Understanding Machine Learning Model Updates Based on Changes in Feature Attributions

YUN FAN,
Graduate School of Information Science and Technology,
Osaka University

Contents

Background

- In cybersecurity, Machine learning (ML) has been applied to many systems such as malware detection
- ML performance degrades when statistical characteristics of data change over time → concept drift
- ML models need **updates** to improve the performance
  - update: add new data to the training dataset and re-train the model
- After updates, the new model needs to be **validated**
  - accuracy
  - the area under the curve (AUC)
  - …

Research Purpose

- Common validation methods only calculate accuracy or AUC scores of ML models
    - *why performance improved ?*
    - *what changes in the update affect performance?*

Obtain **detailed information** to understand the model updates

- ➢ What causes the performance changes
- ➢ Whether there are slight changes not showing in the accuracy and AUC scores

Related Works

- **Importance values** are typically used to explain ML models
  - Permutation importance, Local Interpretable Model agnostic Explanations (LIME), etc.
- **Inconsistency**: When the model has changed and a feature has higher impact on the model, the importance of that feature can actually be lower.
- Inconsistency make comparison between different models meaningless
  - Only comparison between different features in the same model is meaningful

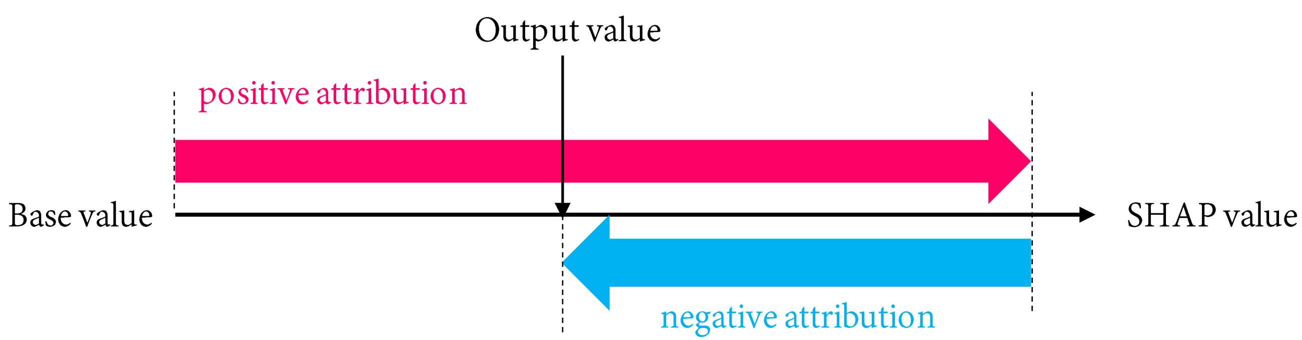
- A **consistent** feature attribution method is necessary
  - **Shapley additive explanations (SHAP)**

SHAP

- Shapley additive explanations (SHAP) is a **consistent** feature attribution method
- SHAP explains the output as a sum of the effects of each feature

  (M: feature number, φ_i: feature attribution value, z_i: binary variable to represent a feature being observed or unknown)

