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.

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

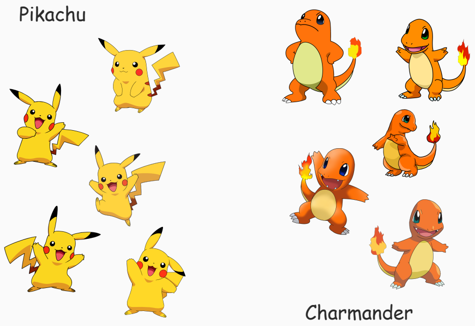
Objects are described as vectors of features

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

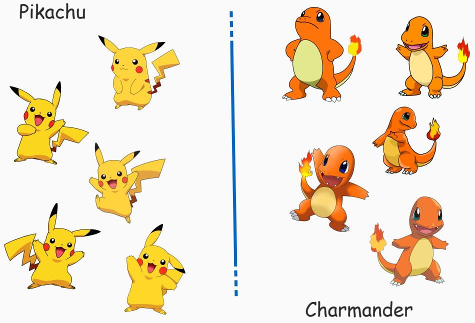


Machine Learning Classification

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



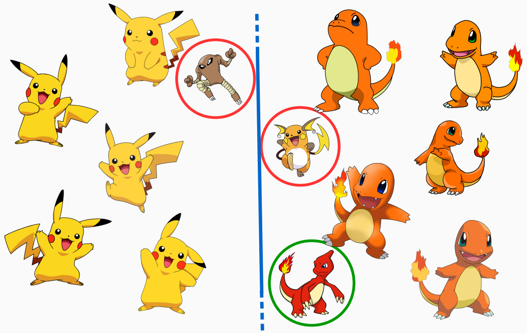
Machine Learning Classification Problem: Concept Drift

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

Hitmonlee
(new pokémon family)
Raichu
(evolution of Pikachu)
Charmeleon
(evolution of Charmander)

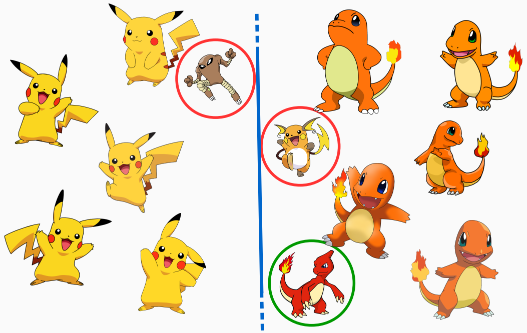
Machine Learning Classification Problem: Concept Drift

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



Machine Learning Classification Problem: Concept Drift

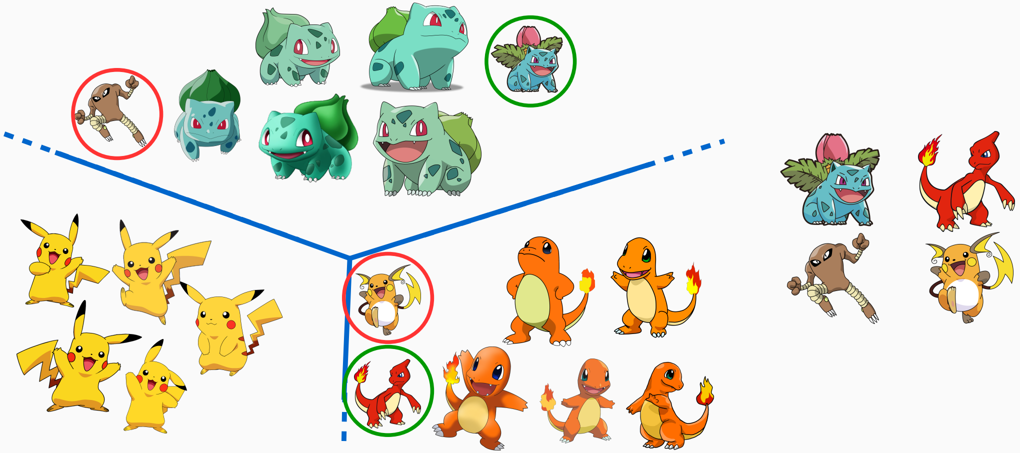
- *Concept drift* is the change in the statistical properties of an object in unforeseen ways
- Drifted objects will likely be wrongly classified



Of course, the problem exists in multiclass classification settings. . .

