

# Understanding Machine Learning Model Updates Based on Changes in Feature Attributions

FAN YUN<sup>1</sup> TOSHIKI SHIBAHARA<sup>2</sup> YUICHI OHSITA<sup>1</sup> DAIKI CHIBA<sup>2</sup> MITSUAKI AKIYAMA<sup>2</sup>  
MASAYUKI MURATA<sup>1</sup>

**Abstract:** Machine learning operations (MLOps) are widely adopted in various real-world machine learning (ML) systems. To ensure ML model performance in such systems, it is crucial to minimize the influence of insufficient data and concept drift. One simple and effective solution is to update models with newly added data. In MLOps, after updating a model, accuracy scores are usually used to validate the model. However, it is hard to obtain detailed information regarding the causes of performance changes. We therefore propose a method for understanding ML model updates by using a feature attribution method called Shapley additive explanations (SHAP), which explains the output of a ML model by assigning an importance value called a SHAP value to each feature. We calculate SHAP values using models before and after updates to investigate changes in SHAP values. By analyzing the extent of changes, we can identify the slight changes in models due to updates and the data related to the changes.

**Keywords:** MLOps, SHAP, feature contribution

## 1. Introduction

Machine learning (ML) has been applied to many systems to solve real-world problems, including computer vision and cybersecurity. Such ML-based systems adopt machine learning operations (MLOps) to automate and monitor the process of ML-based systems, from development to deployment and operation [1]. In operation, ML models might not achieve the expected classification performance. For example, when training data is insufficient or biased, classification performance will be lower in operation than in development [2]. Also, classification performance gradually degrades when statistical characteristics of data change over time [3], a phenomenon called concept drift [4]. In such situations, adding new data to the training dataset and updating the model can effectively improve classification performance of models.

After the update, the new model is validated using validation data in terms of accuracy or other aspects of classification performance [5]. Once the model is successfully validated, it can be deployed in real-world tasks. At validation, it is difficult to obtain information other than accuracy values. Although evaluating accuracy effectively confirms expected classification performance in real-world operation, we cannot obtain detailed information such as why performance improved or slight changes in the model affecting predictions for a small amount of validation data. Such information is beneficial in terms of MLOps because we can improve data collection efficiency based on types of insufficient data, and we can prevent unexpected predictions caused by

slight changes in the model.

To obtain detailed information about model updates, we propose a method for identifying samples whose feature attributions in predictions have significantly changed after the updates. Feature attributions represent the extent to which features contribute to predictions. When a model is retrained using training dataset updated by adding new data, it may find an important feature that was overlooked before the update. The attributions of such features change significantly. In other words, by analyzing significant changes in feature attributions, we can identify model changes in detail. In the proposed method, increasing rates of feature attributions are calculated by comparing attributions before and after the update. As feature attributions, we use Shapley additive explanation (SHAP) values [6], which have a desired property called consistency. Consistency allows for quantitative comparisons of feature attributions between different models.

In our experiments, we use Android application dataset, containing 11,649 benign and 1,430 malicious samples, and build models for detecting malicious samples. We evaluate the effectiveness of the proposed method by analyzing model changes while gradually adding new data to training dataset. The experimental results show that most changes were caused by adding malicious samples. The proposed method also identifies slight model changes that could not be identified based on the area under the curve (AUC). Moreover, we conduct a case study to show that information on identified slight changes is beneficial in terms of MLOps in order to confirm that changes have no negative effect on malware detection.

The remainder of this paper is organized as follows: Section 2 introduces related work and the SHAP concept. Section 3 presents the proposed method. Section 4 shows the experimental

<sup>1</sup> Graduate School of Information Science and Technology, Osaka University

<sup>2</sup> NTT Secure Platform Laboratories

setups and results. Finally, Sections 5 and 6 provide a discussion and our conclusions.

## 2. Related Work

We propose a method for analyzing model changes to confirm that the performance has improved after updates. Before presenting our method, in this section we introduce some other methods for evaluating the appropriateness of models. We also introduce a method for determining features that contribute to classification.

### 2.1 Evaluation Methods

#### 2.1.1 Model Evaluation Metric

For evaluating the classification performance of ML models, there are several common metrics, such as accuracy, precision, recall, F-measure, true positive rate (TPR) and false positive rate (FPR). Those are used to calculate a value indicating the model performance. In binary classification distinguishing between positive and negative classes, samples are divided into 4 different categories based on their predicted and true classes: true positive (TP), true negative (TN), false positive (FP), and false negative (FN). TPs and TNs are samples *correctly* predicted as positive and negative, respectively. FPs and FNs are samples *incorrectly* predicted as positive and negative, respectively. In malware detection, positive and negative samples refer to malicious and benign ones, respectively. The FPs are benign samples which are incorrectly predicted as malicious.

The accuracy simply computes the ratio of correct prediction number to the total sample number:  $\frac{TP+TN}{TP+TN+FP+FN}$ . The precision is the ratio of correct positive prediction number to total positive prediction number:  $\frac{TP}{TP+FP}$ . The recall (also known as the TPR) is the ratio of correct positive prediction number to total positive sample number:  $\frac{TP}{TP+FN}$ . The FPR is the ratio of incorrect positive prediction number to total negative sample number:  $\frac{FP}{FP+TN}$ . The F-measure (or F1-score) is the harmonic mean of precision and recall:

$$2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

The model performance can also be shown in receiver operating characteristic (ROC) curves. ROC curves have the true and false positive rates as vertical and horizontal axes, respectively. ROC curves and the area under the curve (AUC) are commonly used to evaluate the ML model performance in cybersecurity.

