itself is missing. This can be solved using *prior sampling*.
First, samples of functions need to be produced before factoring the data. In the case of Bayesian linear classification, the samples are random lines/planes in the feature space. These lines/planes represent the prior samples. Now Bayes theorem [17] can be used to update the model such that is produces samples that agree with the data. These samples are called *posterior samples*. To get the predictive distribution, noise variance is calculated for each sample which is averaged over multiple (infinite) samples.
Gaussian processes use non linear samples instead of linear samples (in certain cases linear function may also be used). In general, these models are smooth functions.

### 3.2 Explanation methods

This research will mainly fixate on the following five explainers: Anchors, EDC, LIME, MSE and SHAP. The current section will help demystify the mentioned explainers.

#### 3.2.1 Local Interpretable Model-agnostic Explanations (LIME)

The local Interpretable Model-agnostic explanation method is proposed by Ribeiro et al. [32]. It provides explanations for the predictions of any machine learning classifier in an interpretable and faithful manner, by learning an interpretable model locally around the prediction. With locality in this context, local explainability is meant which identifies what features of a specific instance were responsible for the models' output. As mentioned before, LIME stands for Local Interpretable Model-agnostic Explanations. This shows that it is based on three basic properties which characterizes this model explainer.

**Local fidelity:**

Local fidelity describes how closely does the explanation reflect the outcome of a black-box model. High fidelity is a crucial property of an explanation because a low fidelity explanation is incapable of providing the right explanation for the

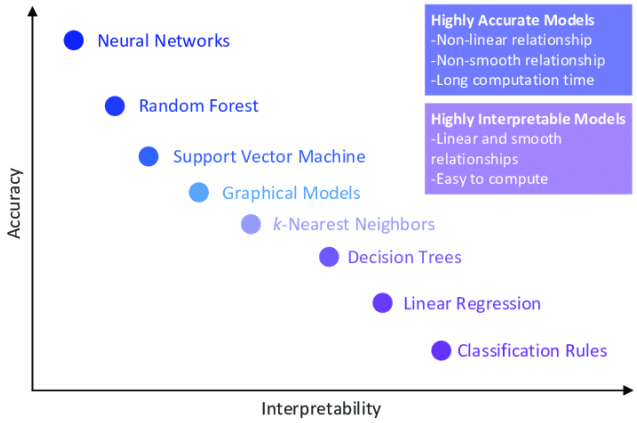


Figure 4: The trade-off between interpretability and accuracy of some relevant ML models. Highly interpretable algorithms such as classification rules, or linear regression, are often inaccurate. Very accurate DNNs are a classic example of black boxes. [30]

machine learning model. A few explanation approaches work well only on a subset of the data or even on a single data instance (i.e. local fidelity). An example of explanations that adhere to the property of local fidelity is Shapley values which originates from the cooperative game theory [11]. Shapley values assign a unique distribution among the players of a total surplus generated by the coalition of all players.

**Interpretability:**

Interpretability is the degree to which the user is able to accurately and efficiently predict the method's results i.e. how well a human can predict the group it belongs to (accuracy) and how fast they can perform the task (efficiency). [20]. Returning to the basic concept of classification, the goal is to achieve the highest prediction accuracy as possible, while preserving this interpretability property. Because of the nature of the both aspects, it is simply not possible to have a fully accurate model that is highly interpretable. Consider the one of the simplest models linear regression. This models is only capable to generate linear functions which are considered restrictive and generally inaccurate. Despite its restrictive property, this model is highly interpretable in the sense that it is possible to reverse the relationship between both the response variable and the explanatory variables i.e. interpreting the model. However, interpreting the model becomes more difficult when using less restrictive models such as polynomial models, non linear models and even more difficult with non parametric models. Figure 4 shows a scatter plots that demonstrates the trade off between interpretability and accuracy of a model.

**Model-agnostic:**

A model-agnostic explanation is an explanation that is not tied to a specific model and can thus be applied to any model. This can be achieved by learning an interpretable model [3] which essentially restricts the usable models. This can also be achieved by perturbing the inputs and measuring the variability of the output [39]. This aims at extracting post-hoc explanations and treating the original model as black-box which does not require any knowledge about the mechanics of the model [33].

LIME explanations are produced by the following formula:

$$\xi(x) = argmin_{g \in G} \mathcal{L}(f, g, \pi_x) + \Omega(g)$$

where
$G$: the class of a potentially interpretable models.
$\Omega(g)$: Complexity of the explanation g in G.
$f(x)$: Model to be explained. In our scope it is the probability that x belongs to a class.
$\pi_x(z)$: To define locality around x, $\pi_x$ is the distance between z and x.
$\mathcal{L}$: Degree of faithfulness of g in approximating f with respect to the locality around x ($\pi_x$)

### 3.2.2 Anchors

This explanation method is a rule-based variant of the LIME explainer and is also proposed by Ribeiro et al. [34]. Rule-based explanation produce rules like: if features x has a certain value and feature y has a value greater than another value, then the instance has a certain classification. Anchors is also model-agnostic which means that is can be useful when it operates on interpretable models as well as less interpretable models as mentioned in section 3.2.1.
Anchors and LIME focus on a single instance which is more efficient in contrary to interpreting the model globally which is considered to be complex and simply unfeasible. Anchor is a rule based explainer, which means that it presents certain conditions in a way that if they hold, the prediction of the model on a specific instance can be explained by those specific conditions with high accuracy. The used rules are if-then rules which are called Anchors. The anchors are formally defined by Ribeiro et al. [34] as follows:

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

where
x: Instance
f: Black box model
A: Rule that consists of set of predicated
Dx: Perturbation distribution on instance x
$D(.|A)$: Conditional distribution when A is applicable

$D(z|A)$: Conditional distribution on sample z when A is applicable
$\tau$: Preselected degree of precision

The Anchors explainer takes an incremental bottom-up approach to construct the individual anchors. Firstly an empty rule $\mathcal{A}$ is initialized, then on each iteration a candidate rule is generated and added to $\mathcal{A}$. We end up with the following rule set $\mathcal{A} = \{A \wedge a_i \wedge a_{i+1} \wedge a_{i+2} \wedge ... \wedge a_n\}$. A is then replaced by the candidate rule with the highest estimated precision. This precision is calculated using a pure exploration multi-armed bandit [19] where A represents the arm and the true precision of A on $D(.|A)$ represents the latent reward [22]. Each action in the environment represents checking whether a sample z from $D(z|A)$ can be predicted as true. The iteration terminates on a certain rule candidate if the following anchor definition holds:

$$P(prec(A) \geq \tau) \geq 1 - \delta$$

with:

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

It is a recurrent scenario that multiple anchors satisfy this requirement. In this case, the anchor that covers the largest part of the input space should be selected.

### 3.2.3 Morris Sensitivity Explanation (MSE):

The Morris Sensitivity Explanation utilizes a method proposed by Max D. Morris in [31] currently called the Morris method. The Morris method is suitable for models whose input features and output features are quantitative. It is part of the OAT (One At a Time) methods, meaning a process of exploration of the domain of definition by varying the factors one step at a time. In the case of a model that is costly in execution time, or a model with a large number of features, the method is a simple way of making an initial sorting among the factors according to their influence.
Consider a model consisting of n features whose output is denoted Y = Y ($X_1$, ..., $X_K$). The exploration of Y is performed on a grid overlayed on the domain of definition of the model. "Side" effects and variations of Y are calculated by the trajectories defined on the nodes of the grid by changing each feature on each trajectory only once.
Let $d_{i,j}(Y)$ be the variation of Y on a trajectory i with $i < r$ relative to the variation of the feature $X_j$. The statistic $\mu_j$ is defined as

$$\mu_j = \frac{1}{r} \sum_{i=1}^{r} d_{i,j}$$

In order to guarantee a uniform sampling of the different levels of each factor, it is advised to construct a grid with an even and identical number of levels. The desirable variation step of a factor is half the number of levels.

### 3.2.4 Shapley Additive Explanation (SHAP)

This explanation approach proposed by Lunderberg Lee [25] is based on feature importance estimations (i.e. SHAP values). SHAP is defined as an additive feature attribution method, which means that it has a linear function model of binary variables: $g(z') = \phi_0 + \sum_{i=1}^{M} \phi_i z'_i$, with $z' \in \{0,1\}^M$ and $\phi_0 \in \mathbb{R}$ Lunderberg Lee defined three desirable properties of Shapley value estimation methods which are:

**Local accuracy:**

Describes the match between the explanation model $g(x')$ and the original model $f(x)$. Following the local accuracy property, it can be said that there is a match if $x = h_x(x')$. This expression can be formulated as follows:

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

Where
$h_x$: Maps simplified inputs to the original input space
$f$: Prediction model
$\phi_0$: The model output when all simplified inputs are missing (i.e. $f(h_x(0))$)
M: Number of simplified input features

**Missingness:**

This property suggests that if there is a missing feature from the original input, it should not have an effect on the explaining features.

$$x'_i \Rightarrow \phi_i = 0$$

**Consistency:**

This property says that if the model changes so that the contribution of a specific feature value increases or stays the same, the Shapley value also increases or stays the same.
Let $f_x(z') = f(h_x(z'))z' \setminus i$ denotes that $z'_i = 0$.

$$\forall f, f' \text{ and } z' \in \{0,1\}^M$$
$$f'_x(z') - f'_x(z' \setminus i) \geq f'\_x(z') - f_x(z' \setminus i) \Longrightarrow \phi_i(f', x) \geq \phi_i(f, x)$$

Shap uses Shapley sampling methods [40] and Kernel SHAP to approximate the SHAP values. The Kernel SHAP works as follows. Different permutation samples are passed to the model while excluding certain feature(s) each time. For each sample, the average model output for the newly created instances is calculated. Subsequently, a weighted linear regression can be formulated based on the averages over all the samples. This forms the basis of the Kernel Shap. Besides Kernel Shap, there are more model-specific approximations such as Linear SHAP, Low-order SHAP and Max-SHAP. Nevertheless, the focus for this research will be on Kernel SHAP due to its applicability on every model.

