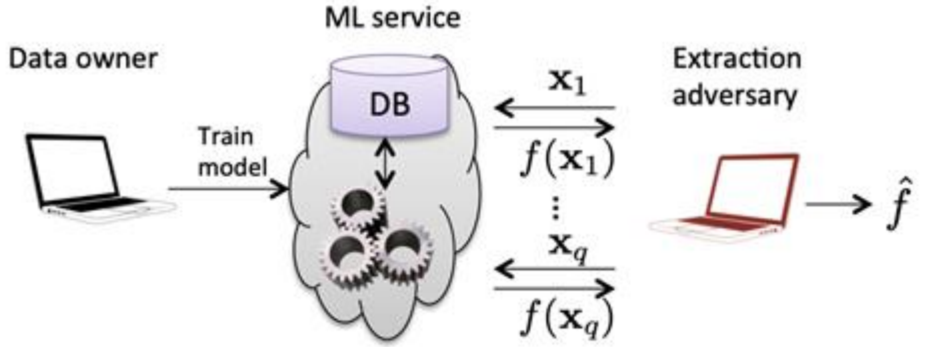
Model Stealing Attacks

Ryan DeVries, Russell Barton, Noah Fredericks, Charlie Berens

Model Stealing

Background

- Function *f* models feature space *X* to *f(x)*, the codomain *Y*
  - $f: X \to Y$
- Probability of every class label is evaluated, and the largest is determined as the determined class label
- In comparing class probabilities, we use: $d(\mathbf{y},\mathbf{y}') = \frac{1}{2}\sum|\mathbf{y}[i]-\mathbf{y}'[i]|$
- Gaining access to target model $\hat{f}$

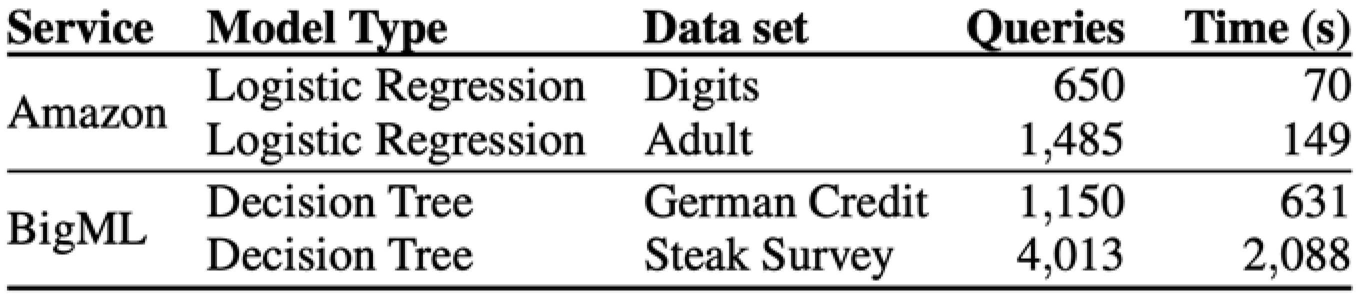


Model Extraction Methods

- Simple equation solving
  - Logistic regression
  - Non-adaptive, random queries
  - Requires confidence values
- Path-finding algorithms
  - Decision trees
  - Requires confidence values
- Model-extraction (only class labels)
  - No confidence values
  - Utilizes learning theory
  - Requires, on average, 100 times more queries