Besides those metrics, there are also some criteria to evaluate the model in terms of other perspectives. To avoid overfitting, two well-known criteria, the Akaike information criterion (AIC) [7] and the Bayesian information criterion (BIC) [8], are usually used. They are defined as

$$AIC = -2 \ln(\mathcal{L}) + 2K, \quad (1)$$

$$BIC = -2 \ln(\mathcal{L}) + K \ln(n), \quad (2)$$

where  $K$  is the number of learnable parameters in the model,  $\mathcal{L}$  is the maximum likelihood of the model, and  $n$  is the number of samples.

#### 2.1.2 Cross-validation

Cross-validation evaluates ML models by dividing a dataset

into several subsets. To estimate the model's classification performance, one subset is used for validation and the others are used for training. In  $k$ -fold cross-validation, a dataset  $D$  is randomly split into  $k$  mutually exclusive subsets  $D_1, D_2, \dots, D_k$ . The model is then trained and tested over  $k$  rounds. In each round  $i \in \{1, 2, \dots, k\}$ , training is performed on subset  $D \setminus D_i$  and testing on subset  $D_i$ . In validation, evaluation metrics such as accuracy and AUC score is usually used to estimate classification performance. To reduce variability, the validation results are combined or averaged over all rounds to give a final estimate of classification performance. In stratified cross-validation, subsets are stratified so that they contain approximately the same proportions of labels as the original dataset.

Although those evaluation methods can compute indicators reflecting model performance, they cannot provide sufficient details of model updates.

### 2.2 Feature Attribution Methods

To explain predictions by ML models, importance values are typically attributed to each feature to show its impact on predictions. The importance values of features can be output by some popular ML packages such as scikit-learn [9], where permutation importance are frequently used. Permutation importance randomly permutes the values of a feature in the test dataset and observes change in error. If a feature is important, then permuting it should largely increase model error [10].

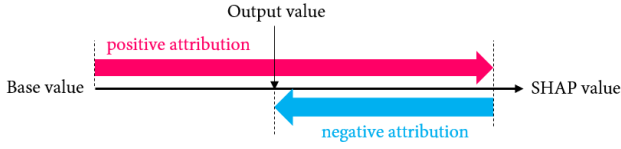
Another method for interpreting ML models is partial dependence plots (PDPs) [11]. A PDP can show how a feature affects model predictions by the relation between the target prediction and feature (e.g., linear, monotonic, or more complex). However, a PDP can compute two features at most, and it assumes those features are not correlated with other features. It is thus unrealistic to use PDP for models trained on data containing numerous features.

Another popular approach called local interpretable model-agnostic explanations (LIME) [12] explains a given prediction by learning a model around that prediction. By computing the feature importance values of a single prediction, we can easily analyze what made the classifier output that prediction. Instead of explaining the whole model, LIME explains only a single sample's prediction result. However, LIME still uses permutation to compute feature importance values, making LIME an inconsistent method.

Although these methods are meant to provide insight into how features affect model predictions, the feature attribution methods described above are all inconsistent, meaning that when the model has changed and a feature impact on the model's output has increased, the importance of that feature can actually be lower. Inconsistency makes comparison of attribution values across models meaningless because it implies that a feature with a large attribution value might be less important than another feature with a smaller attribution.

### 2.3 SHAP

The inconsistency of methods in Section 2.2 makes it meaningless to compare feature attributions across models, which



**Fig. 1** SHAP values explain model output as a sum of the attributions of each feature.

means we need a consistent method to analyze feature attribution changes in different models.

SHAP [6] is a method that explains individual predictions based on Shapley values from game theory. The Shapley value method is represented as an additive feature attribution method (demonstrated in Fig. 1) with a linear explanation model  $g$ , described as

$$g(z) = \phi_0 + \sum_{i=1}^M \phi_i z_i, \quad (3)$$

where  $z \in (0, 1)^M$ ,  $M$  is the number of input features, and  $\phi_i \in \mathbb{R}$ . The  $z_i$  is a binary decision variable to represent a feature being observed or unknown, and  $\phi_i$  are feature attribution values.

Currently, SHAP is the only consistent and locally accurate individualized feature attribution method. According to Ref. [6], SHAP has three desirable properties: local accuracy, missingness, and consistency. Local accuracy means the sum of feature attributions equals the output of the model we want to explain. Missingness means that missing features are attributed no importance, i.e., 0. Consistency means that the attribution assigned to a feature will not be decreased when we change a model so that the feature have a larger impact on the model. Consistency enables comparison of attribution values across models.

When explaining a model  $f$ , SHAP assigns  $\phi_i$  values to each feature [10] as

$$\phi_i = \sum_{S \subseteq A \setminus \{i\}} \frac{|S|!(M - |S| - 1)!}{M!} [f_x(S \cup \{i\}) - f_x(S)], \quad (4)$$

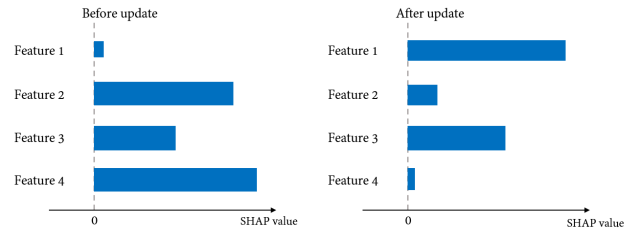
where  $f_x(S) = f(h_x(z)) = E[f(x)|x_S]$ ,  $E[f(x)|x_S]$  is the expected value of a function conditioned on a subset  $S$  of the input features,  $S$  is the set of nonzero indexes in  $z$ , and  $A$  is the set of all input features. The  $h_x$  maps the relation between the pattern of binary features  $z$  and the input vector space.

