Detecting Concept Drift in Malware Classification Models

Roberto Jordaney+, Kumar Sharad§, Santanu Kumar Dash‡, Zhi Wang†, Davide Papini•, Ilia Nouretdinov+, Lorenzo Cavallaro+

USENIX Security Symposium
Thu Aug 17, 2017

+Royal Holloway, University of London
§NEC Laboratories Europe
‡University College London
†Nankai University
•Elettronica S.p.A.
Usually, a 2-phase process:

1. Training: build a model $M$, given labeled objects
2. Testing: given $M$, predict the labels of unknown objects

Objects are described as vectors of features
Usually, a 2-phase process:

1. Training: build a model $M$, given labeled objects
2. Testing: given $M$, predict the labels of unknown objects

Objects are described as vectors of features
Machine Learning Classification

**Usually, a 2-phase process:**

1. **Training:** build a model $M$, given labeled objects
2. **Testing:** given $M$, predict the labels of unknown objects

Objects are described as vectors of features
• **Concept drift** is the **change in the statistical properties** of an object in unforeseen ways

• Drifted objects will likely be wrongly classified
• **Concept drift** is the change in the statistical properties of an object in unforeseen ways

• Drifted objects will likely be wrongly classified
Machine Learning Classification Problem: Concept Drift

- *Concept drift* is the change in the statistical properties of an object in unforeseen ways.
- Drifted objects will likely be wrongly classified.

Of course, the problem exists in multiclass classification settings...
Multiclass classification is a generalization of the binary case.
Concept Drift

- In *non-stationary* contexts classifiers will suffer from concept drift due to:
  - malware evolution
  - new malware families
- Need a way to **assess the predictions** of classifiers
  - Ideally classifier-agnostic assessments
- Need to identify objects that fit a model and those drifting away
Concept Drift

- In *non-stationary* contexts classifiers will suffer from concept drift due to:
  - malware evolution
  - new malware families
- Need a way to **assess the predictions** of classifiers
  - Ideally classifier-agnostic assessments
- Need to identify objects that fit a model and those drifting away

**Our Contributions**

- Conformal Evaluator: statistical evaluation of ML classifiers
- Per-class quality threshold to identify reliable and unreliable predictions
Conformal Evaluator
Conformal Evaluator

- Assesses decisions made by a classifier
  - Mark each decision as **reliable** or **unreliable**
- Builds and makes use of p-value as assessment criteria
- Computes **per-class thresholds** to divide reliable decisions from unreliable ones
Conformal Evaluator: P-value?

- Used to measure “how well” a sample fits into a single class
- Conformal Evaluator computes a p-value for each class, for each test element

**Definition**

\[ \alpha_t = \text{Non-conformity score for test element } t \]

\[ \forall i \in \mathcal{K}, \alpha_i = \text{Non-conformity score for train element } i \]

\[ \text{p-value} = \frac{|\{i : \alpha_i \geq \alpha_t\}|}{|\mathcal{K}|} \]

\[ \mathcal{K} = \text{Total number of element} \]

**P-value**

Ratio between the number of training elements that are more dissimilar than the element under test
1. Setting: 3-class classification
1. Setting: 3-class classification
2. Test object
Conformal Evaluator: P-value Example

ML classifier:
distance from centroid

1. Setting: 3-class classification
2. Test object
   3.1 Compute distance to blue class

9
P-value $\star = \frac{9}{10}$
Conformal Evaluator: P-value Example

ML classifier:
\textit{distance from centroid}

1. Setting: 3-class classification
2. Test object
   3.1 Compute distance to blue class
   3.2 How many objects are more dissimilar than the one under test?
Conformal Evaluator: P-value Example