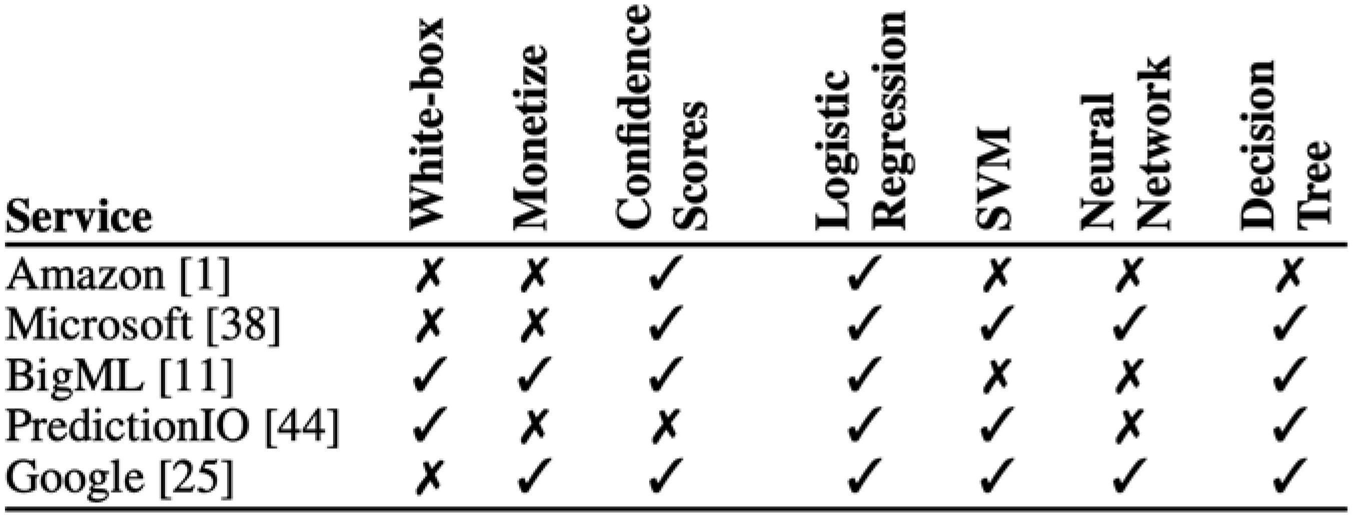
Model-Extracting Existing Services

- These services return confidence values as well as class labels
- Equation-solving
  - Works for logistic regression
  - Does not work for decision trees



Service	Model Type	Data set	Queries	Time (s)
Amazon	Logistic Regression	Digits	650	70
	Logistic Regression	Adult	1,485	149
BigML	Decision Tree	German Credit	1,150	631
	Decision Tree	Steak Survey	4,013	2,088

Model Extraction Attacks (ML Services)