### 3.2.5 Explaining Document Classification (EDC)

EDC was proposed by Martens et al. in [29] This explanation approach is used mainly for explaining classifications of text documents based on words and phrases. These classifications perform sentiment analysis, spam identification, web page classification, and document classification for topical web search. Typically, this explainer is used on instances with high dimensionality.
EDC constructs the explanation as follows:
Given a document vocabulary with m words, let the mask vector $\mu$ be a binary vector of length m with each element in $\mu$ is a word in the vocabulary. An explanation is a mask vector $\mu_E$ with $\mu_E(i) = 1 \iff w_i \in E$ and $\mu_E(i) = 0 \iff w_i \notin E$. D\E is the Hadamard product of the feature vector of document D with the 1's complement of $\mu_E$. Thus, the goal is to find $\mu_E$ such that $C_M(D\backslash E) \neq C_M(D)$, if any bit of $\mu_E$ is zero form E' such that $C_M(D\backslash E') \neq C_M(D)$.
Martens et al. proposed also metrics that measure the performance of an algorithm that searches for an explanation that matches the aforementioned formula. Such algorithm may do the following:

- Find a minimal explanation such that no other explanation of smaller size exists.
- Find all minimal explanations.
- Find all explanations of size smaller than a given k.
- Find l explanations, as quickly as possible (l = 1 may be a common objective).
- Find as many explanations as possible within a fixed time period.

Concerning the metrics, Martens proposed the following:

- Search effectiveness: Defined by the percentage of test instances explained (PE)
- Explanation complexity: Defined by the average number of words in the smallest explanation (AWS)
- Problem complexity: Defined by two metrics namely the average number of smallest explanations given (ANS) and average number of total explanations given (ANT)
- Computational complexity: Defined by two metrics: the average duration to find first explanation (ADF) and the average duration to find all explanations (ADA)

Martens et al. proposed in their paper an algorithm shown in figure 5 that uses Best-first search with pruning to find an algorithm that searches for an explanation that matches the aforementioned formula.

**Inputs**:
$W_D = \{w_i, i = 1, 2, \ldots, m_D\}$ % Document $D$ to classify, with $m_D$ words
$C_M : \mathcal{D} \rightarrow \{1, 2, \ldots, k\}$ % Trained classifier $C_M$ with scoring function $f_{C_M}$
$max\_iteration = 30$ % Maximum number of iterations
**Output**:
Explanatory list of rule R

$c = C_M(D)$ % The class predicted by the trained classifier
$p = f_{C_M}(D)$ % Corresponding probability or score
$R = \{\}$ % The explanatory list that is gradually constructed
$combinations\_to\_expand\_on = \{\}$
$P\_combinations\_to\_expand\_on = \{\}$
**for** $i = 1 \rightarrow m_D$ **do**
  $c_{new} = C_M(D \setminus w_i)$ % The class predicted by the trained classifier if word $w_i$ did not appear in the document
  $p_{new} = f_{C_M}(D \setminus w_i)$ % The probability or score predicted by the trained classifier if word $w_i$ did not appear in the document
  **if** $c_{new} \neq c$ **then**
    $R = R \cup$ '*if* word $w_i$ is removed *then* class changes'
  **else**
    $combinations\_to\_expand\_on = combinations\_to\_expand\_on \cup w_i$
    $P\_combinations\_to\_expand\_on = P\_combinations\_to\_expand\_on \cup p_{new}$
  **end if**
**end for**
**for** $iteration = 1 \rightarrow max\_iteration$ **do**
  $combo$ = word combination in $combinations\_to\_expand\_on$ for which $(p - p\_combinations\_to\_expand\_on)$ is maximal % The best first
  $combo\_set$ = create all expansions of $combo$ with one word
  $combo\_set2$ = remove combinations containing already found explanations of $R$ from $combo\_set$ % The pruning step
  **for** all combos $C_o$ in $combo\_set2$ **do**
    $c_{new} = C_M(D \setminus C_o)$ % The class predicted by the trained classifier if the words in $C_o$ did not appear in the document
    $p_{new} = f_{C_M}(D \setminus C_o)$ % The probability or score predicted by the trained classifier if the words in $C_o$ did not appear in the document
    **if** $c_{new} \neq c$ **then**
      $R = R \cup$ '*if* words $C_o$ are removed *then* class changes'
    **else**
      $combinations\_to\_expand\_on = combinations\_to\_expand\_on \cup C_o$
      $P\_combinations\_to\_expand\_on = P\_combinations\_to\_expand\_on \cup p_{new}$
    **end if**
  **end for**
**end for**

Figure 5: Search algorithm for Explanations for Document Classification [29]

# 4 Datasets

In order to perform some experiments using the aforementioned evaluation metrics (6.2) of different explanation approaches, we used a dataset that was made publicly available for researchers. It concerns *Investigation of the Android Malware (CIC-InvesAndMal2019)* dataset [41]. This dataset was also used in the paper Extensible Android Malware Detection and Family Classification Using Network-Flows and API-Calls [42].

## 4.1 Malware families:

This dataset consists of four categories: Adware, Ransomware, Scareware and SMS Malware. The Adware and Ransomware categories contain each 10 malware families. The Scareware and SMS Malware each consist of 11 malware families.
Table 1 describes the behavior of all collected malware families. This information is used to understand, interpret and potentially confirm the accuracy of

explanations. Due to the lack of a description for some malware families, the behavior cell of those malware families is left empty.

The aforementioned dataset contains only Malware sample. In order to classify Android application into Malware and benign, another dataset was additionally used. It concerns the CICMalDroid 2020 dataset [27] which was used in the papers [26] and [28]. This dataset contains different malware and benign Android applications. Only the subset of benign Android applications was used from this dataset. It contains over 1700 application samples from which 600 applications were used due to hardware limitations which are discussed in section 10. From the CIC-InvesAndMal2019, 426 Android malware samples were used.

| Category | Families | Behavior |
|---|---|---|
| Adware | Dowgin | Display advertising content and gather data |
| | Ewind | Show ads and inserts ads in browsers |
| | Feiwo | Gather phone number, IMEI and list of apps. |
| | Gooligan | Steal Gmail credentials, install apps and rate them, install adware |
| | Kemoge | Access camera, hardware details, phone state, network connection... |
| | koodous | – |
| | Mobidash | Display aggressive advertisements. |
| | Selfmite | Spread through SMS and display adds |
| | Shuanet | Root devices and obtainsystem-level persistence. |
| | Youmi | Gather GPS and cell tower location, IMEI, phone number |
| Ransomware | Charger | Copy data from agenda, messages, etc., and obtain root permission |
| | Jisut | Delete user data |
| | Koler | Display fake notice and demand payment |
| | LockerPin | Steal PINs, locks Android devices |
| | Simplocker | Collect sensitive information and encrypt files |
| | Pletor | Encrypt files with AES and decrypt after receiving certain message |
| | PornDroid | Download LockerPin and lock user out. |
| | RansomBO | – |
| | Svpeng | Obtain root permissions, intercept paid services |
| | WannaLocker | Encrypt files of external storage |
| Scareware | AndroidDefender | Urge users to buy antivirus |
| | AndroidSpy | Infect Windows machines linked to the device |
| | AV | – |
| | AVpass | – |
| | FakeApp | Pretends to be a legitimate app or to be an app that solves a problem |
| | FakeApp.AL | – |
| | FakeAV | Simulate antivirus software or parts of the operating system security |
| | FakeJobOffer | Display image of a (fraudulent) job offer |
| | FakeTaoBao | Pretends to be legitimate Taobao app |
| | Penetho | Pretends to generate password for a WiFi router |
| | VirusShield | Pretends to be an Android antivirus app |
| SMS Malware | BeanBot | Send premium-rate SMS messages from the infected device |
| | Biige | – |
| | FakeInst | pretend to be app installer, send premium-rate SMS |
| | FakeMart | Pretend to be black market app and SMS messages to short numbers |
| | FakeNotify | Send SMS messages to premium-rate numbers, redirect to website |
| | Jifake | Pretends to be messaging app, sends SMS messages to premium-rate numbers |
| | Mazarbot | Launch MITM, take control over Android functionalities |
| | Nandrobox | Send premium-rate messages and delete response |
| | Plankton | Forward information about the device to a remote location |
| | SMSsniffer | – |
| | Zsone | Send SMS messages to premium-rate numbers related for subscription |

Table 1: Behavior of malware families per category contained in the dataset

## 4.2 APK file components:

Focusing on an individual Android application, there are mainly seven components [2] of which an Android application consists:

- **META-INF:** directory which contains information about the files of the APK file
- **CERT.SF:** Contains a list of resources and a of the corresponding lines in the MANIFEST.MF file
- **AndroidManifest.xml:** An additional Android manifest file, describing the name, version, access rights, referenced library files for the application. (Elababorated more in the following paragraph)
- The certificate of the application
- **lib:** Directory with compiled native libraries used by your app. Contains multiple directories, one for each supported CPU architecture
- **resources.arsc:** File containing precompiled resources, such as binary XML for example
- **classes.dex:** The classes compiled in the dex file format executed by Android Runtime

Our focus lays mainly on two file types that allow simple static analysis, namely the Android Manifest file and the Dalvik Executable File.

**Android Manifest file:**
Contains all components of the application such as activities, services, broadcast receivers and content providers. This file also contains permissions the app needs in order to perform specific actions such as accessing the memory or the camera. Moreover, this file contains device compatibility information such as the minimum hardware and software requirements to run the application.