Machine Learning Classification Problem: Concept Drift

- 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

$$\begin{aligned}
\alpha_t &= \text{Non-conformity score for test element t} \\
\forall i \in \mathcal{K}, \alpha_i &= \text{Non-conformity score for train element i} \\
\text{p-value} &= \frac{|\{i : \alpha_i \geq \alpha_t\}|}{|\mathcal{K}|} \\
\mathcal{K} &= \text{Total number of element}
\end{aligned}$$

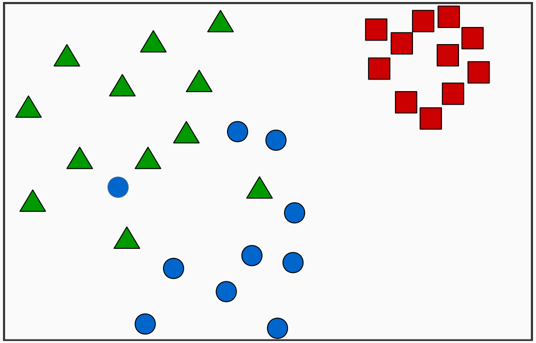
P-value

Ratio between the number of training elements that are more dissimilar than the element under test

Conformal Evaluator: P-value Example

ML classifier:

distance from centroid

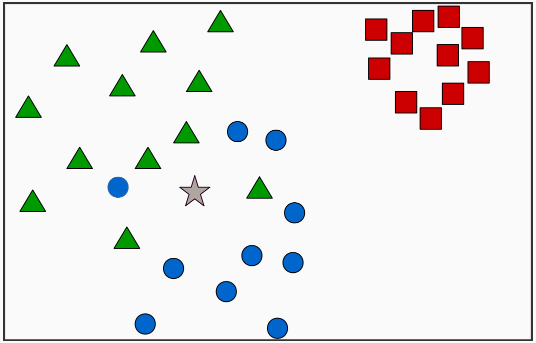


1. Setting: 3-class classification

Conformal Evaluator: P-value Example

ML classifier:

distance from centroid

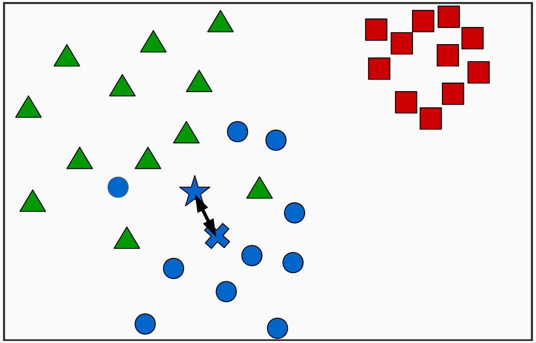


1. Setting: 3-class classification
2. Test object

Conformal Evaluator: P-value Example

ML classifier:

distance from centroid

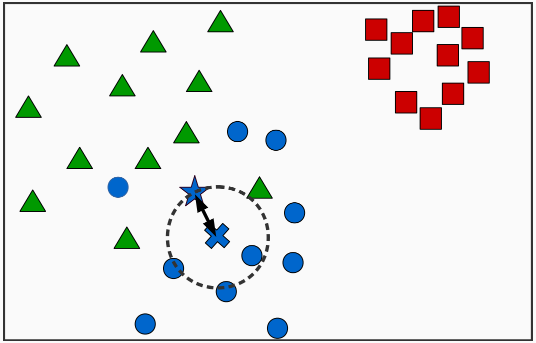


1. Setting: 3-class classification
2. Test object
    3.1 Compute distance to blue class

Conformal Evaluator: P-value Example

ML classifier:

distance from centroid

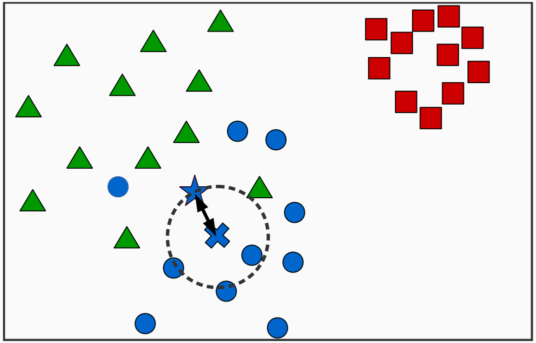


1. Setting: 3-class classification
2. Test object
    3.1 Compute distance to blue class
    3.2 How many objects are more dissimilar than the one under test?