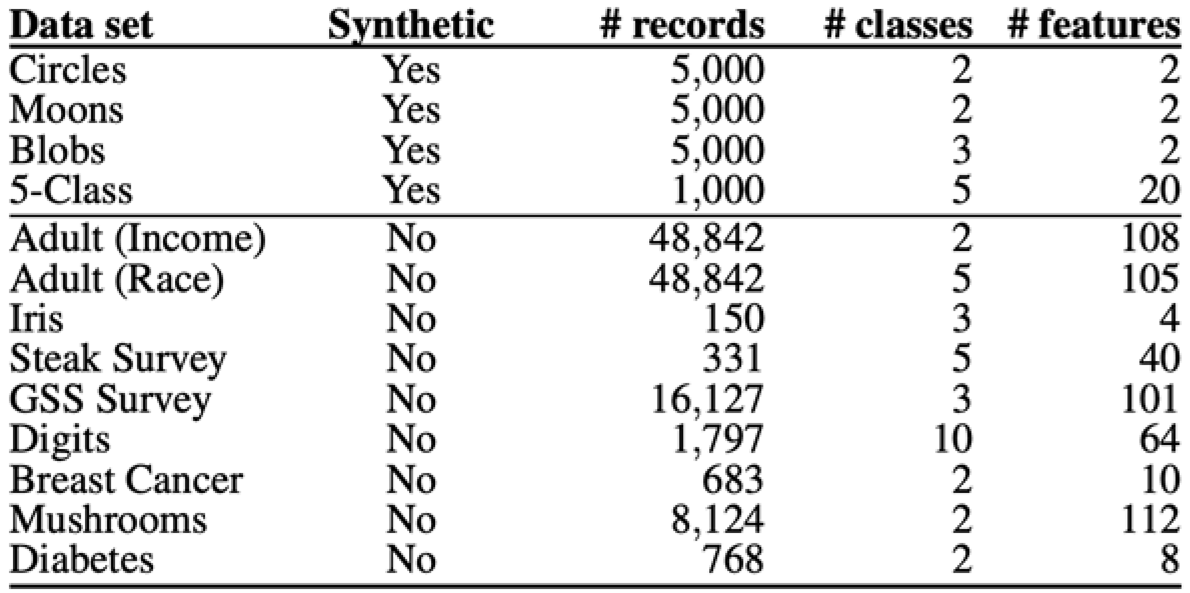


Service	White-box	Monetize	Confidence Scores	Logistic Regression	SVM	Neural Network	Decision Tree
Amazon [1]	✗	✗	✓	✓	✗	✗	✗
Microsoft [38]	✗	✗	✓	✓	✓	✓	✓
BigML [11]	✓	✓	✓	✓	✗	✗	✓
PredictionIO [44]	✓	✗	✗	✓	✓	✗	✓
Google [25]	✗	✓	✓	✓	✓	✓	✓

Adversarial Accuracy Metrics

- Test error R_{test}: $R_{\text{test}}(f,\hat{f}) = \sum_{(\mathbf{x},y)\in D} d(f(\mathbf{x}),\hat{f}(\mathbf{x}))/|D|$
- Uniform Error R_{unif}: $R_{\text{unif}}(f,\hat{f}) = \sum_{\mathbf{x}\in U} d(f(\mathbf{x}),\hat{f}(\mathbf{x}))/|U|$
- 1 - R represents extraction accuracy

Relevant Datasets



Data set	Synthetic	# records	# classes	# features
Circles	Yes	5,000	2	2
Moons	Yes	5,000	2	2
Blobs	Yes	5,000	3	2
5-Class	Yes	1,000	5	20
Adult (Income)	No	48,842	2	108
Adult (Race)	No	48,842	5	105
Iris	No	150	3	4
Steak Survey	No	331	5	40
GSS Survey	No	16,127	3	101
Digits	No	1,797	10	64
Breast Cancer	No	683	2	10
Mushrooms	No	8,124	2	112
Diabetes	No	768	2	8

Binary Logistic Regression

- Binary classification (c=2) and calculates probabilities based on independent features
- With **w** and **β** contained within all real values:

$$f_1(\mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x} + \beta) \qquad \sigma(t) = 1/(1+e^{-t})$$

- A linear classifier, with hyperplane separating the two classes: $\mathbf{w} \cdot \mathbf{x} + \beta = 0$
- With queried **x**, d+1 samples is necessary for the linear equation:

$$\mathbf{w} \cdot \mathbf{x} + \beta = \sigma^{-1}(f_1(\mathbf{x}))$$

- After d+1 samples, $R_{test} = R_{unif} = 0$
- On average, d = 40, with 113 queries being the maximum

Multiclass Logistic Regression (MLR)

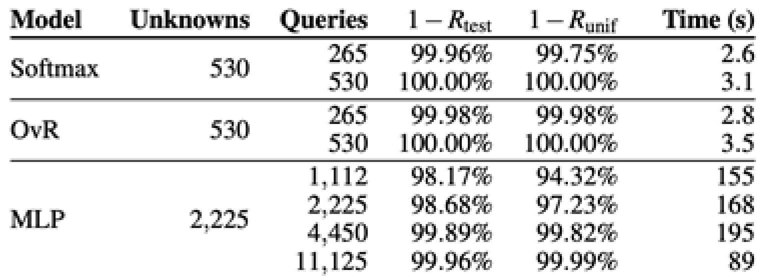
- For logistic regression models of c > 2
- **w**$_i$ and **β**$_i$ instead of **w** and **β** to form multiclass model
- Two forms of multiclass logistic regression models:
  - Softmax: fits a singular joint multinomial distribution to all training samples
  - OvR: trains a separate binary LR for each class, then normalizes the class probabilities
- For softmax models:

$$f_i(\mathbf{x}) = e^{\mathbf{w}_i \cdot \mathbf{x} + \beta_i} / (\textstyle\sum_{j=0}^{c-1} e^{\mathbf{w}_j \cdot \mathbf{x} + \beta_j})$$

- Solved by minimizing the loss function:
  - The loss function is strongly convex
  - The innate concavity means there is a global maximum