**Dalvik Executable File:**
Android applications are commonly written in Java. In most cases, Java programs run on the Java Virtual Machine (JVM) which requires a lot of recources and is thus not efficient. That is why an Android optimised virtual machine is used or the Dalvik VM. The application is built and compiled to a JVM bytecode and converted to a Dalvik Executable file (classes.dex).
For this research, we only focussed on the Android Manifest file because of the complexity and the amount of the Dalvik Executable files.
The malware and benign apps of the two dataset should also be unpacked. In order to do this, a Python script was created (unpack.py). This script crawls through the dataset files and extracts each APK file that it finds. Because of the huge amount of APK files, this is done in a multi-threaded manner. For the actual APK extraction, APKtool [9] was used. It is a tool for reverse engineering

third party, closed, binary Android apps. It disassembles APK files to nearly original form and rebuild them after making some modifications, including almost all the aforementioned components of an APK file. For more information about the tool itself and the installation instructions, visit the website [9].

## 4.3 Extracted features:

After extracting each APK file, the significant features should be extracted. For this project, we focused on permissions, activities and actions.
**Permissions:**
The permissions located in the Android Manifest file represent systems permissions such as *Android.permission.ACCESS_WIFI_STATE* or
*Android.permission.CAMERA* that must be granted by the user to allow all functionalities that use those permissions to function properly.
**Activities:**
The activities located in the Android Manifest file represent parts of the application's visual user interface such as *com.paktor.activity.HelpActivity* and *com.abs.MainActivity*. They must be included in the Android manifest file and represented by <activity>for the system, in order to display those activities.
**Actions:** Different actions are added to the intent filters which declares the capabilities of its parent components in terms of what an activity or service can do and what types of broadcasts a receiver can handle. Such intent filters (<intent-filter>) must contain the actions (<Action>) to be accepted by an Intent object.
To exctract the aforementioned features from the Android Manifest file, a Python script has been created that analyzes the Android Manifest file of every malware APK and benign APK. Subsequently, all features are listed in one file.

## 4.4 Data preprocessing:

To simplify the classification and explanation of the Android applications, each application was converted to a bitstring of the same length as the total number of features present in all the applications. Bit 1 means the presence of a certain feature in the Android application and bit 0 means the absence of the Android application.
This creates an issue for training the classification model which is the huge amount of features that the model has to train on. There are in total 18531 unique features present in all the application samples, so the classification model has to train on apps consisting of bitstring of 18531. Because most of the features were always absent in most applications, in exception to some applications, they had to be removed. In order to remove certain features, we resorted to feature selection. This reduces then number of input variables and subsequently reduces the computational cost of training the model and probably improve the accuracy of the classification model and the explanation approach.

# 5 Related work

Multiple metrics to evaluate explanation approaches have been proposed. The metrics proposed in this research were build upon metrics proposed by Ming Fan et al. in the paper [12]
Here, three metrics are proposed that evaluate Android application classifications: Stability, Robustness and Effectiveness. Stability refers to the stability in the generated explanations which must not influenced by fluctuation techniques used in the explanation approaches. Robustness refers to the similarity in the produced explaining features for similar instances. Ajay Chander Ramya Srinivasan also proposed the Effectiveness metric [7] which describes the ability of an explanation approach to support faster and better user decision-making. To measure the effectiveness of an explanation, the same classifier should be tested, with and without explanation facility and evaluate if instances who receive explanations end up with a classification more suited to the properties of its class. In contrary to changing fluctuation techniques or the input data generally, we propose two evaluation metrics that measure the effect on the produced explanation. The first metric measures the dependency on the classification model using the same input data. The second metric measures the effect of irrelevant features on the explanation. Detailed information will be given later on.
Different evaluation metrics were proposed in other papers. Kindermans et al. proposed the Sensitivity metric [21] which describes the capacity of an explanation approach to reflect the sensitivity of the underlying model with respect to variations in the input feature space. This metric states that if for every input and baseline that differ in one feature but have different predictions, then the differing feature should be given a non-zero attribution.
Freitas Alex A proposed also an evaluation metric which is Monotonicity [14]. This evaluation metric proposed that the relationship between a numerical predictor and the predicted class that occurs when increasing the value of the predictor leads to either always increase or decrease the probability of an instance's membership to the class. For example, when predicting whether or not a customer will buy a product, the probability of class "buy = yes" tends to decrease monotonically with an increase in the product's price.
Alvarez et al. proposed the Faithfulness metric [1] which describes the ability of an explanation approach to select truly relevant features. To measure this metric, some features are removed. Then, the drop in probability of the predicted class. In other work, Lipton proposed Informativeness[24], which describes the ability of an explanation approach to provide useful information to end-users.

# 6 Evaluation

In this section of research, the focus will lay on the evaluation of the different aforementioned explanation approaches (Anchors, EDC, LIME, MSE and SHAP). Based on the evaluation of these explanations, improvements may be applied to them in order to get more accurate explanation of the decisions made

by the used models, subsequently improvements to the classification models can also be made for example to remove bias and improve their performance.
Essentially, an explanation can be defined as the result of a classification in combination with a feature set. So there exists an explanation for a certain *classification* and a certain *feature set*. Based on this definition of an explanation, some properties can be deduced.

## 6.1 Explanation properties

**Property 1:**

Notice that an explanation is the result of a classification and not the model used for the explanation. Which means that the explaining feature set should be identical for the same feature set and different classification models.
So regardless of the model, the explanation remains the same. What defines a malicious application is the different malicious behaviors of that specific applications. For example gathering user data and storing them in a sqLite databse (GingerMaster Malware family). Each of these behaviors can be expressed as a set of features. When a classification model classifies this application samples, it should be able to identify the set of features that represent this malicious behavior. Of course, an application sample cannot adhere completely to this property without any caveats which will be discussed in chapter 10. This property can be summarized in the following formula:

$$Exp(c_i, F) = Exp(c_j, F)$$

where
$c_i, c_j$: $c_i, c_j \in C^2$, where C is the set containing all classification models
F: The original feature set of the application sample
Exp(.): Exp is the explanation function with Exp: $C \times \mathcal{P}(F) \longrightarrow \mathcal{P}(F)$

Another property can be deduced based on the definition which focuses on the feature set part.

**Property 2:**

Irrelevant features do not change the explaining feature set of an application sample substantially.

$$Exp(c_i, F) = Exp(c_i, F \cup S)$$

With $S$ being a set of irrelevant features. The irrelevant features can be extracted using feature selection. For feature selection we use variance of features across the applications present in the dataset. The most irrelevant features are features with low variance.

A feature can be for example a permission, intent or action. Thus, each feature describes a part from a certain behavior of an application sample. So a combination of features can fully describe a behavior of an application which probably makes the classification model classify the application as malware. By adding irrelevant features, some behavior could be added to the application (a behavior in terms of features and what the possible actions can be based on the added feature). But these irrelevant features only add a behavior to the already existing set of behaviors of the application and can not remove the fact that another set of features results the application to be classified as malware. On the other hand, irrelevant features are most of the time standard permissions, actions and intents, thus they do not contribute to the classification of the application.
For property 2, we assume that the classification model is the same for both explanations to remove any external noise from the explanation not adhering to the first property. Irrelevant features only add noise to the dataset thus the core explanation remains the same regardless of the added irrelevant features. Irrelevant features could for example be features that were removed during the preprocess phase which do not add any contribution to the classification of the application sample.

Based on the aforementioned two properties, we can deduce a metric for each of the two properties.

## 6.2 Evaluation metrics

**Consistency Rate (CR):**

This metric expresses the degree to which an explanation approach adheres to the first explanation property. That is, how much the resulting explaining features change if different classification methods where used on the same feature set?
Consistency Rate can formally defined by the following:

Let $C = \{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, ..., \mathcal{C}_n\}$ be a set of classification models and $S = \{\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3, ..., \mathcal{E}_n\}$ a set of explanations, where $\mathcal{E}_i$ is the explaining feature set produced by an explainer using classification model $\mathcal{C}_i$. We define set $Z$ as $Z = \bigcup_{i=1}^{n} \mathcal{E}_i$. In other words, Z contains all individual features produced by the explanations contained in $S$. CR is formulated as follows:

$$CR = \sum_{i \in Z} \frac{\sum_{s \in S} \frac{f(i,s)}{|S|}}{|Z|} \tag{1}$$

With:

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

We elaborate more on the functionality of each term of the CR formula above as follows. $f(i, s)$ controls the presence of feature i in the explanation feature set $s$. It returns 1 if it is present, otherwise, it returns 0. The presence of feature i is being checked in each explanation features sets produced by the explainer using different classifiers. Subsequently, this value is normalized over the number of classifiers. Again, this is summed up over all produced features and normalized by the total number of features.
As an example for this metric, the following situation can be presented:
An explainer produces the features $\{a, b, c\}$ using classification model $c_1$ and features $\{b, c, f\}$ using classification model $c_2$.

| feature | Feature presence over classifiers |
|---|---|
| a | 1/2 |
| b | 2/2 |
| c | 2/2 |
| d | 1/2 |

Feature presence over classifiers represent the the subformula: $\sum_{i \in Z} \frac{\sum_{s \in S} f(i,s)}{|S|}$.
The values of the feature presence over classifiers are then summed up and divided by the number of total number of features $|Z|$ giving the total value of CR=(1/2+2/2+2/2+1/2)/4=0.75.
The higher the CR value is, the more the identical the produced futures are between different classification models. Conversely, the lower the CR value is, the less identical the features are between different classification models. Ideally, this value would be 1 (i.e. if all the features are present in all explanations produced using different classification models). The worst case of the CR value of an explanation is the case where all features are present in a single set of produced features, namely their original set resulting in $\sum_{s \in S} f(i, s) = 1$. If we take the fraction of this value over the number of classifiers $|S|$ we get $\frac{\sum_{s \in S} f(i,s)}{|S|} = \frac{1}{|S|}$. Then if we sum up these fractions over the total number of features we get $\sum_{i \in Z} \frac{\sum_{s \in S} f(i,s)}{|S|} = \frac{|Z|}{|S|}$ Finally to calculate the CR value we still have to divide this value over the number of features getting CR=$\frac{1}{|S|}$.