ML classifier: distance from centroid

1. Setting: 3-class classification
2. Test object
   3.1 Compute distance to blue class
   3.2 How many objects are more dissimilar than the one under test?
   3.3 9

\[ P-value = \frac{9}{10} \]
1. Setting: 3-class classification
2. Test object
   3.1 Compute distance to blue class
   3.2 How many objects are more dissimilar than the one under test?
   3.3 9
   3.4 P-value\(\star = \frac{9}{10}\)
1. Initial situation: three classes
2. Test object
   4.1 Calculate distance to green class
   4.2 How many objects are more dissimilar than the one under test?
   4.3 4
   4.4 P-value⋆ = \frac{4}{12}
1. Initial situation: three classes
2. Test object
   5.1 Calculate distance to red class
   5.2 How many objects are more dissimilar than the one under test?
   5.3 0
   5.4 P-value\(\star = \frac{0}{11}\)
Conformal Evaluator: P-value Example

Let’s see how p-values are used within Conformal Evaluator.

1. Initial situation: three classes
2. Test object
   5.1 Calculate distance to red class
   5.2 How many objects are more dissimilar than the one under test?
   5.3 0
   5.4 P-value \[ \star = \frac{0}{11} \]
Conformal Evaluator: How Does it Work?

1. Extracts the **non-conformity measure** (NCM) from the decision making algorithm
   - NCM provides non-conformity scores for p-value computations
   - Example: distance from hyperplane, Random Forest probability (adapted to satisfy the non-conformity requirement)

Decision algorithm | Non-conformity measure | Threshold analysis

| 1 | 2 | 3 |

Threshold for class A
Threshold for class B

Training dataset
Conformal Evaluator: How Does it Work?

1. Extracts the **non-conformity measure** (NCM) from the decision making algorithm
2. Builds p-values for all training samples in a **cross-validation** fashion

![Diagram](image_url)

- **Decision algorithm**
- **Non-conformity measure**
- **Threshold analysis**

<table>
<thead>
<tr>
<th>Training dataset</th>
<th>Decision algorithm</th>
<th>Non-conformity measure</th>
<th>Threshold analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Threshold for class A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Threshold for class B</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>. . . . . . . . . . .</td>
</tr>
</tbody>
</table>
Conformal Evaluator: How Does it Work?

1. Extracts the **non-conformity measure** (NCM) from the decision making algorithm
2. Builds p-values for all training samples in a **cross-validation** fashion
3. Computes **per-class threshold** to divide reliable predictions from unreliable ones
Customizable constraints:

- Desired performance (of the predictions marked as reliable)
  - E.g.: high-level performance will raise the threshold
- Number of unreliable prediction tolerated
  - E.g.: low number of unreliable prediction will lower the threshold

Assumptions

- Performance of non-drifted elements are similar to the one declared by the algorithm
- Predictions with high confidence will have higher p-values
Conformal Evaluator: Identifying per-class Thresholds

- We use the p-values and prediction labels from training samples.
- From the thresholds that satisfy the constraints we chose the one that maximize one or the other.

![Diagram showing correct and incorrect decisions with p-values and identified threshold.](image-url)
Experimental Results
Experimental Results: Case Studies

- **Binary case study: Android malware detection algorithm**
  - Reimplemented Drebin\(^1\) algorithm with similar results
    (0.95-0.92 precision-recall on malicious apps and 0.99-0.99 precision-recall on benign apps)
  - Static features of Android apps, linear SVM (used as NCM)
  - Concept drift scenario: malware evolution

- **Multiclass case study: Microsoft malware classification algorithm**
  - Solution to Microsoft Kaggle competition\(^2\), ranked among the top ones
  - Static features from Windows PE binaries, Random Forest (used as NCM)
  - Concept drift scenario: family discovery


Experimental Results: Case Studies

- Binary case study: Android malware detection algorithm
  - Reimplemented Drebin\(^1\) algorithm with similar results
    (0.95-0.92 precision-recall on malicious apps and 0.99-0.99 precision-recall on benign apps)
  - Static features of Android apps, linear SVM (used as NCM)
  - Concept drift scenario: malware evolution

---


Experimental Results: Binary Classification (Malware Evolution)