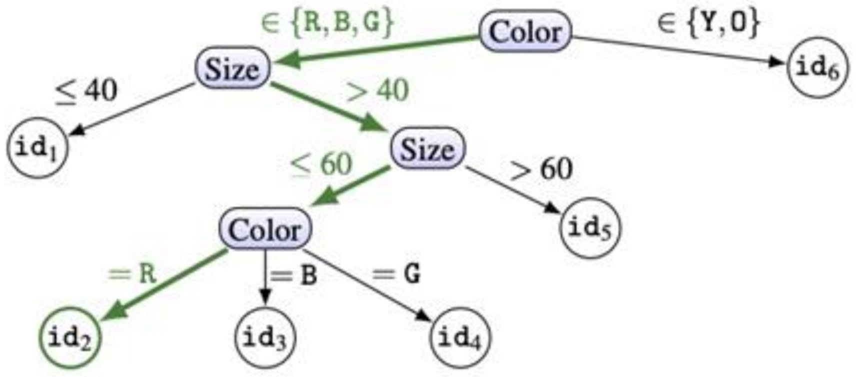
Multilayer Perceptrons (MLP)

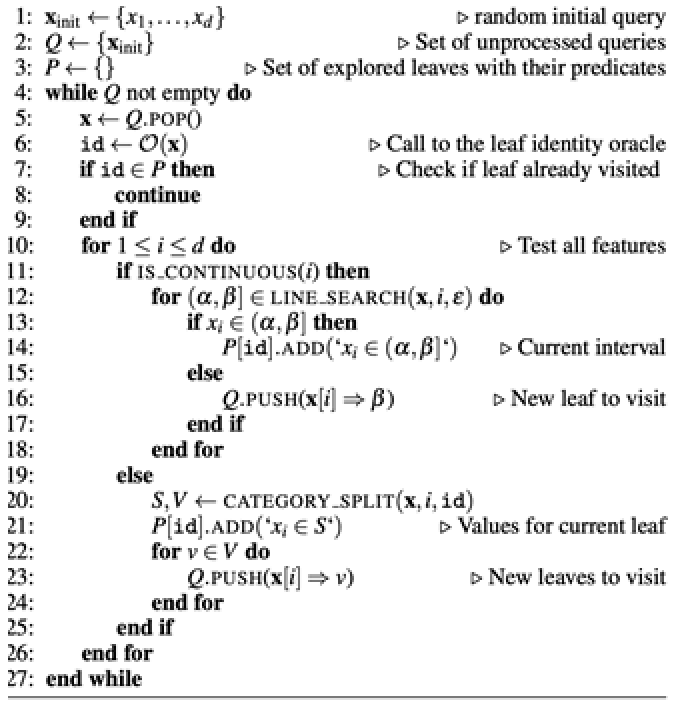
- Regarding perceptrons with one hidden layer: $\mathbf{w} \in \mathbb{R}^{dh+hc}$, $\boldsymbol{\beta} \in \mathbb{R}^{h+c}$ where *h* is the number of hidden nodes
- Compared to MLR, MLP are not strongly convex and thus do not converge to a global maximum
- Instead, it may converge to a local minimum, meaning $\hat{f}$ will not exactly match *f*
- MLPs have many more unknowns than MLRs, resulting in significantly more queries to achieve the same result
  - MLR: 260 query average, 650 maximum [Digits]
  - MLP: 5,410 query average, 11,125 maximum (> 99.9% accuracy) [Adult]
  - 54,100 average queries to achieve 100% accuracy