**Soundness Rate (SR):**

This metric expresses the degree to which an explanation approach adheres to the second explanation property. That is, how much does the resulting explaining features change if irrelevant features (features filtered out from the original feature set in the preprocess stage) were to be added to the feature set?
Now, two probably different feature sets should be compared. The explaining feature set produced by the explanation approach using the original feature set with the explaining feature set produced by the explanation approach using the feature set including the irrelevant features. In addition to the existing irrelevant features (if any) with the added irrelevant features to the original feature set.
We hereby propose the metric Soundness Rate (SR) as follows:
Let $\mathcal{F}_1$ be the original feature set and $\mathcal{S}_1$ the explaining feature set produced by an explanation approach using a certain classification model $C_i$ and feature set $\mathcal{F}_1$. Let $\mathcal{F}_2$ be the added irrelevant features to the original feature set $\mathcal{F}_1$ such that $\mathcal{F}_1 \cap \mathcal{F}_2 = \emptyset$ and $\mathcal{S}_2$ the produced features by the explanation approach using the irrelevant features. As mentioned in 6.1, irrelevant features are the features with the lowest variance across the dataset. For that, 50% of the features were relevant features (all features excluding irrelevant features) and 50% of the features are irrelevant features. This is to ensure that the both parts have the the same probability to be influenced by the other (relevant features may be influenced by relevant features and vice versa)
In other words, without adding irrelevant features: $Exp(c_i, \mathcal{F}_1) = \mathcal{S}_1$ and by adding the irrelevant features: $Exp(c_i, \mathcal{F}_1 \cup \mathcal{F}_2) = \mathcal{S}_2$. Additionally, let $\mathcal{R}_{irr}$ the set of irrelevant features present in $\mathcal{S}_2$ and $\mathcal{R}_{or}$ the original features present in $\mathcal{S}_2$. Original features are features that are present in the explaining feature set produced by the explainer without using the irrelevant features. So $Exp(c_i, \mathcal{F}_1 \cup \mathcal{F}_2) = \mathcal{S}_2 = \mathcal{R}_{or} \cup \mathcal{R}_{irr}$ with $\mathcal{R}_{or} \subseteq \mathcal{F}_1$ and $\mathcal{R}_{irr} \subseteq \mathcal{F}_2$.
To illustrate this, figure 6 shows a simplified version of the explanation in action. This explanation was performed using the original features set. Additionally, figure 7 shows a simplified version of the explanation in action using the original feature set and the irrelevant feature set.

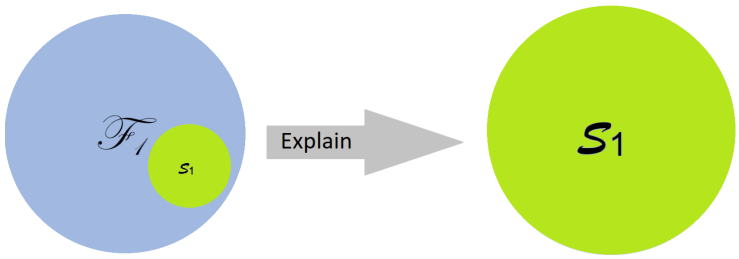


Figure 6: Simplified illustration of the explanation using the original feature set $\mathcal{F}_1$ that produces explaining feature set $\mathcal{S}_1$

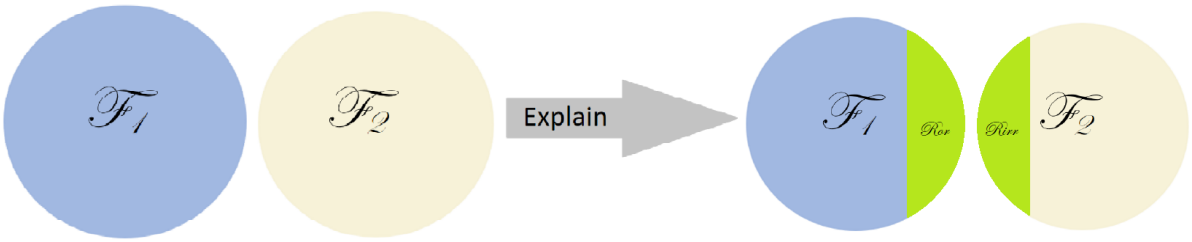


Figure 7: Simplified illustration of the explanation using the original feature set $\mathcal{F}_1$ and including irrelevant feature set $\mathcal{F}_2$ that produces explaining feature set $R_o \cup R_i = \mathcal{S}_1$

The Soundness Rate is formally defined as follows:

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

The goal is to measure the presence of irrelevant features in the produced feature set in addition to the effect of the irrelevant features on the original feature set. So $\frac{\|\mathcal{R}_{irr}\|}{\|\mathcal{F}_2\|}$ describes the ratio of the produced features over the added features. The closer this value is to 1, the more features from the irrelevant features are present in the produced feature set. After adding the irrelevant features to the original set, it might be the case that the explanation approach decides that the original explaining features are not relevant anymore for the explanation. To counter this the term CR was added to measure the difference between original explaining features and the original features present in the produced features by the explaining using the irrelevant feature. In other words, we want to measure the effect of the irrelevant features on the explaining original feature set.
Concerning $CR$, we use formula 1 with $Z = \mathcal{R}_{or} \cup \mathcal{S}_1$ and $\mathcal{S}$ such that $\mathcal{S} = \{\mathcal{R}_{or}, \mathcal{S}_1\}$. With this, CR is only used to measure in feature sets and not the influence of classification models on an explanation approach.
As an example of this metric, the following situation can be presented: A certain explanation approach uses features $\mathcal{F}_1 = \{a, b, c, d, e\}$ to produce an explanation containing features $\mathcal{S}_1 = \{b, d, e\}$. We now add a set of irrelevant features $\mathcal{F}_2 = \{x, y, z\}$ to the original feature set and get $\mathcal{F}_1 \cup \mathcal{F}_2 = \{a, b, c, d, e, x, y, z\}$. The same explanation approach uses this new feature set and produces the new set of explanation features $\{b, d, y\}$ with $\mathcal{R}_{or} = \{b, d\}$ and $\mathcal{R}_{irr} = \{y\}$.
If we calculate the Soundness rate we get SR=0.25*2/3 + 0.75.5/6≈0.8
Dependent on the performance of an explanation approach, the value fluctuates between $0.25 * 0 + 0.75 * 1/2 = 0.375$ and 0.25+0.75=1. Explanation approaches that produces feature sets that contain all the added irrelevant features and totally different features from the features produced by the explainer using the dataset without irrelevant features have the lowest SR value which is 0.375. Alternatively, explainers that produce exactly the same features as the feature set produced without adding the irrelevant features.

# 7 Experiment setup

In this section, the experiments of the proposed metrics are described. First we specify the goals of the performed experiments. Afterwards, we describe the performed experiments and then present the pseudocode of each performed experiment.

## 7.1 Experiment 1: Consistency Rate

As mentioned in section 6.2, Consistency Rate refers to the degree to which the explaining features change if the classification models mentioned in section 3.1 were used. In other words, the questions that we want to answer are:

1. Is there any difference in the explaining features when different classification models are used?
2. If there is a difference in the explaining features when different classification models are used, how big is it?
3. Is this difference the same for other explanation approaches? Or is it bigger/smaller?
4. What is according to this metric the *best* performing explanation approach?
5. What is according to this metric the *worst* performing explanation approach?

To answer these questions we have to run each of the explanations on different classifiers. The used classifiers are mentioned in section 3.1 in addition to the `BaggingClassifier()` provided by scikit-learn package and the explanation approaches mentioned in section 3.2. So the relevant features are first selected to reduce the amount of features and improve the accuracy of the classifiers. Each time, a new classification model is trained. Based on this classification and the type of classifier, a record will be given to the explainer (in case it is a local explainer) The explanation approach then produces an explanation in the form of a feature set.
To answer the first question, the explaining feature set is stored in a list. After running the explanation approach on the ten classification approaches, 10 features sets are stored in the list. These sets are then compared with each other to determine whether there is a difference between them.
To answer the second question, the Consistency Rate metric can determine how big the difference is between these feature sets are. This also answers question three.
To answer question four and five, we run this experiment on different explanation approaches. As mentioned in section 3.2, we focused for our scope of research on LIME, Anchors, EDC, SHAP and Morris sensitivity, but this metric is also applicable on some other explanation approaches. After running the

explanations, we can determine the best performing explanation approach and the worst performing explanation approach based on their Consistency Rate value. The explanation approach with highest Consistency Rate is the worst performing and the explanation approach with the lowest Consistency Rate is the best performing one.