Since SHAP is the only consistent, locally accurate method for measuring missingness, there is a strong motivation to use SHAP values for feature attribution. However, there are two practical problems remaining to be solved, namely, (1) efficiently estimating  $E[f(x)|x_S]$ , and (2) the exponential complexity of Eq. 4.

When estimating the predictions of tree models, Lundberg and Lee [10] designed a fast SHAP value estimation algorithm specific to trees and tree ensembles. This algorithm runs in polynomial time instead of exponential time, reducing the computational complexity of exact SHAP value computations for trees and tree ensembles.

### 3. Proposed Method

When updating a ML model for real-world deployment, detailed information about model updates is beneficial for prevent-



**Fig. 2** The SHAP values of features change after updates.

ing unexpected predictions in production. To obtain detailed information, we identify samples whose feature attributions significantly changed after the update. Since SHAP is a consistent attribution method, meaning that SHAP values are invariant regardless of models, we use SHAP values to measure the attribution changes of features across different models. We investigate changes in models in detail by analyzing changes in the SHAP values of features.

Figure 2 shows an example of the changes in SHAP values before and after an update regarding predictions of the same sample. A SHAP value is assigned to each feature to show how important it is. A high SHAP value means that the corresponding feature has large effect on the prediction, and a SHAP value close to 0 means that the corresponding feature has almost no effect on the prediction. SHAP values for Features 2 and 4 decreased to near 0, and Feature 1's SHAP value increase greatly from a value near 0 after the update, indicating that the model significantly changed regarding these features. On the other hand, the SHAP values of Feature 3 has no significant change, meaning the model did not change regarding this feature. By analyzing features whose SHAP values have significantly changed, we can infer the cause of model updates and its effect on classification performance.

Our method defines an increasing rate that indicates the significance of changes in feature attributions after a model update. Specifically, we compute SHAP values for different models, then calculate the significance of the increase in each feature's SHAP value due to the update. This increasing rate also shows whether changes in SHAP values are increasing or decreasing. As shown in Fig. 2, Feature 1 has a significant increase, while Feature 2 and 4 have significant decrease after update. Unlike these features, Feature 3's increasing rate is close to 0 because its SHAP value has no significant change after the update.

The following describes our definition of the increasing rate. Let  $D_1$  be the dataset on which the model was trained before the update and let  $D'$  be the data added for the update. After updating, the model will be trained on dataset  $D_2 = D_1 \cup D'$ . Then, let the model as trained on  $D_1$  and  $D_2$  be  $f_1$  and  $f_2$ , respectively. When predicting a label for data  $x$  with model  $f_m$ , we denote the SHAP value of the  $i$ -th feature  $x_i$  as  $v_{mx_i}$ .

We define the increasing rate  $I_{x_i}$  of a feature  $x_i$  as the ratio of the SHAP value's increase to the smallest absolute SHAP value. Let  $v_{1x_i}$  be the SHAP value of feature  $x_i$  in the old model, and let  $v_{2x_i}$  be the SHAP value of feature  $x_i$  in the new model. The increasing rate is large only if the absolute value of one SHAP value ( $v_{1x_i}$  or  $v_{2x_i}$ ) is large and the other is close to zero. In other words, if the absolute values of both SHAP values are either large

**Table 1** Model training data sizes

	Malicious	Benign
Model 1	101	816
Model 2	151	1,224
Model 3	201	1,631
Model 4	251	2,039
Model 5	301	2,447
Model 6	351	2,854
Model 7	401	3,262
Model 8	451	3,670
Model 9	501	4,077

or small, the increasing rate is small. We add constant terms  $c_1$  and  $c_2$  to make the increasing rate small when both SHAP values are close to zero. The increasing rate for feature  $x_i$  is defined as

$$I_{x_i} = \frac{v_{2,x_i} - v_{1,x_i} + c_1}{\min(|v_{1,x_i}|, |v_{2,x_i}|) + c_2},$$

where  $c_2 > 0$ ,

$$c_1 = \begin{cases} c_2, & \text{when } v_{2,x_i} - v_{1,x_i} \geq 0, \\ -c_2, & \text{when } v_{2,x_i} - v_{1,x_i} < 0. \end{cases}$$

In this paper, we set the constant term  $c_2 = 0.01$ .

The SHAP values of a sample  $\mathbf{x}$  is an array of size  $N$ , where  $N$  is the number of features:

$$\mathbf{v}_{m\mathbf{x}} = [v_{mx_1}, v_{mx_2}, \dots, v_{mx_i}, \dots, v_{mx_N}].$$

The increasing rate of a sample is also an array of size  $N$ :

$$\mathbf{I}_{\mathbf{x}} = [I_{x_1}, I_{x_2}, \dots, I_{x_i}, \dots, I_{x_N}].$$

The increasing rate indicates the significance of a change in feature attributions due to a model update. Based on the increasing rate, we identify samples whose feature attributions have significantly changed. To that end, we use a threshold pair  $(k_1, k_2)$  to select the high increasing rate and feature number, where  $k_1$  and  $k_2$  are mutually independent. We identify a change in feature attributions of a sample  $\mathbf{x}$  as significant by counting the number of increasing rates whose absolute values exceed  $k_1$  and where the number of increasing rates is larger than  $k_2$ . By analyzing features whose absolute values of increasing rates are larger than  $k_1$ , we can infer changes in the model caused by the update.