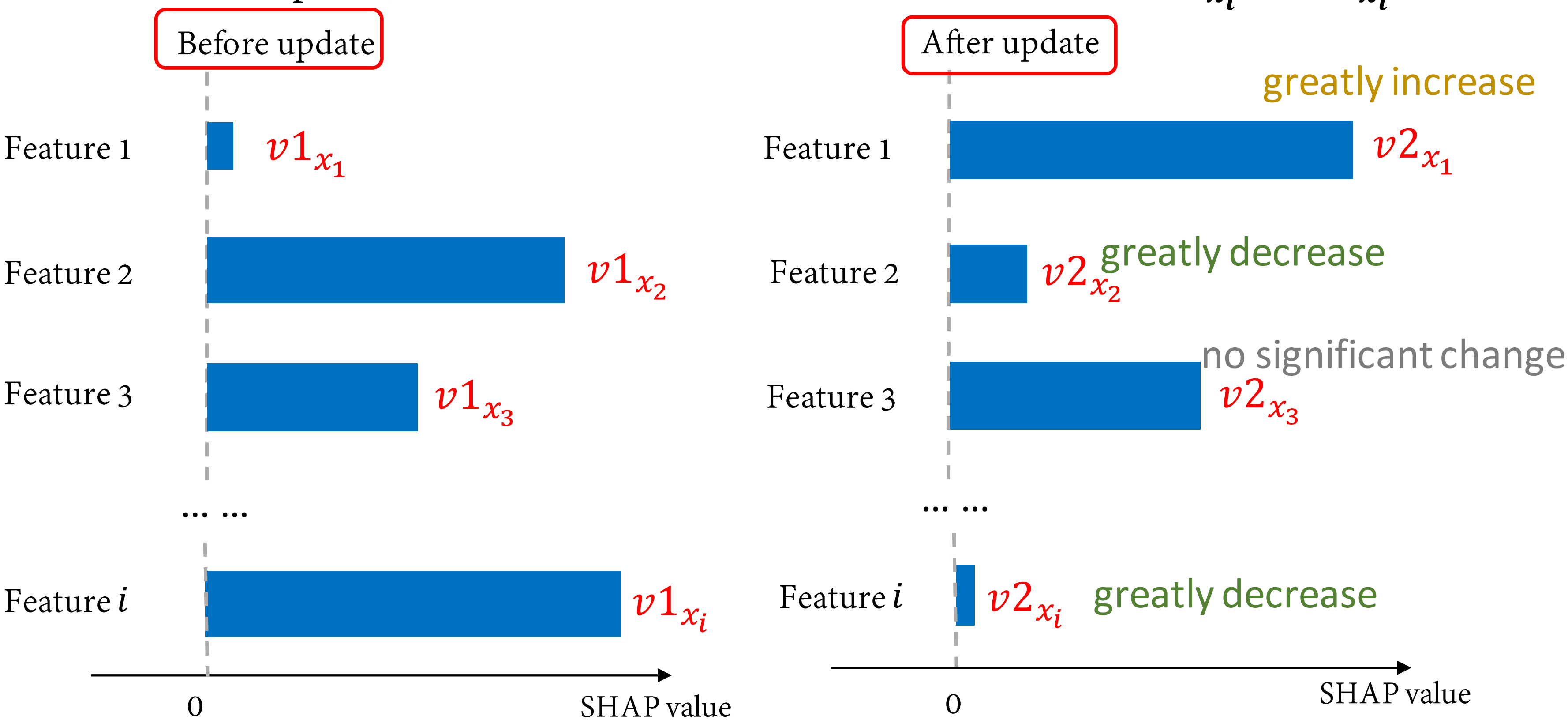
$$g(z) = \varphi_0 + \sum_{i=1}^{M} \varphi_i z_i$$