## 7.2 Pseudocode:

**Algorithm 1** Pseudocode Consistency Rate experiment

```
Initialize:
Explainers, list containing all the explainer objects
explanation, a list of explaining features
Classifiers, list containing all the classifier objects
App, a list of 0's and 1's describing the state of the features of the application to be classified and explained
CR, a list of Consistency rate values of the given list of Explainers

for explainer in Explainers do
    Initialize: ExplainingFeatures, an empty list
    for classifier in Classifiers do
        classification ← classifier(App)
        explanation ← explainer(classifier, App)
        Append explanation to ExplainingFeatures
    end for
    Append CRValue(ExplainingFeatures) to CR
end for

Return CR
```

## 7.3 Experiment 2: Soundness Rate

As mentioned in section 7.4, the Soundness Rate refers to the degree to which the explaining features of a certain explanation approach if irrelevant features were added to the original feature set. Here the following questions should be asked:

1. Does adding irrelevant features effect the explaining feature set?
2. Are the added irrelevant features present in the explaining features set?
3. Are there any new relevant features present in the explaining features?
4. How much effect does adding irrelevant features have to the explaining feature set

5. Is the same effect present in the explaining feature sets of the other explainers?

6. What is according to this metric the *best* performing explanation approach?

7. What is according to this metric the *worst* performing explanation approach?

To answer all these questions, we run each explanation approach twice. On the first run, the explainer works on the original features set were the most relevant features are selected. This will generate an explaining feature set that is stored as a list. On the second run, the explainer works with the original feature set in addition to a randomly selected subset of the irrelevant features. The explaining features in this case are also stored in a list.
Now, we compare the first and second explaining feature sets. If the two are the same it can be said that adding irrelevant feature set does not any effect on the explainer. Otherwise, there will be clearly an effect. Question two can be answered by looking into the explaining feature set produced on the second run. If the added irrelevant features are present, the answer to the second question will be trivial.
For question three, we will compare the produced relevant features of the second run with the feature set produces on the first run. If there is a difference between the two, there is clearly an effect on the original features.
The fourth question can be answered by the Soundness Rate value defined in section 7.4. The higher this value is, the more effect the irrelevant feature have on the explainer. The lower this value, the less effect these feature have on the explainer.
Question four can be answered by comparing the Soundness rate of the explainers. For the last two question hold that the explanation approach with highest Soundness Rate is the worst performing and the explanation approach with the lowest Soundness Rate is the best performing one.

## 7.4 Pseudocode:

**Algorithm 2** Pseudocode Soundness Rate experiment

Initialize:
**Explainers**, list containing all the explainer objects
**Apps** ← training data
**ReleventFeatures** ← selectFeatures(**Apps**)
**IrrelevantFeatures** ← Random sample from the features not present in **ReleventFeatures**
**Classifier**, used classification model
**explanation**, a list of explaining features
**App**, a list of 0's and 1's describing the state of the features of the application to be classified and explained
**SR**, a list of Soundness rate values of the given list of *Explainers*

**for** *explainer* in **Explainers do**
    Train **classifier** on **ReleventFeatures**
    **explanation1** ← explainer(classifier, **App**)

    Train **classifier** on **ReleventFeatures+IrrelevantFeatures**
    **explanation2** ← explainer(classifier, **App**)
    Append SRValue([explanation1, explanation2]) to **SR**

**end for**

Return **SR**

The classification models were called as follows:

```
AdaBoostClassifier()
SVC(probability=True)
RandomForestClassifier(n_estimators=50, n_jobs=5)
BaggingClassifier()
MLPClassifier(alpha=1, max_iter=1000)
KNeighborsClassifier(1)
GaussianNB()
GradientBoostingClassifier()
GaussianProcessClassifier()
DecisionTreeClassifier(max_depth=5)
```

Concerning the irrelevant features, the number of relevant features and irrelevant features were equal as mentioned in . For more information about the the implementation of the following: extraction of the apk files in the dataset, collecting features from AndroidManifest files, training the classification models and implementing the five aforementioned explanation approaches as well as the experiments refer to the source code of this research [38] Concerning SHAP, this

explanation approach produces explanations in form of rulesets. Each rule in the produced ruleset is a combination of a certain feature and a specific value of that feature. In order to perform measurements on this explanation approach, each possible ruleset (each occuring feature with its value) is encoded as an array of characters which is perceived by the proposed metrics as a single feature.

## 7.5 Hardware

All the experiments including training the classification models and generating their explanations were run on the Duranium server of the LIACS (Leiden Institute of Advanced Computer Science ) Data Science Lab. Duranium has the following specifications:

| | |
|---|---|
| CPU | 20 Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz cores |
| GPU | 2x GeForce GTX TITAN X and 6x GeForce GTX 980 Ti |
| Storage | 3TB local storage |
| RAM | 256GB |
| OS | CentOS Linux 7 (Core) |

# 8 Experiment results

In this section, the results of the two experiments are presented.

## 8.1 Experiment 1: Consistency Rate

Here, we want to check if there is a difference between the explaining features when different classification models are used. This difference is illustrated by the presence of different colors. When performing the explanations for different classification models, we get the following results:

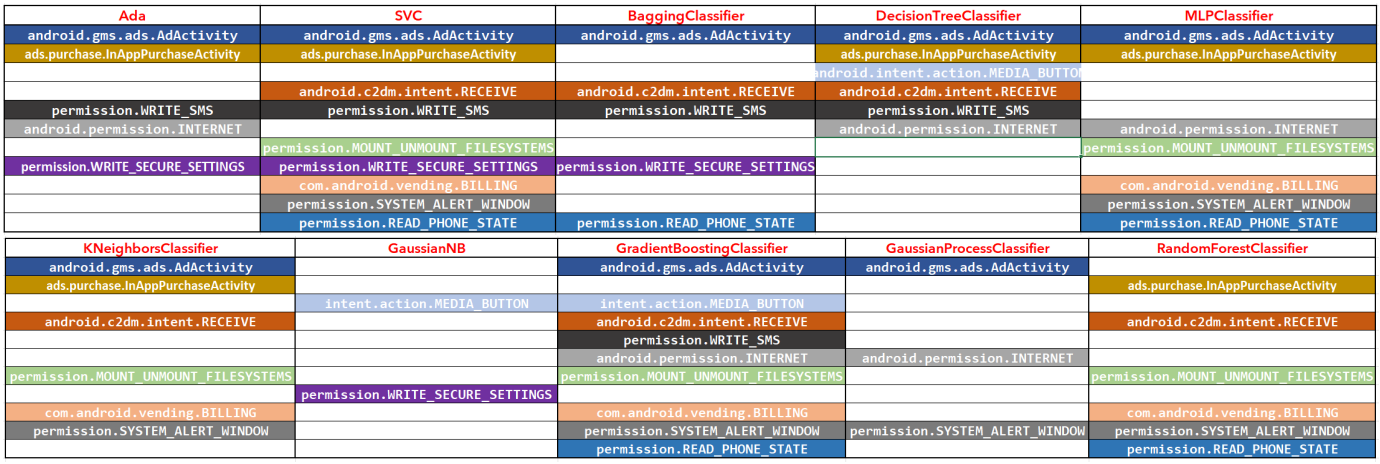


Figure 8: Most occuring features for the MorrisSensitivity explanation approach for a single application instance.

Cells with the same color represent the same feature. Each row is specified for a certain feature. The dominance of a specific color in a certain row means the consistency of that explanation on that feature. The opposite holds also. If a specific color is present in just a few cells of a certain row, then the explanation approach is not consistent on that specific feature over different classification models. Why certain explanation approaches are performed on just a few classification models will be discussed in section 10.

The tables shown before, present just a small subset of the explaining features. They are certainly not enough to evaluate the explanation. We need to evaluate the explanation approaches based on all produced explaining features by using their Consistency Rate value. The calculated Consistency rate value for each explanation is shown in figure 13 This allows us to compare the performance of different explanation approaches an evaluate their consistency.

| Ada | BaggingClassifier | MLPClassifier |
|---|---|---|
| android.permission.WRITE_SMS | android.permission.WRITE_SMS | android.permission.WRITE_SMS |
| permission.ACCESS_COARSE_LOCATION | | permission.ACCESS_COARSE_LOCATION |
| android.intent.action.BOOT_COMPLETED | | android.intent.action.BOOT_COMPLETED |
| android.permission.INSTALL_PACKAGES | | android.permission.INSTALL_PACKAGES |
| app.action.DEVICE_ADMIN_ENABLED | | |
| | | app.action.DEVICE_ADMIN_DISABLED |
| permission.READ_PHONE_STATE | permission.READ_PHONE_STATE | |

| GradientBoostingClassifier | RandomForestClassifier | DecisionTreeClassifier |
|---|---|---|
| android.permission.WRITE_SMS | android.permission.WRITE_SMS | android.permission.WRITE_SMS |
| permission.ACCESS_COARSE_LOCATION | | |
| android.intent.action.BOOT_COMPLETED | android.intent.action.BOOT_COMPLETED | |
| android.permission.INSTALL_PACKAGES | android.permission.INSTALL_PACKAGES | |
| app.action.DEVICE_ADMIN_ENABLED | app.action.DEVICE_ADMIN_ENABLED | |
| | app.action.DEVICE_ADMIN_DISABLED | |
| permission.READ_PHONE_STATE | | |
| | | android.c2dm.intent.RECEIVE |

Figure 9: Most occuring features for the EDC explanation approach.

| Ada | BaggingClassifier | RandomForestClassifier |
|---|---|---|
| android.permission.WRITE_EXTERNAL_STORAGE | android.permission.WRITE_EXTERNAL_STORAGE | android.permission.WRITE_EXTERNAL_STORAGE |
| google.android.c2dm.permission.RECEIVE | | google.android.c2dm.permission.RECEIVE |
| | android.permission.WAKE_LOCK | android.permission.WAKE_LOCK |
| permission.ACCESS_NETWORK_STATE | permission.ACCESS_NETWORK_STATE | permission.ACCESS_NETWORK_STATE |
| | google.android.c2dm.intent.RECEIVE | google.android.c2dm.intent.RECEIVE |
| | android.permission.INTERNET | android.permission.INTERNET |
| | | android.permission.GET_ACCOUNTS |
| | ACCESS_FINE_LOCATION | |

Figure 10: Most occuring features for the SHAP explanation approach.

## 8.2 Experiment 2: Soundness Rate

In this section, we want to know whether the produced explaining features are truly the reason behind the explanation. In order to test this, we add irrelevant features to the original dataset and check their effect on the explaining features. Tables 14 and 15 are presented to describe the output of different explanation approach before and after adding the irrelevant features. The green colored features show features that are present in both explanations: the explanation using the original explaining features and the explanation that also includes the irrelevant features. The orange colored features are features from the original dataset but different from the original explaining features. The red features are irrelevant features the appear as explaining features.