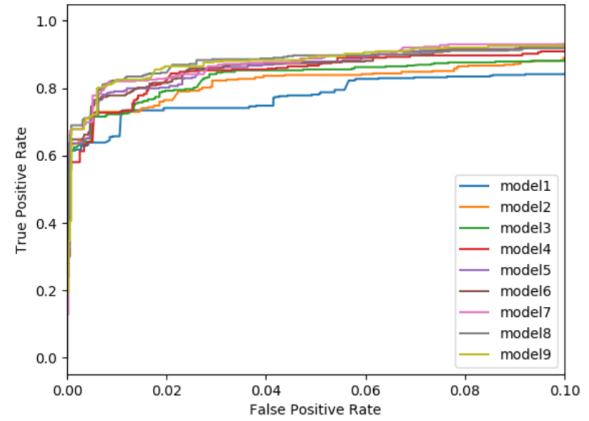
To investigate types of insufficient data, we count the number of significantly changed samples for each type of change. Specifically, we count the number of samples  $N_{I+}$  and  $N_{I-}$  in the shared dataset  $D_1$ , focusing on increasing rates  $I > k_1$  and  $I < -k_1$ , respectively. The more samples the dataset  $D_1$  contains, the larger  $N_{I+}$  and  $N_{I-}$  will be. We thus use ratios  $N_{I+}/|D_1|$  and  $N_{I-}/|D_1|$  to investigate the extent to which a certain data type is insufficient.

## 4. Experiments

In this section, we use Android applications to evaluate the effectiveness of the proposed method. After introducing the dataset and machine learning models used in the experimental setup, we show the experimental results when models are updated by gradually increasing the training data size.

### 4.1 Experimental Setup

**Dataset.** We use samples from AndroZoo [13] to conduct the experiments. AndroZoo is a collection of Android applications



**Fig. 3** ROC curves for models 1–9.

**Table 2** AUC scores for models 1–9

	AUC
Model 1	0.9389
Model 2	0.9588
Model 3	0.9607
Model 4	0.9664
Model 5	0.9695
Model 6	0.9709
Model 7	0.9740
Model 8	0.9735
Model 9	0.9745

from several sources, including the official Google Play app market and VirusShare. It contains over ten million Android application package (APK) files. Each file has been analyzed by over 70 antivirus software packages, providing knowledge of which are malware. We selected files not detected as malware by any antivirus software for use as benign samples. For malicious samples, we selected files that were detected as malware by at least four antivirus software packages.

We collected over 1,000 samples per month from AndroZoo between 2016 and 2018. In total, we gathered 61,724 benign samples and 11,160 malicious samples. Our experiments use applications collected from July to December 2017 to consider the stability of antivirus detection and concept drift. We use applications collected over one year ago because Miller et al. [14] empirically showed that antivirus detections become stable after approximately one year. We minimize the influence of concept drift by using applications collected within six months. We followed Ref. [2] when adjusting the ratio of malicious samples to benign ones. Specifically, we set the percentage of malicious samples to 10% and benign samples to 90% in the dataset. The resulting dataset therefore contains 11,649 benign samples and 1,430 malicious samples.

We split the dataset into training and test datasets. We conduct experiments assuming models are updated by gradually increasing the number of samples in the training dataset. For this experiment, we prepare nine training datasets with different sizes, as shown in **Table 1**. The smallest dataset contains 10% of the randomly selected samples from the adjusted dataset. We prepare the other datasets by repeatedly adding 5% of the remaining adjusted dataset by random sampling.

During the experiments, we noticed that classification performance no longer increases after 50% of the adjusted dataset is used, so we decided to stop increasing the dataset size after 4,077 benign and 501 malicious samples. For the test dataset, we randomly select 30% of the adjusted dataset (3,495 benign and 429 malicious samples). Note that the training and test datasets do not overlap. The test dataset is used to evaluate the classification performance of the models, with all models being evaluated using this test dataset in the experiments.

**Feature.** Before building models, we need to extract features from APK files in order to apply machine learning. To extract features in our experiments, we use Drebin [15], a lightweight method for detecting malicious APK files based on broad static analyses. Features are extracted from the manifest and the disassembled dex code of the APK file. From these, Drebin collects discriminative strings such as permissions, API calls, and network addresses. In particular, Drebin extracts following eight sets of strings: four from manifests and four from dex code.

- (1) Hardware components
- (2) Requested permissions
- (3) App components
- (4) Filtered intents
- (5) Restricted API calls
- (6) Used permissions
- (7) Suspicious API calls
- (8) Network addresses

The features are embedded into an  $N$ -dimensional vector space, where each element is either 0 or 1: Each element corresponds to a string, with 1 representing the presence of the string, and 0 representing its absence. The extracted feature vector  $\mathbf{x}$  is denoted as

$$\mathbf{x} = (\dots \ 0 \ 1 \ \dots \ 0 \ 1 \ \dots).$$

The feature vector can be used as input for a machine learning model.