- Drebin dataset: samples collected from 2010 to 2012
- Marvin dataset\(^3\): malware apps collected from 2010 to 2014 (no duplicates)
  - We expect some object to drift from objects in the Drebin dataset

<table>
<thead>
<tr>
<th>Drebin Dataset</th>
<th>Marvin Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Samples</td>
</tr>
<tr>
<td>Benign</td>
<td>123,435</td>
</tr>
<tr>
<td>Malware</td>
<td>5,560</td>
</tr>
</tbody>
</table>

Experimental Results: Binary Classification (Malware Evolution)

Experiment: Drift Confirmation

- Training dataset: Drebin dataset
- Testing dataset: 4,500 benign and 4,500 malicious random samples from Marvin dataset

<table>
<thead>
<tr>
<th>Original label</th>
<th>Benign</th>
<th>Malicious</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benign</td>
<td>4,498</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Malicious</td>
<td>2,890</td>
<td>1,610</td>
<td>0.36</td>
</tr>
</tbody>
</table>

Precision

- 0.61
- 1
Experimental Results: Binary Classification (Malware Evolution)

Experiment: Drift Confirmation

- Training dataset: Drebin dataset
- Testing dataset: 4,500 benign and 4,500 malicious random samples from Marvin dataset

<table>
<thead>
<tr>
<th>Prediction label</th>
<th>Original label</th>
<th>Benign</th>
<th>Malicious</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Benign</td>
<td>4,498</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Malicious</td>
<td>2,890</td>
<td>1,610</td>
<td>0.36</td>
</tr>
<tr>
<td>Precision</td>
<td>0.61</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Marvin malicious app
Drebin malicious app
Experimental Results: Binary Classification (Malware Evolution)

Experiment: Drift Confirmation

- Training dataset: Drebin dataset
- Testing dataset: 4,500 benign and 4,500 malicious random samples from Marvin dataset

<table>
<thead>
<tr>
<th>Original label</th>
<th>Benign</th>
<th>Malicious</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benign</td>
<td>4,498</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Malicious</td>
<td>2,890</td>
<td>1,610</td>
<td>0.36</td>
</tr>
</tbody>
</table>

| Precision | 0.61 | 1 |

![Graph showing distribution of Marvin malicious apps, Drebin benign apps, and Drebin malicious apps]
Experimental Results: Binary Classification (Malware Evolution)

Experiment: Threshold Identification

- Training dataset: Drebin dataset
- Testing dataset: 4,500 benign and 4,500 malicious random samples from Marvin dataset
- Make use of Conformal Evaluator’s prediction assessment algorithm
  - Constraints: F1-score of 0.99 and 0.76 of elements marked as reliable

<table>
<thead>
<tr>
<th>Prediction label</th>
<th>Original label</th>
<th>Benign</th>
<th>Malicious</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Benign</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4,257</td>
<td>2</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Malicious</td>
<td>504</td>
<td>1,610</td>
<td>0.76</td>
</tr>
<tr>
<td>Precision</td>
<td>0.89</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Experimental Results: Binary Classification (Malware Evolution)

Experiment: Retraining

- Training dataset: Drebin dataset + samples marked as unreliable from previous experiment
- Testing dataset: 4,500 benign and 4,500 malicious random samples of Marvin dataset
  (no sample overlap from previous experiment)

<table>
<thead>
<tr>
<th>Sample</th>
<th>Benign</th>
<th>Malicious</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benign</td>
<td>4,413</td>
<td>87</td>
<td>0.98</td>
</tr>
<tr>
<td>Malicious</td>
<td>255</td>
<td>4,245</td>
<td>0.94</td>
</tr>
</tbody>
</table>

Precision: 0.96 0.98
Experimental Results: Binary Classification (Malware Evolution)

Experiment: Threshold Comparison

- Compare probability- and p-value-based thresholds
  - Central tendency and dispersion points of true positive distribution
- Training dataset: Drebin dataset
- Testing dataset: 4,500 benign and 4,500 malicious apps from Marvin dataset (random sampling)