Concerning the Soundness rates for the different explanation approaches, the produced values are shown in figure 16

| Ada | SVC | BaggingClassifier | DecisionTreeClassifier | MLPClassifier |
|---|---|---|---|---|
| android.permission.CALL_PRIVILEGED | android.permission.CALL_PRIVILEGED | android.permission.CALL_PRIVILEGED | android.permission.CALL_PRIVILEGED | |
| android.app.action.DEVICE_ADMIN_ENABLED | android.app.action.DEVICE_ADMIN_ENABLED | android.app.action.DEVICE_ADMIN_ENABLED | android.app.action.DEVICE_ADMIN_ENABLED | android.app.action.DEVICE_ADMIN_ENABLED |
| android.permission.WRITE_SMS | android.permission.WRITE_SMS | android.permission.WRITE_SMS | android.permission.WRITE_SMS | android.permission.WRITE_SMS |
| android.permission.DELETE_PACKAGES | android.permission.DELETE_PACKAGES | android.permission.DELETE_PACKAGES | android.permission.DELETE_PACKAGES | android.permission.DELETE_PACKAGES |
| android.gms.ads.AdActivity | android.gms.ads.AdActivity | android.gms.ads.AdActivity | android.gms.ads.AdActivity | android.gms.ads.AdActivity |
| ussd.IExtendedNetworkService | ussd.IExtendedNetworkService | ussd.IExtendedNetworkService | ussd.IExtendedNetworkService | |
| permission.ACCESS_COARSE_UPDATES | permission.ACCESS_COARSE_UPDATES | permission.ACCESS_COARSE_UPDATES | permission.ACCESS_COARSE_UPDATES | |
| permission.INSTALL_PACKAGES | permission.INSTALL_PACKAGES | permission.INSTALL_PACKAGES | | permission.INSTALL_PACKAGES |
| action.DEVICE_ADMIN_DISABLE_REQUESTED | action.DEVICE_ADMIN_DISABLE_REQUESTED | action.DEVICE_ADMIN_DISABLE_REQUESTED | action.DEVICE_ADMIN_DISABLE_REQUESTED | |
| launcher.permission.INSTALL_SHORTCUT | launcher.permission.INSTALL_SHORTCUT | | | launcher.permission.INSTALL_SHORTCU |
| android.permission.AUTHENTICATE_ACCOUN | | android.permission.AUTHENTICATE_ACC | | |

| KNeighborsClassifier | GaussianNB | GradientBoostingClassifier | GaussianProcessClassifier | RandomForestClassifier |
|---|---|---|---|---|
| | android.permission.CALL_PRIVILEGED | android.permission.CALL_PRIVILEGED | android.permission.CALL_PRIVILEGED | android.permission.CALL_PRIVILEGED |
| android.app.action.DEVICE_ADMIN_ENABLED | android.app.action.DEVICE_ADMIN_ENABLED | android.app.action.DEVICE_ADMIN_ENABLED | android.app.action.DEVICE_ADMIN_ENABLED | android.app.action.DEVICE_ADMIN_ENABLED |
| | android.permission.WRITE_SMS | android.permission.WRITE_SMS | android.permission.WRITE_SMS | android.permission.WRITE_SMS |
| | android.permission.DELETE_PACKAGES | android.permission.DELETE_PACKAGES | android.permission.DELETE_PACKAGES | android.permission.DELETE_PACKAGES |
| android.gms.ads.AdActivity | | android.gms.ads.AdActivity | | android.gms.ads.AdActivity |
| | ussd.IExtendedNetworkService | ussd.IExtendedNetworkService | ussd.IExtendedNetworkService | ussd.IExtendedNetworkService |
| permission.ACCESS_COARSE_UPDATES | permission.ACCESS_COARSE_UPDATES | | permission.ACCESS_COARSE_UPDATES | permission.ACCESS_COARSE_UPDATES |
| | permission.INSTALL_PACKAGES | permission.INSTALL_PACKAGES | permission.INSTALL_PACKAGES | |
| action.DEVICE_ADMIN_DISABLE_REQUESTED | action.DEVICE_ADMIN_DISABLE_REQUESTED | | action.DEVICE_ADMIN_DISABLE_REQUESTED | |
| | launcher.permission.INSTALL_SHORTCUT | launcher.permission.INSTALL_SHORTCUT | launcher.permission.INSTALL_SHORTCUT | launcher.permission.INSTALL_SHORTCUT |
| | android.permission.AUTHENTICATE_ACCOUNTS | | android.permission.AUTHENTICATE_ACCOUNTS | |

Figure 11: Most occuring features for LIME explanation approach.

| Ada | SVC | BaggingClassifier | DecisionTreeClassifier | MLPClassifier |
|---|---|---|---|---|
| | android.permission.DELETE_PACKAGES | | | android.permission.DELETE_PACKAGES |
| | | | android.permission.CALL_PRIVILEGED | |
| android.permission.INSTALL_PACKAGES | | android.permission.INSTALL_PACKAGES | | |
| | android.app.action.DEVICE_ADMIN_ENABLED | | | android.app.action.DEVICE_ADMIN_ENABLED |
| .AppEntry | | | | |
| android.accounts.AccountAuthenticator | | | | |
| com.android.vending.INSTALL_REFERRER | | | | |
| | | com.google.android.gms.ads.AdActivity | | |
| | | | | |
| | | | com.google.android.gms.ads.AdActivity | |

| KNeighborsClassifier | GaussianNB | GradientBoostingClassifier | GaussianProcessClassifier | RandomForestClassifier |
|---|---|---|---|---|
| | android.permission.DELETE_PACKAGES | android.permission.DELETE_PACKAGES | | |
| android.permission.CALL_PRIVILEGED | android.permission.CALL_PRIVILEGED | | | android.permission.CALL_PRIVILEGED |
| | | | android.permission.INSTALL_PACKAGES | |
| | | android.app.action.DEVICE_ADMIN_ENABLED | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | android.permission.FLASHLIGHT | |
| | | | | |

Figure 12: Most occuring features for the Anchors explanation approach.