Conformal Evaluator: P-value Example

ML classifier:

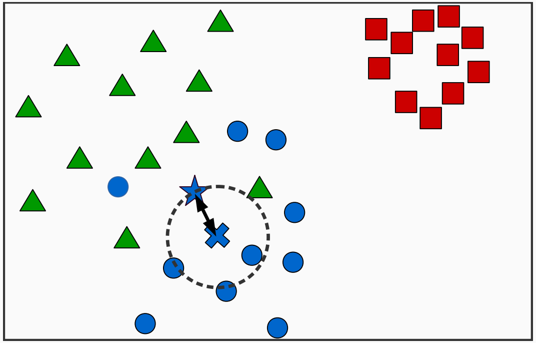
distance from centroid



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

Conformal Evaluator: P-value Example

ML classifier:
distance from centroid

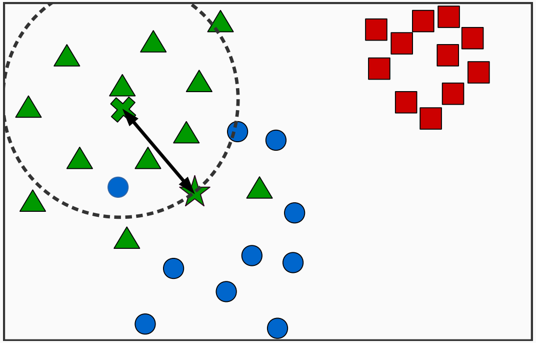


1. Setting: 3-class classification
2. Test object
    3.1 Compute distance to blue class
    3.2 How many objects are more dissimilar than the one under test?
    3.3 9
    3.4 P-value$_{\star} = \frac{9}{10}$

Conformal Evaluator: P-value Example

Machine learning classifier:

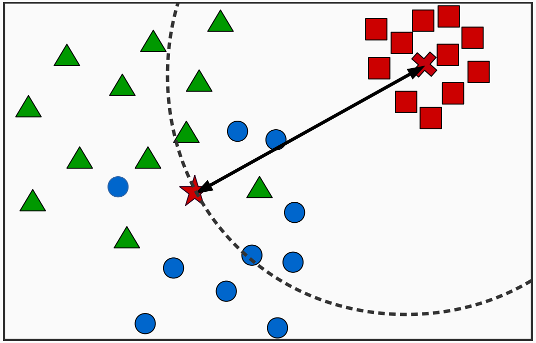
distance from centroid



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$_{\star} = \frac{4}{12}$

Conformal Evaluator: P-value Example

Machine learning classifier:
distance from centroid

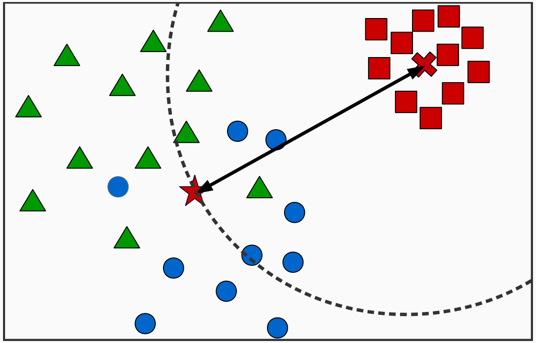


1. Initial situation: three classes
2. Test object
    - 5.1 Calculate distance to red class
    - 5.2 How many objects are more dissimilar than the one under test?
    - 5.3 0
    - 5.4 P-value$_{\star} = \frac{0}{11}$

Conformal Evaluator: P-value Example

Machine learning classifier:

distance from centroid

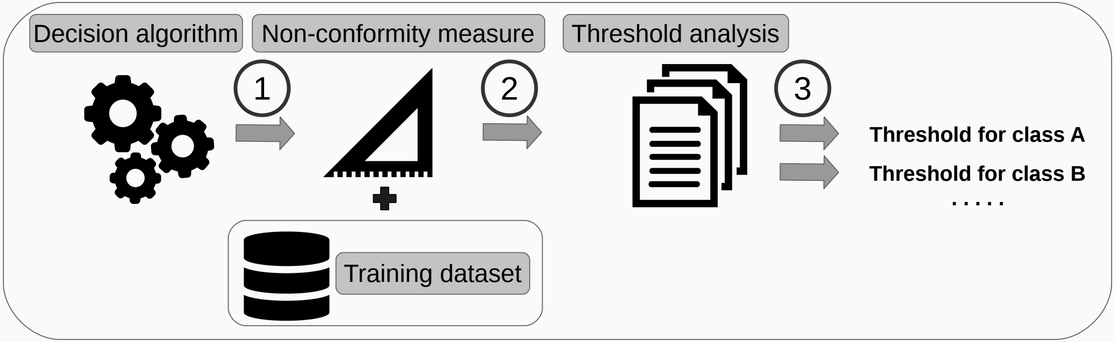


1. Initial situation: three classes
2. Test object
   - 5.1 Calculate distance to red class
   - 5.2 How many objects are more dissimilar than the one under test?
   - 5.3 0
   - 5.4 P-value$_\star$ $= \frac{0}{11}$

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

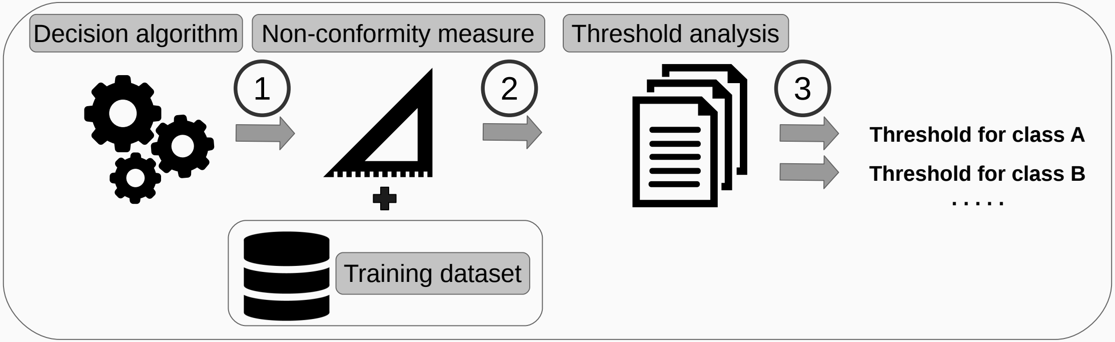
Conformal Evaluator: How Does it Work?

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)



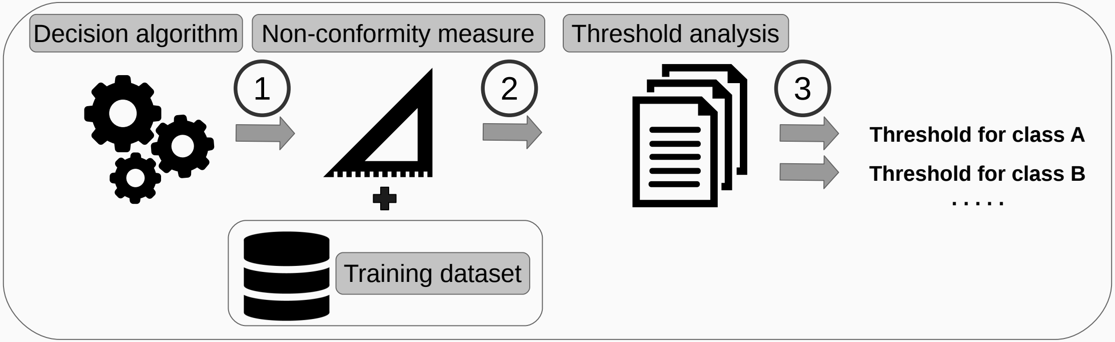
Conformal Evaluator: How Does it Work?

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



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



Conformal Evaluator: Identifying per-class Thresholds

Customizable constraints:

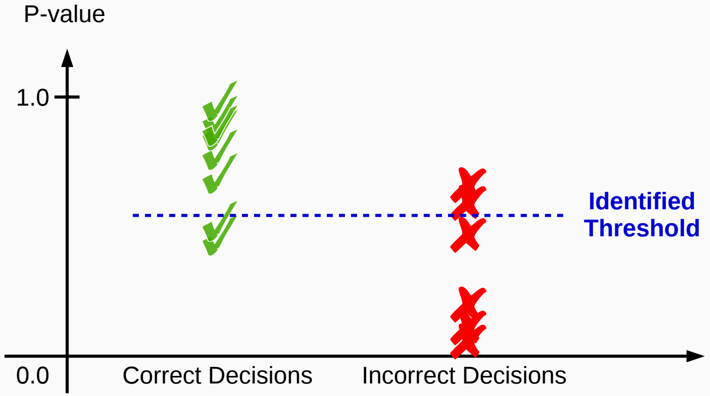
- Desired performance (of the predictions marked as reliable)
    - E.g.: high-level performance will raise the threshold
- Number of unreliable prediction tolerated
    - E.g.: low number of unreliable prediction will lower the threshold