**Classification Models.** Our experiments use random forest [16], a method that is well known for its excellent classification performance and can be applied to many tasks, including malware detection. Random forest is an ensemble of decision trees. Each decision tree is built using a randomly sampled subset of data and features. By creating an ensemble of many decision trees, random forest achieves high classification performance even when the dimensions of feature vectors exceed the dataset size. Furthermore, the SHAP package [17] associated to Ref. [10] provides a high-speed algorithm called TreeExplainer for tree ensemble methods, including random forest.

**Hyperparameter Optimization.** We use the datasets shown in Table 1 to train the different models. For each random forest model, we conduct a grid search to determine the best combination of parameters among the following candidates:

- (1) Number of trees: 10, 100, 200, 300, 400.
- (2) Maximum depth of each tree: 10, 100, 300, 500.
- (3) Ratio of features used for each tree: 0.02, 0.05, 0.07, 0.1, 0.2.
- (4) Minimum number of samples required at a leaf node: 5, 7,

10, 20.

Each candidate combination is validated using five-fold cross validation. Specifically, we calculate an average of five AUC scores for each combination and select the best combination in terms of average AUC score as the result of the grid search.

**Baseline.** We use AUC scores as the baseline and evaluate whether we can obtain more information with the proposed method than with AUC scores. AUC scores are frequently used to evaluate classification performance of ML models for malware detection. AUC scores can evaluate classification performance by considering true positive rates at various false positive rates.

#### 4.2 Experimental Results

We present the results of three experiments. We first introduce the results of a preliminary experiment for determining the threshold pair in the proposed method. We then compare the proposed method and the baseline when analyzing model updates using the nine models. This section concludes with a case study showing how outputs of the proposed method can be used for further analysis of model updates.

**Preliminary Experiment.** To determine an appropriate threshold pair for the proposed method, we conduct experiments using different threshold pairs and compare the results. Specifically, we calculate the increasing rate using Eq. (5), and count samples containing features with high increasing rate based on different threshold pairs. Each threshold pair contains a threshold  $k_1$ , which defines the minimum absolute value of the increasing rate, and another threshold  $k_2$ , which defines the minimum number of features whose absolute values of increasing rates is larger than  $k_1$ . In other words, samples selected based on a threshold pair  $(k_1, k_2)$  are those containing at least  $k_2$  features whose increasing rates are larger than  $k_1$  or lower than  $-k_1$ . We count such samples for each sign of increasing rate ( $I \geq 0$  for increase and  $I < 0$  for decrease) and each label (malicious and benign). In this experiment, an increase in SHAP values ( $I > 0$ ) means samples are more likely to be detected as malicious, whereas a decrease in SHAP values ( $I < 0$ ) means samples are more likely to be classified as benign.

**Table 3** shows the results when using different threshold pairs. When the thresholds are too high (e.g.,  $k_1 = 5$  or  $k_2 \geq 3$ ), most sample numbers are 0, which are nonsensical results. When the threshold  $k_1$  is too low (e.g.,  $k_1 = 2$ ), the sample numbers are always high, regardless of model updates because increasing rates are larger than 2 even when changes in SHAP values are small. These two are nonsensical results. The threshold pair is appropriate if the proposed method selects only samples whose SHAP values significantly changes without ignoring most samples. We therefore chose the threshold pair (3, 1) when conducting the following experiments.

**Comparison with Baseline.** As baseline results, **Table 2** and **Fig. 3** show the receiver operating characteristic (ROC) curves and AUC scores for each model. Table 2 show that the AUC scores gradually increased and the amount of increase was relatively large for models 1–4 compared with models 4–9. This is because the effect of added data decreased as the training dataset

**Table 3** Number of samples whose increasing rates significantly changed based on different threshold pairs

Threshold pair		(2,1)	(2,3)	(2,5)	(3,1)	(3,3)	(3,5)	(4,1)	(4,3)	(4,5)	(5,1)	(5,3)	(5,5)
Models 1&2	$I \geq 0$	66/115	12/14	5/1	22/38	5/2	0/0	10/14	1/2	0/0	7/6	1/2	0/0
	$I < 0$	75/146	31/9	4/4	56/36	1/2	0/2	24/16	0/2	0/0	6/5	0/2	0/0
Models 2&3	$I \geq 0$	66/77	30/1	8/0	44/10	5/0	5/0	25/3	5/0	5/0	12/1	0/0	0/0
	$I < 0$	97/96	1/0	0/0	12/19	0/0	0/0	1/2	0/0	0/0	1/0	0/0	0/0
Models 3&4	$I \geq 0$	60/115	6/16	2/6	29/46	0/7	0/1	8/17	0/5	0/0	1/10	0/5	0/0
	$I < 0$	24/48	0/6	0/6	0/8	0/5	0/4	0/7	0/5	0/0	0/6	0/5	0/0
Models 4&5	$I \geq 0$	25/33	4/1	0/0	9/16	0/1	0/0	7/11	0/0	0/0	7/3	0/0	0/0
	$I < 0$	8/36	0/1	0/0	0/4	0/0	0/0	0/1	0/0	0/0	0/0	0/0	0/0
Models 5&6	$I \geq 0$	17/61	1/10	0/3	3/26	0/2	0/0	0/8	0/0	0/0	0/1	0/0	0/0
	$I < 0$	18/22	0/2	0/0	0/2	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0
Models 6&7	$I \geq 0$	25/74	4/3	0/2	6/29	4/2	0/0	5/14	4/0	0/0	4/8	4/0	0/0
	$I < 0$	64/61	1/0	0/0	25/8	0/0	0/0	0/3	0/0	0/0	0/3	0/0	0/0
Models 7&8	$I \geq 0$	49/61	1/9	0/0	0/19	0/7	0/0	0/12	0/7	0/0	0/8	0/0	0/0
	$I < 0$	78/46	0/1	0/0	1/9	0/0	0/0	0/4	0/0	0/0	0/1	0/0	0/0
Models 8&9	$I \geq 0$	21/52	0/3	0/0	3/10	0/0	0/0	0/4	0/0	0/0	0/0	0/0	0/0
	$I < 0$	4/12	0/0	0/0	0/1	0/0	0/0	0/1	0/0	0/0	0/0	0/0	0/0

The number before / represents the number of malicious samples, and the number after / represents the number of benign samples.