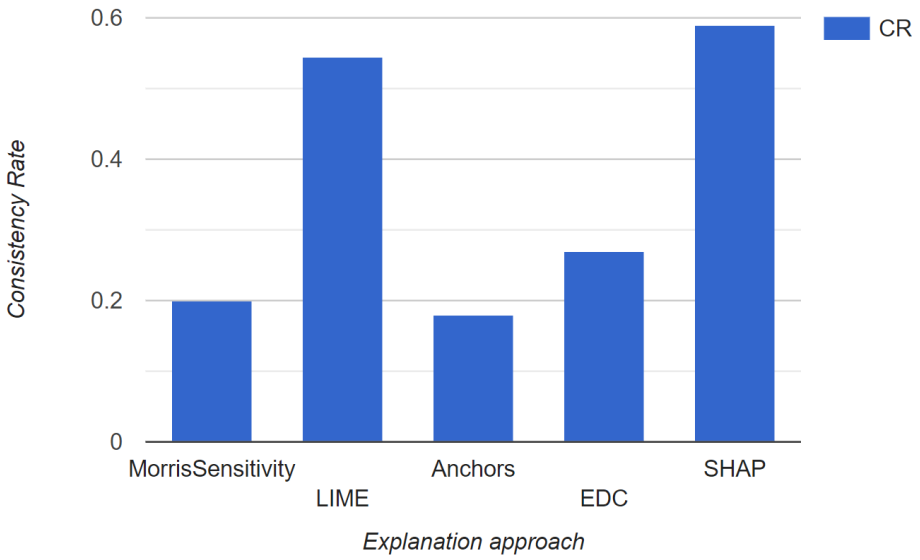


Figure 13: Consistency rate for different explainers

| | Original eplaining features | Eplaining features including irrelevant features |
|---|---|---|
| MorrisSensitivity | com.google.android.gms.measurement.UPLOAD | com.google.android.gms.measurement.UPLOAD |
| | com.google.android.gms.ads.AdActivity | com.google.android.gms.ads.AdActivity |
| | android.permission.RECEIVE_SMS | android.permission.RECEIVE_SMS |
| | android.permission.SYSTEM_ALERT_WINDOW | android.permission.SYSTEM_ALERT_WINDOW |
| | com.android.vending.BILLING | com.android.vending.BILLING |
| | android.permission.USE_CREDENTIALS | android.permission.USE_CREDENTIALS |
| | android.intent.action.VIEW | android.intent.action.VIEW |
| | android.permission.GET_ACCOUNTS | android.permission.GET_ACCOUNTS |
| | android.permission.CALL_PHONE | android.permission.CALL_PHONE |
| | android.permission.READ_PHONE_STATE | android.permission.READ_PHONE_STATE |
| | android.permission.WRITE_ETERNAL_STORAGE | android.permission.WRITE_ETERNAL_STORAGE |
| | com.google.android.c2dm.permission.RECEIVE | com.google.android.c2dm.permission.RECEIVE |
| | com.facebook.FacebookActivity | mobi.supo.battery.ACTION_SWITCH_ANALYTICS_URL_TO_TEST |
| | com.google.android.c2dm.intent.REGISTRATION | com.google.android.c2dm.intent.REGISTRATION |
| | android.permission.SEND_SMS | android.permission.SEND_SMS |
| | android.permission.RECEIVE_BOOT_COMPLETED | droidplugin.stub.ActivityStub$Dialog$P04$SingleTask00 |
| | android.net.conn.CONNECTIVITY_CHANGE | com.taobao.accs.intent.action.RECEIVE |
| | android.intent.action.USER_PRESENT | android.ui.device.CamTalkDeviceCgiTestActivity |
| | android.permission.READ_SMS | droidplugin.stub.ActivityStub$P07$SingleTask00 |
| | android.permission.ACCESS_COARSE_LOCATION | android.intent.action.MEDIA_BUTTON |
| LIME | android.permission.INSTALL_PACKAGES | android.permission.INSTALL_PACKAGES |
| | com.google.android.gms.ads.AdActivity | com.google.android.gms.ads.AdActivity |
| | android.permission.RECEIVE_SMS | android.permission.RECEIVE_SMS |
| | android.net.conn.CONNECTIVITY_CHANGE | android.provider.Telephony.SMS_RECEIVED |
| | android.app.action.DEVICE_ADMIN_ENABLED | android.app.action.DEVICE_ADMIN_ENABLED |
| | android.permission.SYSTEM_ALERT_WINDOW | android.permission.SYSTEM_ALERT_WINDOW |
| | android.permission.WRITE_SMS | android.permission.WRITE_SMS |
| | com.google.android.c2dm.permission.RECEIVE | com.google.android.c2dm.permission.RECEIVE |
| | android.intent.action.VIEW | android.permission.DELETE_PACKAGES |
| | android.permission.READ_PHONE_STATE | android.permission.READ_PHONE_STATE |
| | android.permission.GET_ACCOUNTS | android.permission.GET_ACCOUNTS |
| | com.android.launcher.permission.INSTALL_SHORTCUT | android.permission.CALL_PHONE |
| | android.permission.ACCESS_COARSE_UPDATES | android.permission.ACCESS_COARSE_UPDATES |
| | android.app.action.DEVICE_ADMIN_DISABLED | android.app.action.DEVICE_ADMIN_DISABLED |
| | android.permission.MODIFY_PHONE_STATE | android.intent.action.BOOT_COMPLETED |
| | android.app.action.DEVICE_ADMIN_DISABLE_REQUESTED | android.permission.WRITE_CONTACTS |
| | android.permission.PROCESS_OUTGOING_CALLS | android.permission.AUTHENTICATE_ACCOUNTS |
| | com.android.ussd.IEtendedNetworkService | com.android.ussd.IEtendedNetworkService |
| | android.permission.CALL_PRIVILEGED | android.permission.CALL_PRIVILEGED |
| | android.permission.READ_CALL_LOG | android.permission.READ_CALL_LOG |

Figure 14: Explaining features of MorrisSensitivity and LIME before and after adding irrelevant features

| | | |
|---|---|---|
| ANCHORS | android.permission.INSTALL_PACKAGES | android.permission.INSTALL_PACKAGES |
| | android.accounts.AccountAuthenticator | android.permission.DISABLE_KEYGUARD |
| | com.google.android.gms.ads.AdActivity | android.permission.ACCESS_WIFI_STATE |
| | .AppEntry | |
| EDC | android.app.action.DEVICE_ADMIN_ENABLED | android.app.action.DEVICE_ADMIN_ENABLED |
| | android.permission.READ_PHONE_STATE | android.permission.READ_PHONE_STATE |
| | android.intent.action.BOOT_COMPLETED | android.intent.action.BOOT_COMPLETED |
| | android.permission.RECEIVE_SMS | android.permission.RECEIVE_SMS |
| | android.permission.WRITE_SMS | android.permission.WRITE_SMS |
| | android.permission.INSTALL_PACKAGES | android.permission.ACCESS_COARSE_LOCATION |
| SHAP | android.permission.WRITE_ETERNAL_STORAGE | android.permission.WRITE_ETERNAL_STORAGE |
| | android.permission.ACCESS_NETWORK_STATE | android.permission.ACCESS_NETWORK_STATE |
| | com.google.android.c2dm.permission.RECEIVE | com.google.android.c2dm.permission.RECEIVE |

Figure 15: Explaining features of Anchors, EDC and SHAP before and after adding irrelevant features

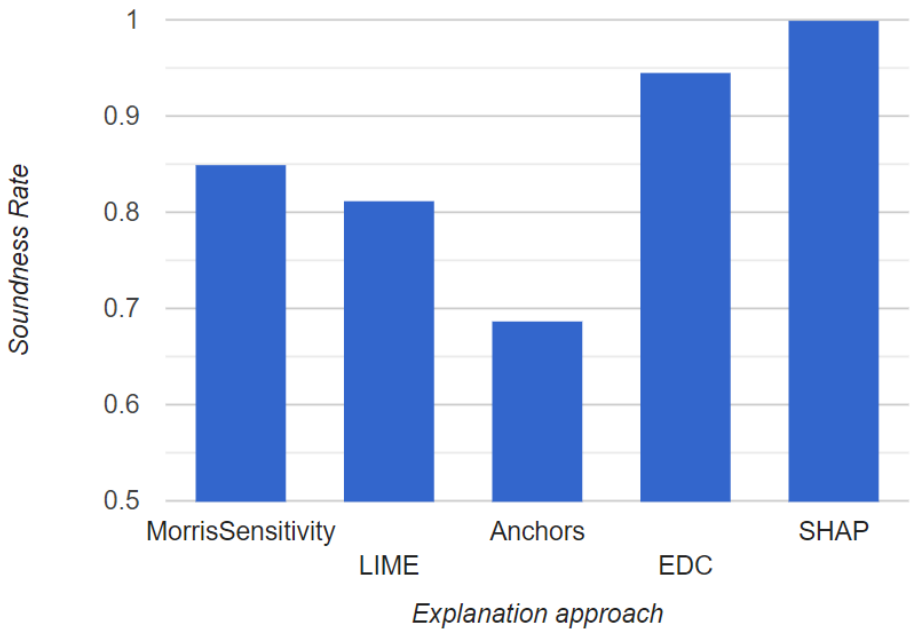


Figure 16: Soundess rate for different explainers

# 9 Discussion

## 9.1 Experiment 1

At this point, we have mentioned the explanation approaches to be evaluated in section 3.2 which are performed on the classification models mentioned in section 3.1. We also proposed two evaluation metrics in 6.2 and 7.4. Based on these two metrics, experiments 7.1 and 7.3 are performed. The results of these two experiments are presented in sections 10 and 8.2.
Concerning table 8, we observe that there are some similarities in terms of the produced explaining features between multiple runs of MorrisSensitivity on different classifiers. The feature "androi.gms.ads.AdActivity" is produced in all runs of the explainer except to the run where GaussianNB is used as classifier as well as GaussianProcess. In contrary, the feature "permission.WRITE_SECURE_SETTINGS" is only present in 4 out of 10 runs: AdaBoost, svc, Bagging and gaussianNB. Overall, this explainer shows big difference between the runs. Most features differ from classification model to the other, others are not even present. This can also be seen by the different colors that were used.
The Consistency rate value reflects this observations greatly. From figure 13, the CR value for MorrisSensitivity is 0.2, which is quite low. This shows that the produced explanation features depend on the classification model used. This explainers is relatively inconsistent.
Concerning table 9, we observe that the difference in colors (features) for the EDC explainer is slightly less than the difference of MorrisSensitivity. Feature "Android.c2dm.intent.RECEIVE" is only present in a single run (DecisionTreeClassifier) and "permission.ACCESS_COARSE_LOCATION" is present in 3 out of 6 runs of the explainer. Namely in AdaBoost, MLPClassifier and GradientBoosting. In contrary, feature "Android.permission.WRITE_EXTERNAL_STORAGE" is present in all runs of the explainer.
The Consistency rate value of EDC greatly agrees with this observation. The CR value of EDC is approximately 0.28 which is higher than MorrisSensitivity.
For table 10, we observe that feature "permission.ACCESS_NETWORK_STATE" and "Android.permission.WRITE_EXTERNAL_STORAGE" are present in all three runs of the SHAP explainer. We also observe that 4 other features are present in 2 out of 3 runs. This explainer has a Consistency Rate value of approximately 0.59 which shows that the explainer is relatively not dependent on the classification model used.
The experiment on the LIME explainer is the most interesting one. Table 11 shows relatively small difference in color between different runs of the explanation approach. Thus most of the features are relatively consistent among the runs of the explainer with different classification models. The Consistency rate of this explainer from 13 is reflects this observation. The CR value of LIME is 0.57 which is relatively high.
Concerning table 12, it is clear that illustrated colors on each run with different

classification model vary. Overall, the presence of each feature in different runs of the explainer is very low. Multiple features are only present in a single run, others are present in 3 out of 10 runs of the explainer. The CR value from figure 13 proves this observation. Anchors has the lowest CR value among all the used explanation approaches.

## 9.2 Experiment 2

The second experiment shows the variability of the explaining features if irrelevant features were to be added.
Table 14 shows that by adding the irrelevant feature set to the input set of MorrisSensitivity, the explaining features change. Six new features appear in the explaining features. These six features are all irrelevant features. However, the remaining 14 original features are still present in the second explaining feature set. This explains the Soundess Rate shown in figure 16.
The presence of these irrelevant features shows that this explanation approach can be effected by adding irrelevant features which by property 6.1 should not be the case.
For LIME (figure 14), no irrelevant features were present. However six new features from the original feature set are present in the explaining feature set that was produces by adding the irrelevant features to the input. 14 features remain untouched. The SR value of this explainer comes near to the SR value of MorrisSensitivity with 0.81.
Anchors shows in table 15 to be the most sensitive explanation approach among the five explanations to irrelevant features. Just a single feature "Android.permission.INSTALL_PACKAGES" is still present in the produced explaining features where the irrelevant features were added to the input. The remaining features are all features from the original feature set. This is also expressed by the SR value of Anchors. This explainer has the lowest SR vaue.
In this explanation, similar to LIME, no irrelevant features were present.
EDC and SHAP explainers are the most impressive explainers in terms of performance. EDC shows just a single alternating feature. For SHAP all the explaining features ramained the same. Their Soundness rates were outstanding compared to the rest of the explainers. EDC has an SR value of approximately 0.95 and SHAP earning the perfect value 1.