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



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

---

[1] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck. Drebin: Effective and Explainable Detection of Android Malware in Your Pocket. In *21st Annual Network and Distributed System Security Symposium (NDSS), San Diego, California, USA, February 23-26, 2014.*

[2] KAGGLE INC. Microsoft Malware Classification Challenge (BIG 2015). https://www.kaggle.com/c/malware-classification, 2015.

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

[1] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck. Drebin: Effective and Explainable Detection of Android Malware in Your Pocket. In *21st Annual Network and Distributed System Security Symposium (NDSS), San Diego, California, USA, February 23-26, 2014.*

[2] KAGGLE INC. Microsoft Malware Classification Challenge (BIG 2015). https://www.kaggle.com/c/malware-classification, 2015.

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

DREBIN DATASET

Type	Samples
Benign	123,435
Malware	5,560

MARVIN DATASET

Type	Samples
Benign	9,592
Malware	9,179

[3] Martina Lindorfer, Matthias Neugschwandtner, and Christian Platzer. MARVIN: Efficient and Comprehensive Mobile App Classification through Static And Dynamic Analysis. In *39th IEEE Annual Computer Software and Applications Conference (COMPSAC), Taichung, Taiwan, July 1-5, 2015.*

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

	Prediction label		
Original label	Benign	Malicious	**Recall**
Benign	4,498	2	1
Malicious	2,890	1,610	0.36
Precision	0.61	1	

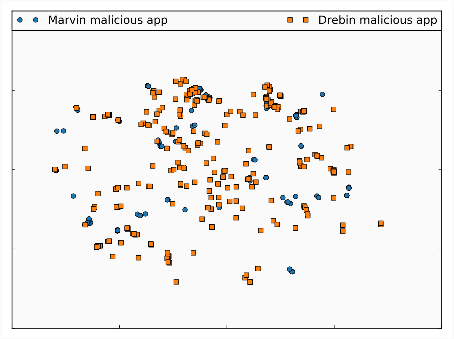
Experimental Results: Binary Classification (Malware Evolution)

Experiment: Drift Confirmation

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

Prediction label

Original label	Benign	Malicious	**Recall**
Benign	4,498	2	1
Malicious	2,890	1,610	0.36
Precision	0.61	1	



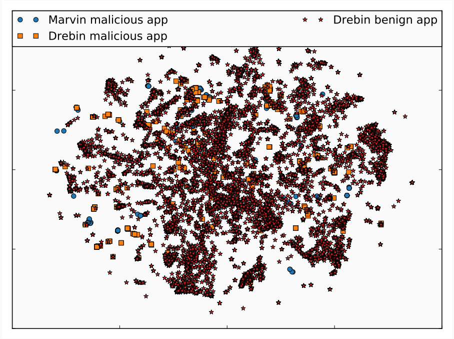
Experimental Results: Binary Classification (Malware Evolution)

Experiment: Drift Confirmation

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

Prediction label

Original label	Benign	Malicious	**Recall**
Benign	4,498	2	1
Malicious	2,890	1,610	0.36
Precision	0.61	1	



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

Prediction label

Original label	Benign	Malicious	**Recall**
Benign	4,257	2	1
Malicious	504	1,610	0.76
Precision	0.89	1	

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)

	Assigned label		
Sample	Benign	Malicious	**Recall**
Benign	4,413	87	0.98
Malicious	255	4,245	0.94
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)

	TPR (reliable predictions)		TPR (unreliable predictions)		FPR (reliable predictions)		FPR (unreliable predictions)	
	p-value	probability	p-value	probability	p-value	probability	p-value	probability
1st quartile	0.9045	0.6654	0.0000	0.3176	0.0007	0.0	0.0000	0.0013
Median	0.8737	0.8061	0.3080	0.3300	0.0000	0.0	0.0008	0.0008
Mean	0.8737	0.4352	0.3080	0.3433	0.0000	0.0	0.0008	0.0018
3rd quartile	0.8723	0.6327	0.3411	0.3548	0.0000	0.0	0.0005	0.0005