**Table 4** Number of samples and ratio with threshold pair (3,1)

		Malicious	Benign	Ratio
Models 1 & 2	$I \geq 0$	22	38	0.065
	$I < 0$	56	36	0.100
Models 2 & 3	$I \geq 0$	44	10	0.039
	$I < 0$	12	19	0.023
Models 3 & 4	$I \geq 0$	29	46	0.041
	$I < 0$	0	8	0.004
Models 4 & 5	$I \geq 0$	9	16	0.011
	$I < 0$	0	4	0.002
Models 5 & 6	$I \geq 0$	3	26	0.011
	$I < 0$	0	2	0.001
Models 6 & 7	$I \geq 0$	6	29	0.011
	$I < 0$	25	8	0.010
Models 7 & 8	$I \geq 0$	0	19	0.005
	$I < 0$	1	9	0.003
Models 8 & 9	$I \geq 0$	3	10	0.003
	$I < 0$	0	1	0.000

size increased. Fig. 3 also shows that ROC curves of models 4–9 were close without depending on false positive rates. In contrast, the ROC curves of models 1–4 were not close when the false positive rate is larger than 0.01. These results are useful for knowing the extent to which adding new data improves classification performance. However, we cannot know why classification performance improved.

**Table 4** shows results under the proposed method when using the threshold pair (3, 1). In that table, the number of samples selected by the proposed method drastically decreased after updating model 5. This result is the same as in the baseline. In other words, the effect of adding data decreased as the training dataset size increased, but the proposed method more clearly shows the change in that effect.

Table 4 also shows ratios of selected samples to the number of training dataset samples for each sign of increasing rate ( $I \geq 0$  and  $I < 0$ ). An increase in SHAP values ( $I > 0$ ), meaning samples are more likely to be detected as malicious, is caused by adding malicious data, and a decrease in SHAP values ( $I < 0$ ), meaning samples are more likely to be classified as benign, is caused by adding benign data. Consequently, a high ratio for positive increasing rate indicates that adding malicious data improves classification performance, whereas a high ratio for negative increasing rate indicates that adding benign data improves

performance. Referring to Table 4, the classification performance of models 4–6 is improved mainly after adding malicious data, whereas the performance of models 2, 3, and 7 improved after adding both malicious and benign data. Adding data did not improve the classification performance of models 8 and 9. The proposed method can thus identify why classification performance improved.

Moreover, the proposed method can identify features that contribute to the performance improvement by updates, namely, those with increasing rates exceeding the threshold  $k_1$ . The following describes some important features of multiple samples in **Table 5**. For reference, we give at least three features in each update, although the total number of features was less than three for updates to models 6 and models 8–9. These features and their increasing rates demonstrate importance changes for classification. For example, `android.app.activitymanager.get_running_tasks` becomes important when adding benign data. If these features are associated with a certain malware family, we can obtain even more information about the dataset. For example, `android.media.ringtonemanager.set_actual_default_ringtone_uri` becomes important when adding samples of the “tachi” family.

**Case Study.** Table 4 shows an interesting result regarding the update between models 6 and 7: the ratio of negative increasing rates is significantly larger than those for updates of models 3–6 and 7–9. This indicates an unusual change between models 6 and 7, so we conduct a more detailed analysis as a case study.

We first focus on the most distinct samples, namely, malicious samples with negative increasing rate. Analyzing their features, we find that 24 of the 25 samples contain both or one of the following features:

- (1) `android.permission.vibrate`
- (2) `android.permission.write_external_storage`

The update significantly changed the SHAP values of these features, the former decreasing from 0.2134 to 0.0377 and the latter decreasing from 0.0924 to 0.0171. These changes show that features become much less important for classification after the update between models 6 and 7. Further analysis focusing on mal-