- **Consistency** enables comparison of attribution values across models

SHAP Values Change

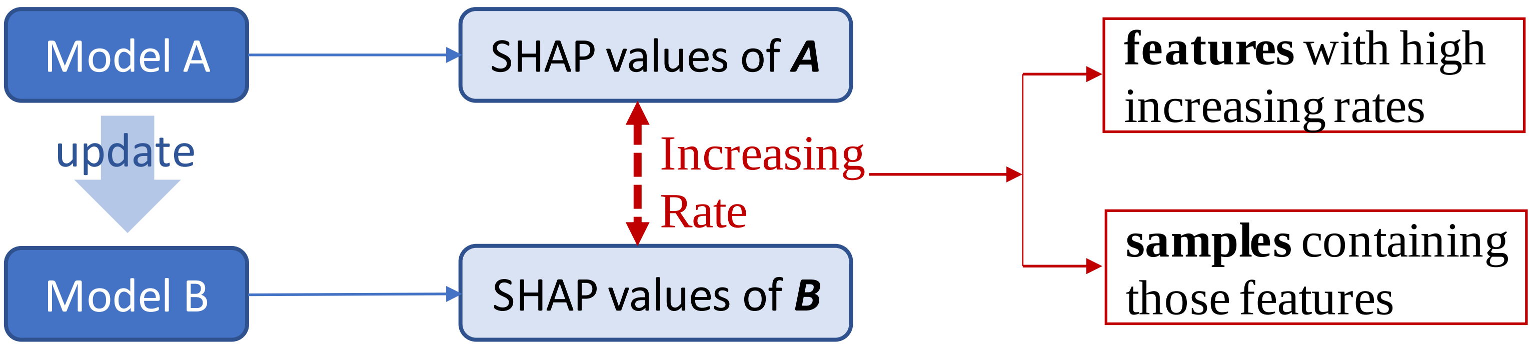
- We can explain the performance changes by measuring the feature contribution (SHAP values) changes
- For a sample $\boldsymbol{x}$, we denote each feature SHAP value as $\boldsymbol{v1}_{\boldsymbol{x}_i}$ or $\boldsymbol{v2}_{\boldsymbol{x}_i}$



Proposed Method

- Since SHAP is a consistent attribution method, we use SHAP values to measure the attribution changes over model updates

- Proposed method



- By identifying the features and sample number, we can analyze what changes affect the performance during updates

Increasing Rate

- The SHAP values of a sample $\boldsymbol{x}$ is:

$$v_{\boldsymbol{x}} = \left[v_{x_1}, v_{x_2}, v_{x_3}, \dots, v_{x_i}, \dots\right]$$

- Define **increasing rate** of feature $\boldsymbol{i}$ in sample $\boldsymbol{x}$:

$$I_{x_i} = \frac{v2_{x_i} - v1_{x_i} + c_1}{\min\left(\left|v1_{x_i}\right|, \left|v2_{x_i}\right|\right) + c_2},$$

$$where\ c_2 > 0, c_1 = \begin{cases} c_2, & when\ v2_{x_i} - v1_{x_i} \geq 0, \\ -c_2, & when v2_{x_i} - v1_{x_i} < 0. \end{cases}$$

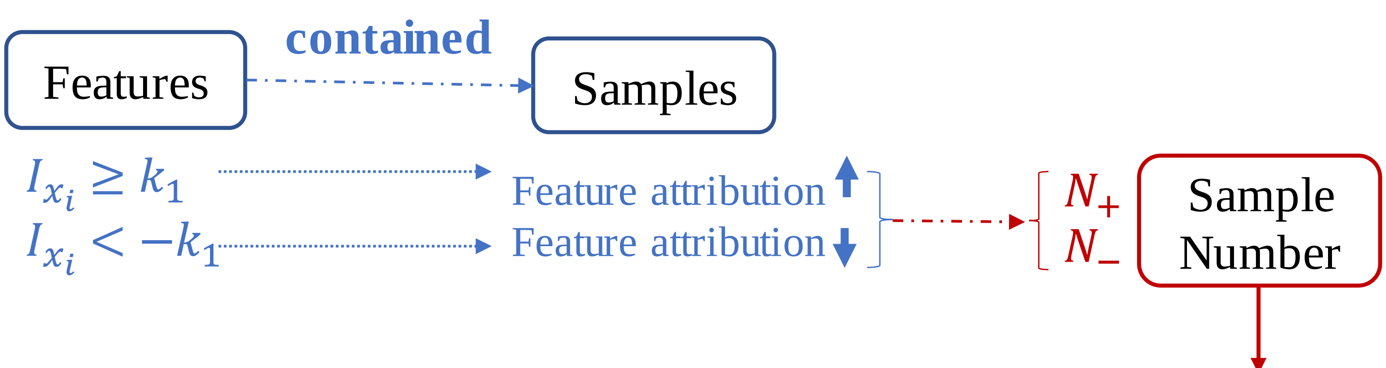
(c_1 , c_2: constant terms to make the I_{x_i} small when both SHAP values are close to zero)