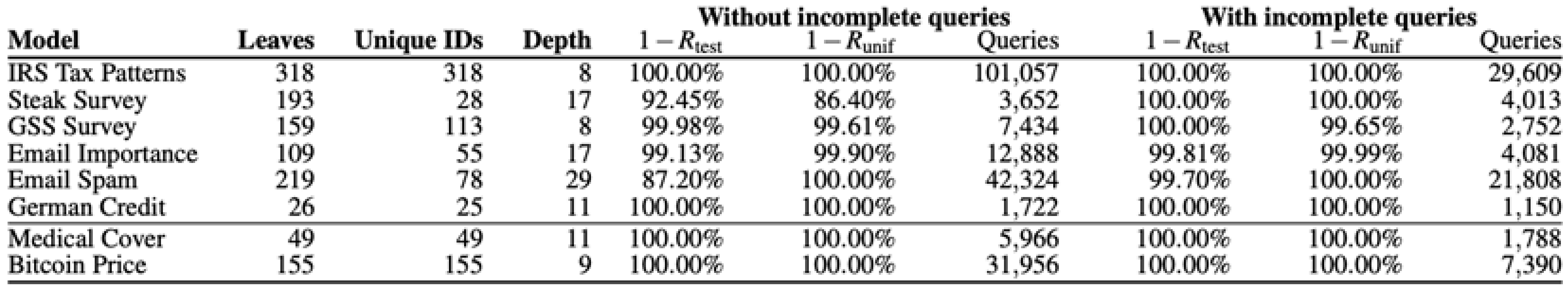
Model	Unknowns	Queries	$1 - R_{\text{test}}$	$1 - R_{\text{unif}}$	Time (s)
Softmax	530	265	99.96%	99.75%	2.6
		530	100.00%	100.00%	3.1
OvR	530	265	99.98%	99.98%	2.8
		530	100.00%	100.00%	3.5
MLP	2,225	1,112	98.17%	94.32%	155
		2,225	98.68%	97.23%	168
		4,450	99.89%	99.82%	195
		11,125	99.96%	99.99%	89

Decision Tree Attacks

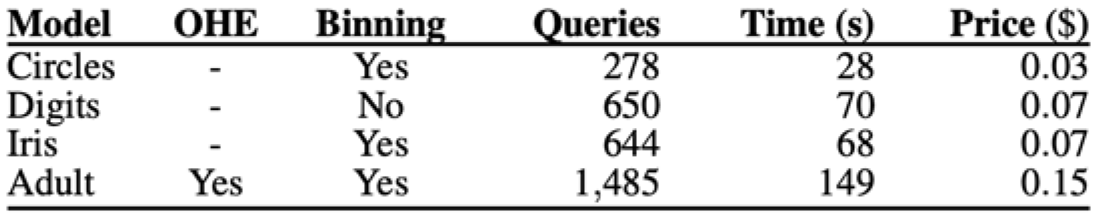



```
$\mathbf{x}_{\text{init}} \leftarrow \{x_1, \dots, x_d\}$                    ▷ random initial query
$Q \leftarrow \{\mathbf{x}_{\text{init}}\}$                    ▷ Set of unprocessed queries
$P \leftarrow \{\}$                    ▷ Set of explored leaves with their predicates
while $Q$ not empty do
    $\mathbf{x} \leftarrow Q$.POP()
    id $\leftarrow \mathcal{O}(\mathbf{x})$                    ▷ Call to the leaf identity oracle
    if id $\in P$ then                    ▷ Check if leaf already visited
        continue
    end if
    for $1 \leq i \leq d$ do                    ▷ Test all features
        if IS_CONTINUOUS($i$) then
            for $(\alpha, \beta] \in$ LINE_SEARCH($\mathbf{x}, i, \varepsilon$) do
                if $x_i \in (\alpha, \beta]$ then
                    $P$[id].ADD('$x_i \in (\alpha, \beta]$')                    ▷ Current interval
                else
                    $Q$.PUSH($\mathbf{x}[i] \Rightarrow \beta$)                    ▷ New leaf to visit
                end if
            end for
        else
            $S, V \leftarrow$ CATEGORY_SPLIT($\mathbf{x}, i$, id)
            $P$[id].ADD('$x_i \in S$')                    ▷ Values for current leaf
            for $v \in V$ do
                $Q$.PUSH($\mathbf{x}[i] \Rightarrow v$)                    ▷ New leaves to visit
            end for
        end if
    end for
end while
```