**Table 5** Features with increasing rates exceeding the threshold  $k_1 = 3$

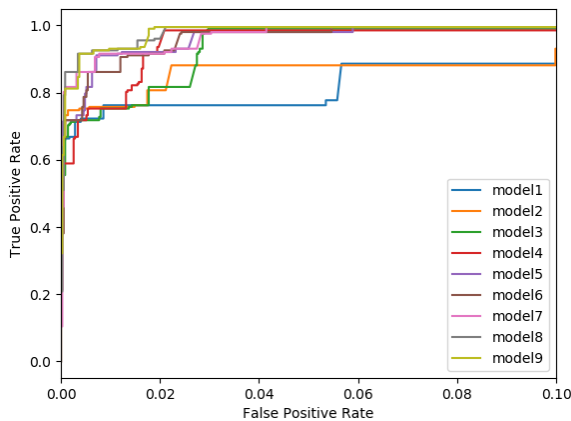
	Feature	$I$	Family	Number
Models 1 & 2	android.app.activitymanager:get_running_tasks	$I < 0$	*	23
	android.media.ringtonemanager:set_actual_default_ringtone_uri	$I < 0$	tachi	13
	android.nfc.tech:NDE_formatable:format	$I < 0$	*	13
Models 2 & 3	android.nfc.tech:Ndef_formatable:format	$I > 0$	*	20
	android.media.ringtonemanager:set_actual_default_ringtone_uri	$I > 0$	tachi	17
	android.permission:change_wifi_state	$I < 0$	piom	5
Models 3 & 4	android.locationmanager:get_provider	$I > 0$	*	18
	android.permission:send_sms	$I > 0$	*	6
	servicelist:com.stub.stub05.stub02	$I > 0$	jiagu	5
Models 4 & 5	servicelist:com.stub.stub02.stub04	$I > 0$	jiagu	6
	android.launcher.permission:read_settings	$I > 0$	*	2
	servicelist:com.stub.stub01.stub01	$I > 0$	drtycow	1
Models 5 & 6	Ndef formatable:connect	$I > 0$	*	2
	android.provider.settings\$system:put_string	$I > 0$	gappusin	1
Models 6 & 7	android.permission:write_external_storage	$I < 0$	fakeapp	24
	android.permission:vibrate	$I < 0$	fakeapp	21
	servicelist:com.stub.plugin.stub03	$I > 0$	jiagu	4
Models 7 & 8	android.telephony.telephonymanager:getline1number	$I < 0$	*	1
Models 8 & 9	android.permission:read_user_dictionary	$I > 0$	*	3

Feature: Features extracted from Android applications.

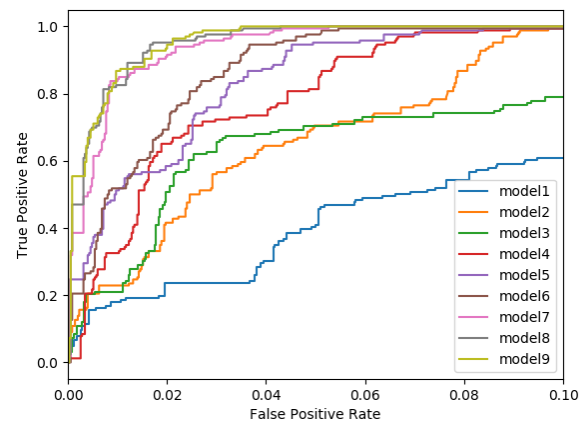
Number: Number of samples containing the feature.

Family: Family to which the samples belong.

\*Sample cannot be associated with a certain family, or is associated with multiple families.



**Fig. 4** ROC curves of the “fakeapp” family for models 1–9.



**Fig. 5** ROC curves of the “jiagu” family for models 1–9.

ware families showed that these features are associated with the “fakeapp” family, which might become more difficult to detect by these features. In other words, the update between models 6 and 7 might decrease true positive rates for the “fakeapp” family. We therefore investigate ROC curves for the “fakeapp” family. **Figure 4** shows that the ROC curve does not degrade after the update between models 6 and 7. This analysis confirms that there is no negative effect on classification performance by these features.

Next, we focus on malicious samples with positive increasing rates. Four of the six samples contain the following features:

- (1) com.stub.plugin.stub03
- (2) com.stub.plugin.stub02
- (3) com.stub.plugin.stub01

SHAP values of all these features increased from 0.000 to at least 0.1383, indicating that they are not important for model 6 but become important when detecting malicious samples for model 7. Given that these features are associated with the “jiagu” family, it may become easier to detect after the update between models 6 and 7. **Figure 5** shows the ROC curves of the “jiagu” family. The true positive rate for model 7 is much higher than that for model 6, with false positive rates from 0.00 to 0.04. This analysis showed

that classification performance for the “jiagu” family is improved by the three features above after the update between models 6 and 7, despite the change in AUC being small (see Table 2).

As this case study showed, the proposed method is useful to find unusual changes in models, identify their cause, and estimate their effects even if the change only slightly affects AUC scores.

## 5. Discussion

**Application of the Proposed Method.** Our method can be applied to machine learning tasks other than malware detection. The SHAP method provides algorithms for estimating SHAP values for any model, allowing application to any ML model regardless of dataset or model. For example, our method can be applied to suspicious URL detection [18], malicious website detection [19], and malware family classification [20]. In multiclass classifications, we can identify changes in important features by analyzing feature attribution changes while focusing on each class.

**Dataset Bias.** Our method measures the extent of change in models by counting samples in which important classification features have changed. Therefore, when the dataset is biased toward just

a few families, leaving other families only a few samples, it is difficult to count many samples of nonbiased families even when their features change significantly. However, as demonstrated in Section 4, we can still manually analyze the details of changed features to determine their influence on families containing few samples.

**Future Work.** The experiments described above performed only malware detection for Android applications using a random forest classification. Although the increasing rate can theoretically be computed for any model, we still need to perform further experiments on different models and datasets to confirm some details, such as how to best choose the threshold pair.

We currently analyze model changes mainly by counting the number of samples, but in practice the extent of value change can also be useful to understand model updates. In future work, we will try to obtain more detailed model information through changes in feature attributions.

## 6. Conclusion

ML methods have been widely applied to many tasks. In practical use, it is necessary to regularly update the model to maintain its classification performance. AUC and accuracy are generally used to validate models to confirm their performance after updates. However, it is difficult to gain sufficiently detailed information for understanding model updates, such as what causes performance improvements and the slight changes in models that affect predictions for a small amount of data.