- The increasing rate of a sample $\boldsymbol{x}$ is:

$$I_{\boldsymbol{x}} = \left[I_{x_1}, I_{x_2}, I_{x_3}, \dots, I_{x_i}, \dots\right]$$

Samples Number

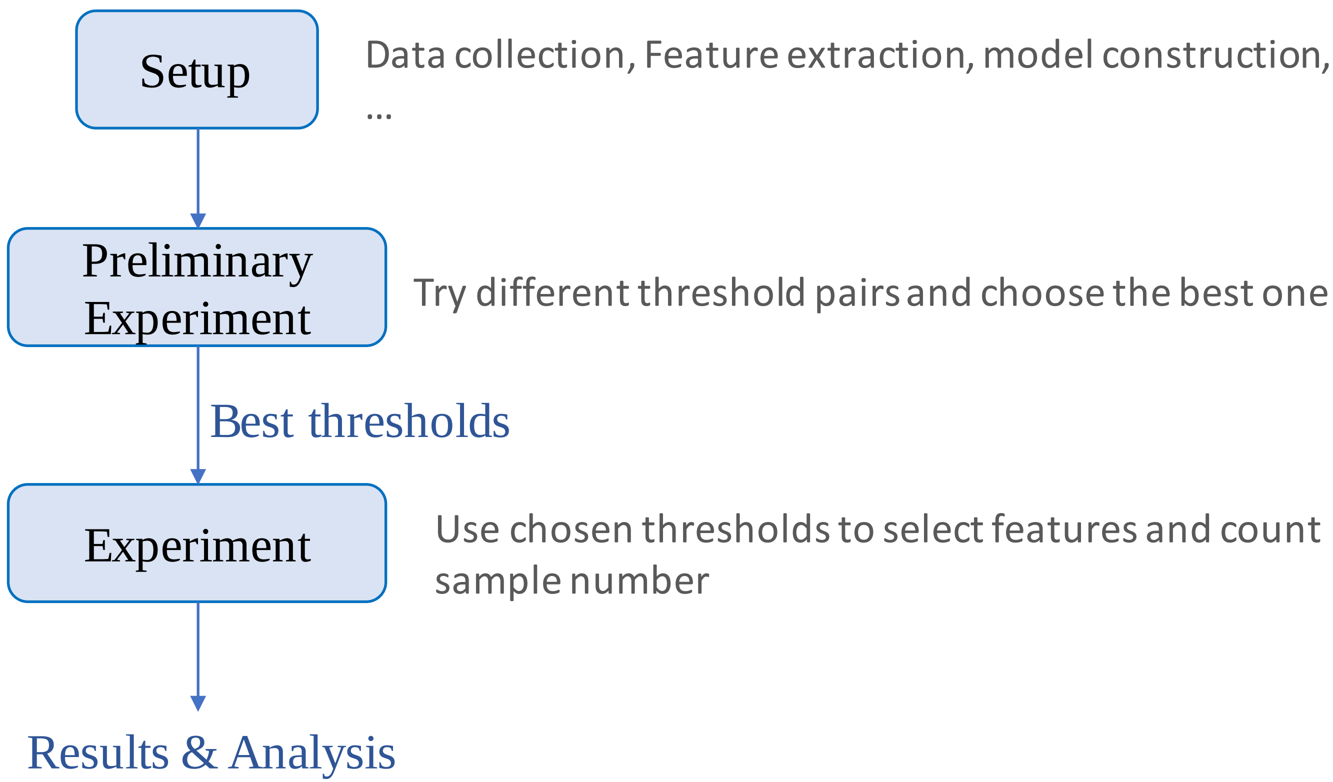
- We select samples whose feature attributions have significantly changed using threshold pair: $(\boldsymbol{k_1}, \boldsymbol{k_2})$
  - If $|I_{x_i}| \geq \boldsymbol{k_1}$, the feature's increasing rate is denoted as high
  - If the number of $|I_{x_i}| \geq k_1$ in sample $\boldsymbol{x}$ is larger than $\boldsymbol{k_2}$, $\boldsymbol{x}$ is selected