Online Model Extraction Attacks



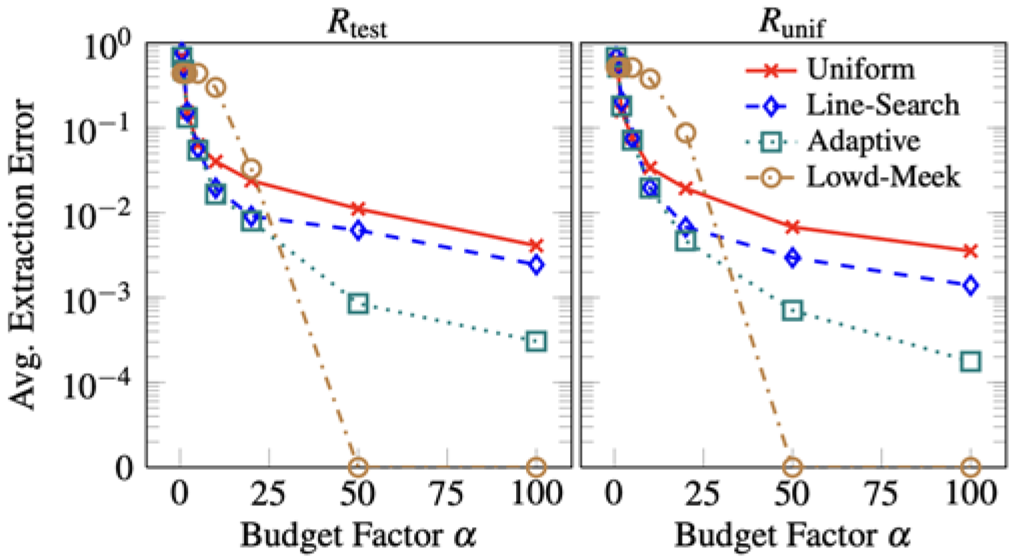
				Without incomplete queries			With incomplete queries		
Model	**Leaves**	**Unique IDs**	**Depth**	$1 - R_{\text{test}}$	$1 - R_{\text{unif}}$	Queries	$1 - R_{\text{test}}$	$1 - R_{\text{unif}}$	Queries
IRS Tax Patterns	318	318	8	100.00%	100.00%	101,057	100.00%	100.00%	29,609
Steak Survey	193	28	17	92.45%	86.40%	3,652	100.00%	100.00%	4,013
GSS Survey	159	113	8	99.98%	99.61%	7,434	100.00%	99.65%	2,752
Email Importance	109	55	17	99.13%	99.90%	12,888	99.81%	99.99%	4,081
Email Spam	219	78	29	87.20%	100.00%	42,324	99.70%	100.00%	21,808
German Credit	26	25	11	100.00%	100.00%	1,722	100.00%	100.00%	1,150
Medical Cover	49	49	11	100.00%	100.00%	5,966	100.00%	100.00%	1,788
Bitcoin Price	155	155	9	100.00%	100.00%	31,956	100.00%	100.00%	7,390



Model	**OHE**	**Binning**	**Queries**	**Time (s)**	**Price ($)**
Circles	-	Yes	278	28	0.03
Digits	-	No	650	70	0.07
Iris	-	Yes	644	68	0.07
Adult	Yes	Yes	1,485	149	0.15

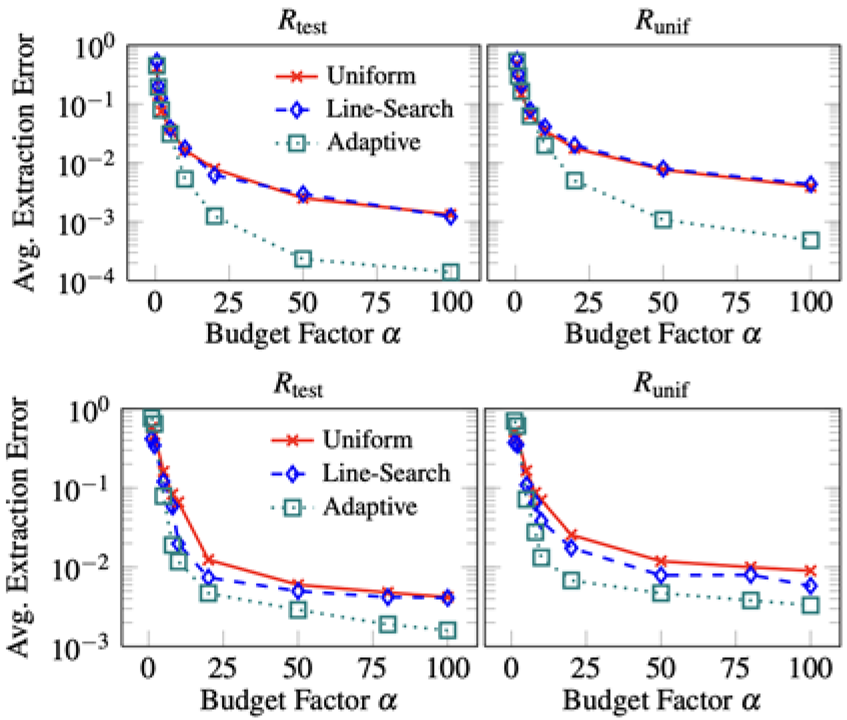
Class Label Only (Linear Models)