We therefore proposed a method for determining samples in which the features important for classification have significant changes. By analyzing those samples and features, we can know more about why performance improved or how an update influences a particular malware family. For the feature contribution computation, we used a consistent importance value called the SHAP value, due to property that SHAP values are comparable across different models. Our proposed method calculates increasing rates of SHAP values after updates to reflect changes in feature importance. We conducted experiments demonstrating that the causes of performance changes by model updates can be identified with the proposed method. Through a case study, we manually analyzed the model update based on the output of the proposed method, finding that it can distinguish slight changes for a particular malware family. Currently, we have only analyzed the results by selecting high increasing rate features, without studying specific rate values. In future work, we will gather more information and conduct qualitative analyses from the results of our method.

## References

[1] Google Cloud: MLOps: Continuous delivery and automation pipelines in machine learning (2020). <https://cloud.google.com/solutions/machine-learning/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning>.

[2] Pendlebury, F., Pierazzi, F., Jordaney, R., Kinder, J. and Cavallaro, L.: TESSERACT: Eliminating experimental bias in malware classification across space and time, *Proceedings of the 28th USENIX Security Symposium*, pp. 729–746 (2019).

[3] Jordaney, R., Sharad, K., Dash, S. K., Wang, Z., Papini, D., Nouruddin, I. and Cavallaro, L.: Transcend: Detecting concept drift in malware classification models, *Proceedings of the 26th USENIX Security Symposium*, pp. 625–642 (2017).

[4] Tsymbal, A.: The problem of concept drift: definitions and related work, *Computer Science Department, Trinity College Dublin*, Vol. 106, No. 2, p. 58 (2004).

[5] Karlaš, B., Interlandi, M., Renggli, C., Wu, W., Zhang, C., Mukunthu Iyappan Babu, D., Edwards, J., Lauren, C., Xu, A. and Weimer, M.: Building Continuous Integration Services for Machine Learning, *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2407–2415 (2020).

[6] Lundberg, S. M. and Lee, S.-I.: A unified approach to interpreting model predictions, *Proceedings of the 31st Advances in Neural Information Processing Systems*, pp. 4765–4774 (2017).

[7] Akaike, H.: Information theory and an extension of the maximum likelihood principle, *Selected Papers of Hirotugu Akaike*, pp. 199–213 (1998).

[8] Schwarz, G.: Estimating the dimension of a model, *The Annals of Statistics*, Vol. 6, No. 2, pp. 461–464 (1978).

[9] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R. and Dubourg, V.: Scikit-learn: Machine learning in Python, *Journal of Machine Learning Research*, Vol. 12, pp. 2825–2830 (2011).

[10] Lundberg, S. M., Erion, G. G. and Lee, S.-I.: Consistent individualized feature attribution for tree ensembles, *arXiv preprint arXiv:1802.03888* (2018).

[11] Friedman, J. H. and Meulman, J. J.: Multiple additive regression trees with application in epidemiology, *Statistics in Medicine*, Vol. 22, No. 9, pp. 1365–1381 (2003).

[12] Ribeiro, M. T., Singh, S. and Guestrin, C.: "Why Should I Trust You?": Explaining the Predictions of Any Classifier, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1135–1144 (2016).

[13] Allix, K., Bissyandé, T. F., Klein, J. and Le Traon, Y.: Androzoo: Collecting millions of android apps for the research community, *Proceedings of the 13th IEEE/ACM Working Conference on Mining Software Repositories*, pp. 468–471 (2016).

[14] Miller, B., Kantchelian, A., Tschantz, M. C., Afroz, S., Bachwani, R., Faizullahbhoj, R., Huang, L., Shankar, V., Wu, T. and Yiu, G.: Reviewer integration and performance measurement for malware detection, *Proceedings of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 122–141 (2016).

[15] Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K. and Siemens, C.: Drebin: Effective and explainable detection of android malware in your pocket., *Proceedings of the 2014 Network and Distributed System Security Symposium*, Vol. 14, pp. 23–26 (2014).

[16] Breiman, L.: Random forests, *Machine Learning*, Vol. 45, No. 1, pp. 5–32 (2001).

[17] Lundberg, S. M., Erion, G., Chen, H., DeGrave, A., Prutkin, J. M., Nair, B., Katz, R., Himmelfarb, J., Bansal, N. and Lee, S.-I.: From local explanations to global understanding with explainable AI for trees, *Nature Machine Intelligence*, Vol. 2, No. 1, pp. 2522–5839 (2020).

[18] Ma, J., Saul, L. K., Savage, S. and Voelker, G. M.: Beyond blacklists: learning to detect malicious web sites from suspicious URLs, *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1245–1254 (2009).

[19] Canali, D., Cova, M., Vigna, G. and Kruegel, C.: Prophilier: a fast filter for the large-scale detection of malicious web pages, *Proceedings of the 20th International Conference on World Wide Web*, pp. 197–206 (2011).

[20] Ahmadi, M., Ulyanov, D., Semenov, S., Trofimov, M. and Giacinto, G.: Novel feature extraction, selection and fusion for effective malware family classification, *Proceedings of the 6th ACM Conference on Data and Application Security and Privacy*, pp. 183–194 (2016).

[21] Burnham, K. P. and Anderson, D. R.: Multimodel inference: understanding AIC and BIC in model selection, *Sociological Methods & Research*, Vol. 33, No. 2, pp. 261–304 (2004).