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)

	TPR (reliable predictions)		TPR (unreliable predictions)		FPR (reliable predictions)		FPR (unreliable predictions)	
	p-value	probability	p-value	probability	p-value	probability	p-value	probability
1st quartile	0.9045	0.6654	0.0000	0.3176	0.0007	0.0	0.0000	0.0013
Median	0.8737	0.8061	0.3080	0.3300	0.0000	0.0	0.0008	0.0008
Mean	0.8737	0.4352	0.3080	0.3433	0.0000	0.0	0.0008	0.0018
3rd quartile	0.8723	0.6327	0.3411	0.3548	0.0000	0.0	0.0005	0.0005

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)

	TPR (reliable predictions)		TPR (unreliable predictions)		FPR (reliable predictions)		FPR (unreliable predictions)	
	p-value	probability	p-value	probability	p-value	probability	p-value	probability
1st quartile	0.9045	0.6654	0.0000	0.3176	0.0007	0.0	0.0000	0.0013
Median	0.8737	0.8061	0.3080	0.3300	0.0000	0.0	0.0008	0.0008
Mean	0.8737	0.4352	0.3080	0.3433	0.0000	0.0	0.0008	0.0018
3rd quartile	0.8723	0.6327	0.3411	0.3548	0.0000	0.0	0.0005	0.0005

Conclusion

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

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`

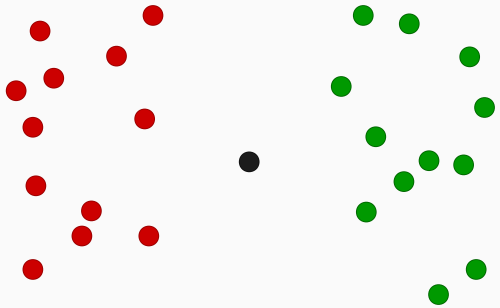
Backup Slides

Binary Classification Case Study: Comparison with Probability

	TPR of kept elements		FPR of kept elements		TPR of discarded elements		FPR of discarded elements		MALICIOUS kept elements		BENIGN kept elements	
	p-value	probability	p-value	probability	p-value	probability	p-value	probability	p-value	probability	p-value	probability
1st quartile	0.9045	0.6654	0.0007	0.0	0.0000	0.3176	0.0000	0.0013	0.3956	0.1156	0.6480	0.6673
Median	0.8737	0.8061	0.0000	0.0	0.3080	0.3300	0.0008	0.0008	0.0880	0.0584	0.4136	0.4304
Mean	0.8737	0.4352	0.0000	0.0	0.3080	0.3433	0.0008	0.0018	0.0880	0.1578	0.4136	0.7513
3rd quartile	0.8723	0.6327	0.0000	0.0	0.3411	0.3548	0.0005	0.0005	0.0313	0.0109	0.1573	0.1629

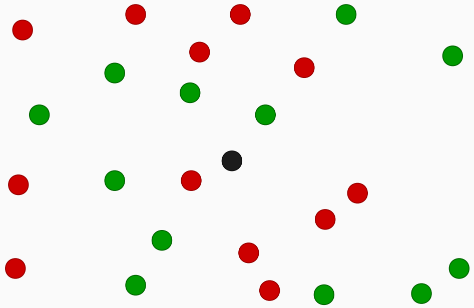
Table 4: Thresholds comparison between p-value and probability. The results show, together with the performance of the sample marked as unreliable, a clear advantage of the p-value metric compared to the probability one.

P-value vs Probability: situation 1



	P-value	Probability
Red	0.0	0.5
Green	0.0	0.5

P-value vs Probability: situation 2



	P-value	Probability
Red	0.5	0.5
Green	0.5	0.5

Experimental Results: Multiclass classification (new family discovery)

- Dataset: Microsoft Malware Classification Challenge (2015)

MICROSOFT MALWARE CLASSIFICATION CHALLENGE DATASET

Malware	Samples	Malware	Samples
Ramnit	1 541	Obfuscator.ACY	1 228
Lollipop	2 478	Gatak	1 013
Kelihos_ver3	2 942	Kelihos_ver1	398
Vundo	4 75	Tracur	751

Experimental Results: Multiclass classification (new family discovery)