<u>We use the number of samples whose feature contributions have significant changes to analyze the model updates</u>

Experiments

We use Android applications to evaluate the effectiveness of the proposed method.



Experimental Setup

- Dataset
  - Android application files: *AndroZoo**
  - 9 dataset with different size (containing 10% malicious samples)
- Features: *Drebin**
  - extracted from the manifest and the disassembled dex code
  - embedded into an N-dimensional vector space
- Classification Models: Random Forest
  - use grid search and cross-validation to choose hyperparameters

	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

**AndroZoo: Allix, K, etc.: Androzoo: Collecting millions of android apps for the research community.(2016)*
**Drebin: Arp, D., etc.: Drebin: Effective and explainable detection of android malware in your pocket.(2014)*

Preliminary Experiment

- Different threshold pairs and their corresponding sample numbers selected by the proposed method:

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

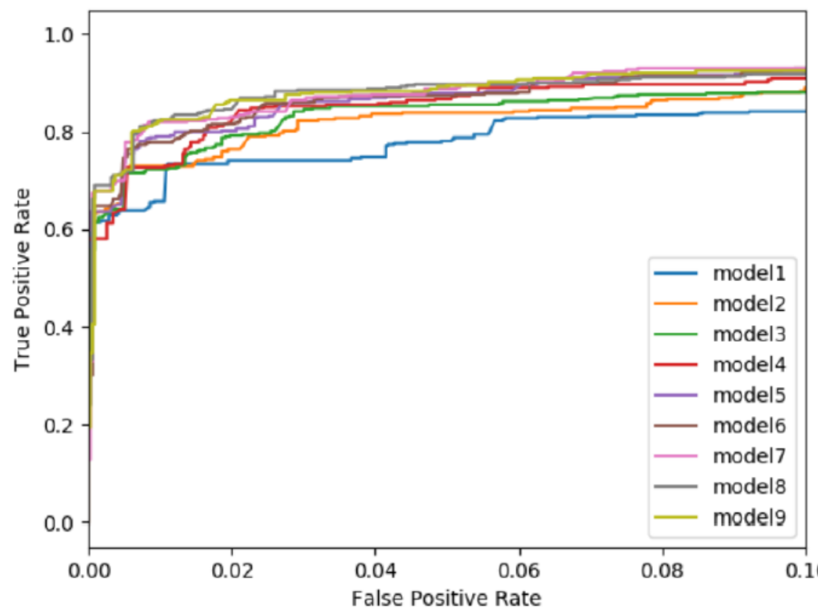
The sample number was counted by $I_{x_i} \geq k_1$ and $I_{x_i} < -k_1$ respectively and noted in malicious/benign

- We choose (3,1) to conduct the experiment

Baseline

- We use ROC curves and AUC scores as the baseline to evaluate whether the proposed method can provide more information

- The extent of improvement in AUC is decreasing as the update going on



	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

Experimental Results

- The improvement by adding data decreased as dataset growing and became small after Model 4&5 → similar to the baseline
- The proposed method can explain how new data affected performance change → the improvement was mainly caused by adding malicious data

more likely to be detected as malicious (caused by adding malicious data)

more likely to be detected as benign (caused by adding benign data)

		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

	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

Feature Details

- The proposed method can identify features that contribute to the performance improvement by updates