<table>
<thead>
<tr>
<th></th>
<th>TPR (reliable predictions)</th>
<th>TPR (unreliable predictions)</th>
<th>FPR (reliable predictions)</th>
<th>FPR (unreliable predictions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>p-value</td>
<td>0.9045</td>
<td>0.0000</td>
<td>0.0007</td>
<td>0.0000</td>
</tr>
<tr>
<td>probability</td>
<td>0.6654</td>
<td>0.3176</td>
<td>0.0</td>
<td>0.0013</td>
</tr>
<tr>
<td>1st quartile</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Median</td>
<td>0.8737</td>
<td>0.3080</td>
<td>0.0000</td>
<td>0.0088</td>
</tr>
<tr>
<td>Mean</td>
<td>0.8737</td>
<td>0.3080</td>
<td>0.0000</td>
<td>0.0018</td>
</tr>
<tr>
<td>3rd quartile</td>
<td>0.8723</td>
<td>0.3411</td>
<td>0.0000</td>
<td>0.0005</td>
</tr>
</tbody>
</table>

TPR: True Positive Rate, FPR: False Positive Rate
**Experimental Results: Binary Classification (Malware Evolution)**

**Experiment: Threshold Comparison**

- Compare probability- and p-value-based thresholds
  - Central tendency and dispersion points of true positive distribution
- Training dataset: Drebin dataset
- Testing dataset: 4,500 benign and 4,500 malicious apps from Marvin dataset (random sampling)

<table>
<thead>
<tr>
<th></th>
<th>TPR (reliable predictions)</th>
<th>TPR (unreliable predictions)</th>
<th>FPR (reliable predictions)</th>
<th>FPR (unreliable predictions)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>p-value</td>
<td>probability</td>
<td>p-value</td>
<td>probability</td>
</tr>
<tr>
<td>1st quartile</td>
<td>0.9045</td>
<td>0.6654</td>
<td>0.0000</td>
<td>0.3176</td>
</tr>
<tr>
<td>Median</td>
<td>0.8737</td>
<td>0.8061</td>
<td>0.3080</td>
<td>0.3300</td>
</tr>
<tr>
<td>Mean</td>
<td>0.8737</td>
<td>0.4352</td>
<td>0.3080</td>
<td>0.3433</td>
</tr>
<tr>
<td>3rd quartile</td>
<td>0.8723</td>
<td>0.6327</td>
<td>0.3411</td>
<td>0.3548</td>
</tr>
</tbody>
</table>
Experimental Results: Binary Classification (Malware Evolution)

Experiment: Threshold Comparison

- Compare probability- and p-value-based thresholds
  - Central tendency and dispersion points of true positive distribution
- Training dataset: Drebin dataset
- Testing dataset: 4,500 benign and 4,500 malicious apps from Marvin dataset (random sampling)

<table>
<thead>
<tr>
<th></th>
<th>TPR</th>
<th>TPR</th>
<th>FPR</th>
<th>FPR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>p-value</td>
<td>probability</td>
<td>p-value</td>
<td>probability</td>
</tr>
<tr>
<td>1st quartile</td>
<td>0.9045</td>
<td>0.6654</td>
<td><strong>0.0000</strong></td>
<td>0.3176</td>
</tr>
<tr>
<td>Median</td>
<td>0.8737</td>
<td>0.8061</td>
<td>0.3080</td>
<td>0.3300</td>
</tr>
<tr>
<td>Mean</td>
<td>0.8737</td>
<td>0.4352</td>
<td>0.3080</td>
<td>0.3433</td>
</tr>
<tr>
<td>3rd quartile</td>
<td>0.8723</td>
<td>0.6327</td>
<td>0.3411</td>
<td>0.3548</td>
</tr>
</tbody>
</table>
Conclusion
Conformal Evaluator (CE)
Statistical evaluation to assess predictions of ML classifiers and identify concept drift
Conclusion