Experiment: Family Discovery

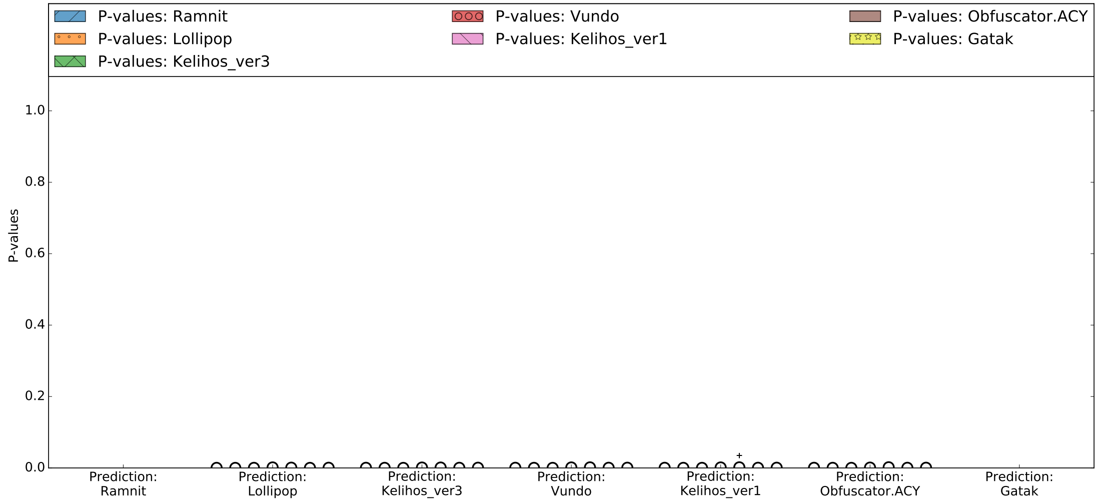
- Training families: Ramnit, Lollipop, Kelihos_ver3, Vundo, Obfuscator.ACY, Gatak, Kelihos_ver1
- Testing family: Tracur

Classification results:

Lollipop	Kelihos_ver3	Vundo	Kelihos_ver1	Obfuscator.ACY
5	6	358	140	242

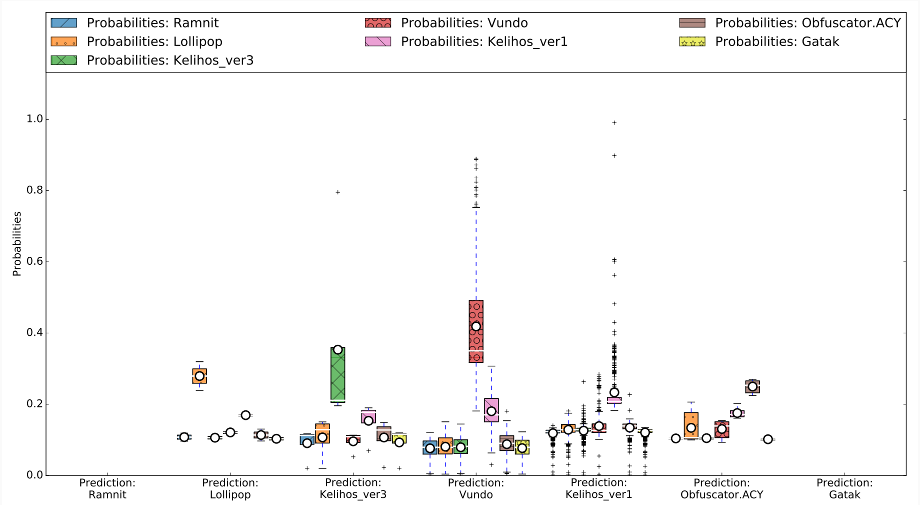
Experimental Results: Multiclass classification (new family discovery)

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



Experimental Results: Multiclass classification (new family discovery)

Probability distribution for samples of Tracur family; bounded to sum to one, the values are different than zero.


Probabilities: Ramnit
Probabilities: Lollipop
Probabilities: Kelihos_ver3
Probabilities: Vundo
Probabilities: Kelihos_ver1
Probabilities: Obfuscator.ACY
Probabilities: Gatak
Probabilities
Prediction: Ramnit
Prediction: Lollipop
Prediction: Kelihos_ver3
Prediction: Vundo
Prediction: Kelihos_ver1
Prediction: Obfuscator.ACY
Prediction: Gatak