	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

Case Study

- The ratio of negative increasing rates is large between models 6 and 7

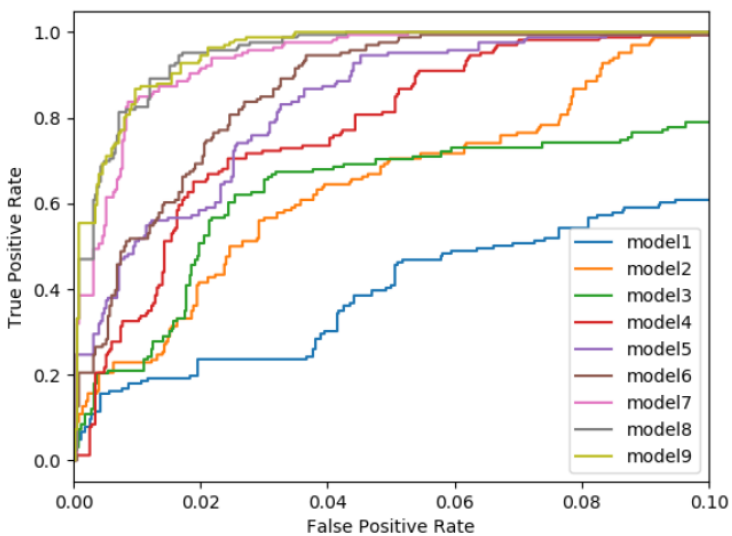
		Malicious	Benign	Ratio
Models 1 & 2	$I \geq 0$	22	38	0.065
	$I < 0$	56	36	0.100
Models 2 & 3	$I \geq 0$	44	10	0.039
	$I < 0$	12	19	0.023
Models 3 & 4	$I \geq 0$	29	46	0.041
	$I < 0$	0	8	0.004
Models 4 & 5	$I \geq 0$	9	16	0.011
	$I < 0$	0	4	0.002
Models 5 & 6	$I \geq 0$	3	26	0.011
	$I < 0$	0	2	0.001
Models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

- 4 of the 6 samples contain the following features:
  - com.stub.plugin.stub03
  - com.stub.plugin.stub02
  - com.stub.plugin.stub01
- these features are associated with the "**jiagu**" family

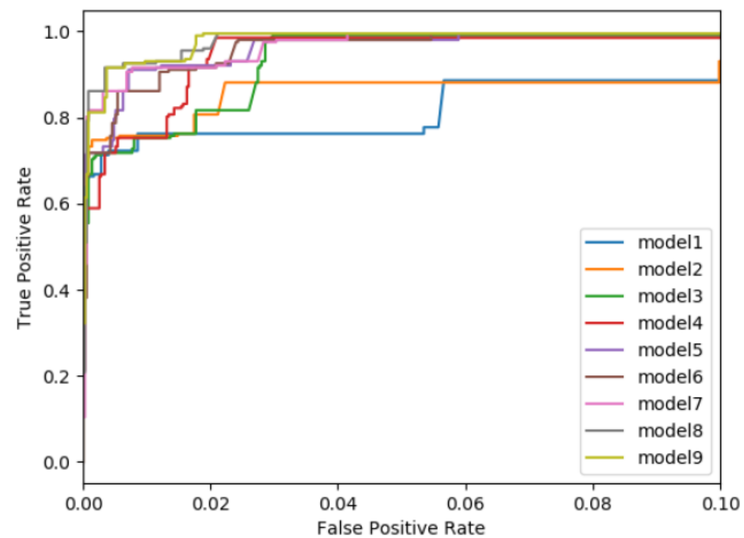
- 24 of the 25 samples contain both or one of the following features:
  - *android.permission.vibrate*
  - *android.permission: write external storage*
- these features are associated with the "**fakeapp**" family

Case Study: Malware Family

➢ Specifically draw the ROC for "jiagu" and "fakeapp" family



ROC of "jiagu"



ROC of "fakeapp"

- Performance on "jiagu" has improved ⟶ not shown in AUC scores
- "fakeapp" has no negative effect on classification performance