# 10 Limitations

This section describes some limitations of the performed experiments and the proposed properties/metrics.

Firstly, we assumed that the explanation approach do not produce random features. This is not completely the case as LIME for example produces random pertubations around the training records. This produces a slight randomness in the produced features. However, because this variability is very small it can be neglected.
Another limitation is that we assumed that the apps contained in the dataset are correctly labeled. Kaspersky estimated that there will be over 6 billion Android users by 2020 [18]. This rise in Android users leads to the rise of Android malware. Subsequently, the number of sophisticated Android malware applications will also rise which makes their classification more complex.

Another limitation for the second metric (Soundess Rate) is that the irrelevant features are selected randomly. The effect of the irrelevant features on the explaining features may change dependent on which irrelevant features were selected. Features with high variance across the application may be more likely to appear in the explaining features. The opposite holds also, features with low variance are likely to appear in the explaining features.

Concerning property 1, it is mentioned that a certain application behavior can be expressed as a set of features and the explainer should be able to identify these features that this malicious behavior consists of.
The issue with this point is that a malicious behavior can also be expressed by alternative features which end up by the same behavior. If the size of the training data is large enough, this limitation may be overcome, but for the size of training data used for this research, it is not clear whether this happens or not.

Concerning the Explanation approaches, due to some implementation limitations of the used libraries, the explanation approaches did not work on every proposed classification models as can be noticed in the experiment results . A typical error that would occur with EDC for example is `A sparse matrix was passed, but dense data is required. Use X.toarray() to convert to a dense numpy array.` With this, the proposed solution was tried but other errors would occur.
Additional to the aforementioned limitations, the first metric tends to be inaccurate for white box models. That is, explainers do not work on every classification model, subsequently, there will be less classifiers that the metric can take into account. If there are less classifiers to be considered, features are more likely to appear in the feature sets of the explanations generated using the different classification models.

The last limitation concerns a hardware limitation. All the classification

models used for this research had an accuracy over 85% on the test set. This accuracy could increase if we used a larger dataset. This dataset has to be extracted, preprocessed, classified and explained. In order to perform all these tasks on a larger scale, the hardware specification need to be better in order to make it feasible.

# 11 Conclusion and future work

There some points that need more research which may help to improve the accuracy of the metrics:

- In addition to the extracted features from the Android Manifest file, we can use the features from the Dalvik executable files. These may reveal important information which the Dalvik executables did not.
- We focused on static analysis to extract features but also dynamic analysis may be utilized. Dynamic analysis is performed while the application is running. This may reveal new information about the application which was not captured before with static analysis. with Code obfuscation for instance, dynamic analysis may be more applicable and somewhat easier to perform than static analysis. A combination of both static and dynamic analysis can be used.
- Range the CR value over multiple application instances. With a single instance it might be the case that for the used application sample the model was more dependable on the classification model then with other application instance, which is why more research on this part is worth.
- Research the experiment results on better performing hardware. With better performing hardware, the classification model can use more data to be trained on, which improves its accuracy and probably the accuracy of explanations. The precise effect on the explanations and the evaluation metrics is for now still unknown.
- Use bigger dataset. As mentioned before, with a bigger dataset the accuracy of the classification models can increase. However, it is still unknown if the explanation approaches will behave differently or not.
- Experiment on alternative datasets. All the experiments were performed on two fixed datasets. It is also unknown if the explanation approach will perform the same on new datasets and if ranking according to the Soundness Rate and Consistency Rate will remain the same.
- Investigate deeper the proposed properties and metrics to have a better understanding of their theoretical underpinnings (basically, how and why they work).
- Concerning CR value, this metric evaluates a local explanation approach based on a single instance. Other application instances may reveal different behavior. In order to overcome this, averaging over multiple application instances may be result more accurate estimations of the performance of the explanation approaches. This is definitely a point that needs further study.

- Currently, the implemented explanation approaches work on different number of classification models due to some occurring errors in implementation, which was mentioned in the limitations. In this case, the Consistency Rate may penalize explanation approaches that work on fewer classification models.

These are all point that may or may not reveal more information about the proposed metrics.

In conclusion, we proposed to properties that explanations need to adhere to. Based on these properties we proposed one evaluation metric for each property. The illustrated tables contain the generation of the explaining features from different sources. These tables demonstrate the strength and accuracy of the proposed metrics. The experiments revealed that, on the basis of the Consistency Rate, SHAP performs the best, following LIME, EDC, MorrisSensitivity and lastly Anchors. Based on the Soundness Rate, SHAP performed the best, following EDC, MorrisSensitivity, LIME and lastly Anchors.So, the SHAP explainer seems to be the best choice according to the proposed two evaluation metrics.

# References


[1] David Alvarez Melis and Tommi Jaakkola. Towards robust interpretability with self-explaining neural networks. *Advances in neural information processing systems*, 31, 2018.

[2] Android developers. Application fundamentals. https://developer.android.com/guide/components/fundamentals.

[3] David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Müller. How to explain individual classification decisions. *The Journal of Machine Learning Research*, 11:1803–1831, 2010.

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

[5] Abraham Bookstein, Vladimir A Kulyukin, and Timo Raita. Generalized hamming distance. *Information Retrieval*, 5(4):353–375, 2002.

[6] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[7] Ajay Chander and Ramya Srinivasan. Evaluating explanations by cognitive value. In *International cross-domain conference for machine learning and knowledge extraction*, pages 314–328. Springer, 2018.

[8] Hongge Chen, Huan Zhang, Si Si, Yang Li, Duane Boning, and Cho-Jui Hsieh. Robustness verification of tree-based models. *Advances in Neural Information Processing Systems*, 32, 2019.

[9] Connor Tumbleson, Ryszard Wiśniewski. Apktool. https://ibotpeaches.github.io/Apktool/.

[10] Per-Erik Danielsson. Euclidean distance mapping. *Computer Graphics and image processing*, 14(3):227–248, 1980.

[11] Elkind, Edith and Rothe, Jörg. Cooperative game theory, 2016.

[12] 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.

[13] Evelyn Fix and Joseph Lawson Hodges. Discriminatory analysis. nonparametric discrimination: Consistency properties. *International Statistical Review/Revue Internationale de Statistique*, 57(3):238–247, 1989.

[14] Alex A Freitas. Comprehensible classification models: a position paper. *ACM SIGKDD explorations newsletter*, 15(1):1–10, 2014.

[15] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

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

[17] James Joyce. Bayes' theorem. 2003.

[18] Kaspersky. Android mobile security threats. https://www.kaspersky.com/resource-center/threats/mobile.

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

[20] Been Kim, Rajiv Khanna, and Oluwasanmi O Koyejo. Examples are not enough, learn to criticize! criticism for interpretability. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[21] Pieter-Jan Kindermans, Sara Hooker, Julius Adebayo, Maximilian Alber, Kristof T Schütt, Sven Dähne, Dumitru Erhan, and Been Kim. The (un) reliability of saliency methods. arxiv e-prints, page. *arXiv preprint arXiv:1711.00867*, 2017.

[22] Alex X. Lee, Anusha Nagabandi, Pieter Abbeel, and Sergey Levine. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 741–752. Curran Associates, Inc., 2020.

[23] Leung, K Ming. Naive bayesian classifier, 2007.

[24] Zachary C Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57, 2018.

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

[26] Samaneh Mahdavifar, Dima Alhadidi, Ali Ghorbani, et al. Effective and efficient hybrid android malware classification using pseudo-label stacked auto-encoder. *Journal of Network and Systems Management*, 30(1):1–34, 2022.

[27] Samaneh Mahdavifar, Dima Alhadidi, Ali Ghorbani, et al. Exteciccicmaldroid 2020. 2022.

[28] Samaneh Mahdavifar, Andi Fitriah Abdul Kadir, Rasool Fatemi, Dima Alhadidi, and Ali A Ghorbani. Dynamic android malware category classification using semi-supervised deep learning. In *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, pages 515–522. IEEE, 2020.

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

[30] Manuel Eugenio Morocho-Cayamcela, Haeyoung Lee, and Wansu Lim. Machine learning for 5g/b5g mobile and wireless communications: Potential, limitations, and future directions. *IEEE Access*, 7:137184–137206, 2019.

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

[32] 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.

[33] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Model-agnostic interpretability of machine learning, 2016.

[34] 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.

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

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

[37] Eric Schulz, Maarten Speekenbrink, and Andreas Krause. A tutorial on gaussian process regression: Modelling, exploring, and exploiting functions. *Journal of Mathematical Psychology*, 85:1–16, 2018.

[38] Oussama Soulimani. Evaluating the explanation of Android malware detection. https://github.com/oussama-soulimani/Evaluating-the-explanation-of-android-malware-detection, 8 2022.

[39] Erik Strumbelj and Igor Kononenko. An efficient explanation of individual classifications using game theory. *The Journal of Machine Learning Research*, 11:1–18, 2010.

[40] Erik Štrumbelj and Igor Kononenko. Explaining prediction models and individual predictions with feature contributions. *Knowledge and information systems*, 41(3):647–665, 2014.

[41] Laya Taheri, Andi Fitriah Abdul Kadir, and Arash Habibi Lashkari. Malware dataset: Extecic-invesandmal2019. https://www.unb.ca/cic/datasets/invesandmal2019.html.

[42] Laya Taheri, Andi Fitriah Abdul Kadir, and Arash Habibi Lashkari. Extensible android malware detection and family classification using network-flows and api-calls. In *2019 International Carnahan Conference on Security Technology (ICCST)*, pages 1–8. IEEE, 2019.

[43] Laurens Verhagen. Laat toeslagenaffaire een les zijn bij beleid kunstmatige intelligentie. *volkskrant*, 2021.