Conformal Evaluator (CE)
Statistical evaluation to assess predictions of ML classifiers and identify concept drift

Algorithm Agnostic: Uses non-conformity measure (NCM) from the ML classifier
Statistical Support: Builds p-values from NCM to statistically-support predictions
Quality Thresholds: Builds thresholds from p-values to identify unreliable predictions

We evaluate the proposed solution on different ML classifiers and case studies
- Android malware apps in binary classification settings
- Windows PE binaries in multi-class classification settings

Information on CE’s python code and dataset availability at:
https://s2lab.isg.rhul.ac.uk/projects/ce
Conclusion

Conformal Evaluator (CE)
Statistical evaluation to assess predictions of ML classifiers and identify concept drift

Algorithm Agnostic: Uses non-conformity measure (NCM) from the ML classifier
Statistical Support: Builds p-values from NCM to statistically-support predictions
Quality Thresholds: Builds thresholds from p-values to identify unreliable predictions

• We evaluate the proposed solution on different ML classifiers and case studies
  • Android malware apps in binary classification settings
  • Windows PE binaries in multi-class classification settings
• Information on CE’s python code and dataset availability at:
  https://s2lab.isg.rhul.ac.uk/projects/ce
Backup Slides
Table 4: Thresholds comparison between p-value and probability. The results show, together with the performance of the sample marked as unreliable, a clear advantage of the p-value metric compared to the probability one.
P-value vs Probability: situation 1

<table>
<thead>
<tr>
<th></th>
<th>P-value</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>0.0</td>
<td>0.5</td>
</tr>
<tr>
<td>Green</td>
<td>0.0</td>
<td>0.5</td>
</tr>
</tbody>
</table>
P-value vs Probability: situation 2

<table>
<thead>
<tr>
<th></th>
<th>P-value</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>Green</td>
<td>0.5</td>
<td>0.5</td>
</tr>
</tbody>
</table>
Experimental Results: Multiclass classification (new family discovery)

- Dataset: Microsoft Malware Classification Challenge (2015)

<table>
<thead>
<tr>
<th>Malware</th>
<th>Samples</th>
<th>Malware</th>
<th>Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ramnit</td>
<td>1,541</td>
<td>Obfuscator.ACY</td>
<td>1,228</td>
</tr>
<tr>
<td>Lollipop</td>
<td>2,478</td>
<td>Gatak</td>
<td>1,013</td>
</tr>
<tr>
<td>Kelihos_ver3</td>
<td>2,942</td>
<td>Kelihos_ver1</td>
<td>398</td>
</tr>
<tr>
<td>Vundo</td>
<td>475</td>
<td>Tracur</td>
<td>751</td>
</tr>
</tbody>
</table>
**Experimental Results: Multiclass classification (new family discovery)**

**Experiment: Family Discovery**

- Training families: Ramnit, Lollipop, Kelihos\_ver3, Vundo, Obfuscator\_ACY, Gatak, Kelihos\_ver1
- Testing family: Tracur

**Classification results:**

<table>
<thead>
<tr>
<th></th>
<th>Lollipop</th>
<th>Kelihos_ver3</th>
<th>Vundo</th>
<th>Kelihos_ver1</th>
<th>Obfuscator_ACY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>5</td>
<td>6</td>
<td>358</td>
<td>140</td>
<td>242</td>
</tr>
</tbody>
</table>
Experimental Results: Multiclass classification (new family discovery)

**P-value distribution** for samples of Tracur family; as expected, the values are all close to zero.

- Prediction: Ramnit
- Prediction: Lollipop
- Prediction: Kelihos_ver3
- Prediction: Vundo
- Prediction: Kelihos_ver1
- Prediction: Obfuscator.ACY
- Prediction: Gatak
Probability distribution for samples of Tracur family; bounded to sum to one, the values are different than zero.