Conclusion

- The causes of performance changes by model updates can be identified with the proposed method
  - how much improvement the update has achieved
  - whether the changes are caused by the malicious or benign data
  - what prediction (positive or negative) the updated model tend to make

- The proposed method can analyze the effects to updates of adding malicious and benign samples respectively

- The proposed method can distinguish slight changes for a particular malware family

Discussion

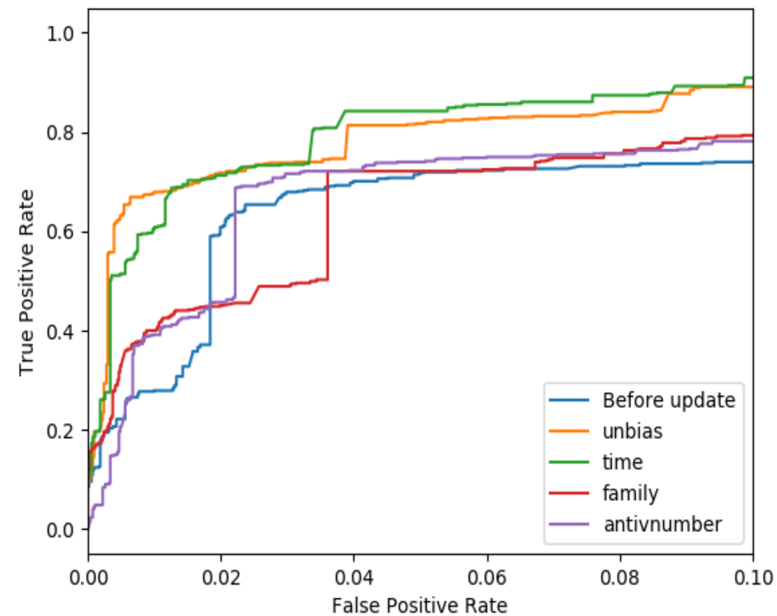
- Application
  - malicious website detection
  - malware family classification

- Future works
  - experiments on other ML models and datasets
  - analysis about data sufficiency
  - better solution for best choosing threshold pair

Update with Biased Data

- Dataset
  - **Unbias**: use random date from all time averagely
  - **Time**: only use the latest data
  - **Family**: only use malware from major families
  - **Antivirus**: only use malware that can be detected by most antivirus software

ROC curves



- Identify features by the average impact of SHAP values changes

(I is the increasing rate, S is the size of the dataset, and k is the threshold)

$$\boxed{\frac{\sum_{I \geq k} I}{S}}$$

Experimental Results

- The ROCs of “unbias” and “time” are better than others and the features are similar
- The ROC of “family” has fell and the identified features are all related to “*com.qihoo.util*”.
- The result of “antivirus” is different from others

Unbias

videoview.setvideopath	1.59
videoview.stopplayback	0.99
videoview.pause	0.88
videoview.start	0.72

Time

videoview.setvideopath	1.27
videoview.pause	1.22
videoview.start	1.1
videoview.stopplayback	1.06

Family

com.qihoo.util.commonactivity	1.0
com.qihoo.util.updateservice	0.74
com.qihoo.util.commonprovider	0.73
com.qihoo.util.commonservice	0.69

Antivirus

permission.get accounts	0.93
permission.read sms	0.74
permission.write sms	0.53

Update with Biased Data

- Dataset
  - **Biased in malware family**: only use malware from major families
- Identify important features by the average impact of SHAP values changes: $\frac{\sum_{I \geq k} I}{S}$

  (I is the increasing rate, S is the size of the dataset, and k is the threshold)
- The identified features are all related to *"com.qihoo.util"*, caused by the bias of dataset.

Family	
com.qihoo.util.commonactivity	1.0
com.qihoo.util.updateservice	0.74
com.qihoo.util.commonprovider	0.73
com.qihoo.util.commonservice	0.69