- Lowd-Meek Attack
  - Only for linear binary models
  - Estimates the direction the input should be modified
- Uniform queries
  - Fully retrains the model sampling points in the feature space
- Line-search
  - Uses *m* adaptive queries to find samples to approximate the decision boundary
- Adaptive
  - Uses *m/r* queries, iterating over *r* rounds to approximate areas of least certainty



Class Label Only (Multiclass LR Models)

- Will use retraining approach
- Treats softmax and OvR the same
- Neural Networks
  - Similar results for all retraining approaches
  - For a budget of 100 • *k*
    - R_{test} = 99.16%
    - R_{unif} = 98.24%
  - Average of 108,200 queries
- RBF Kernel SVMs
  - Adaptive training is most effective
  - Average of 99% accuracy with 2,050 queries



Model Extraction Implications

- Leaks statistics about private training sets (Training-data privacy)
- Avoiding query charges
  - Cross-user model extraction
  - Learn the black-box model to pay less for decreased query payment
- Evasion - learn how models evade adversarial attacks to strength the attacks themselves

Model Extraction Methods

- Simple equation solving
  - Logistic regression
  - Non-adaptive, random queries
  - Requires confidence values
- Path-finding algorithms
  - Decision trees
  - Requires confidence values
- Model-extraction (only class labels)
  - No confidence values
  - Utilizes learning theory
  - Requires, on average, 100 times more queries

Hyperparameter Stealing

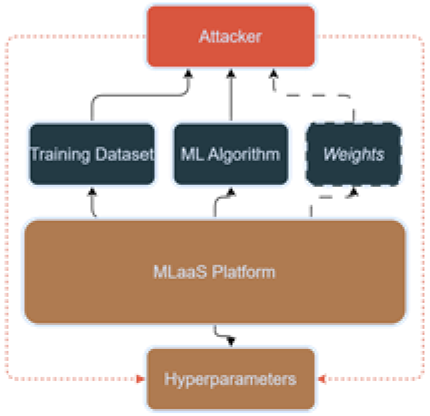
Hyperparameters, why do we care?

- **Hyperparameters** play a crucial role in determining the **performance** of a model and are not commonly disclosed.
    - **Regularization strength**: Controls overfitting by penalizing complex models.
    - **Learning rate**: Determines how quickly a model learns during training.
    - **Momentum strength**: Helps accelerate gradient descent by smoothing the optimization path.
- Expensive to Optimize: Hyperparameter tuning is computationally intensive and costly.
- Commercial Sensitivity: For services like **Machine Learning-as-a-Service (MLaaS)**, hyperparameters represent proprietary knowledge derived from extensive computational effort, providing a competitive advantage.

Which ones can be stolen?

- The hyperparameter stealing attack works for hyperparameters that directly impact the objective function.
- Examples include:
  - Regularization strength: Appears in the objective function and balances the loss and regularization terms.
  - Trade-off parameters in multi-objective loss functions: Such as the weight balancing reconstruction and classification loss in certain models.
- Hyperparameters like learning rate or number of layers do not directly affect the objective function and thus cannot be stolen using this method.
- Unlike model parameters (weights), which are learned from the data during training, hyperparameters are specified before training and significantly impact the model’s final quality.

Attacker’s Perspective



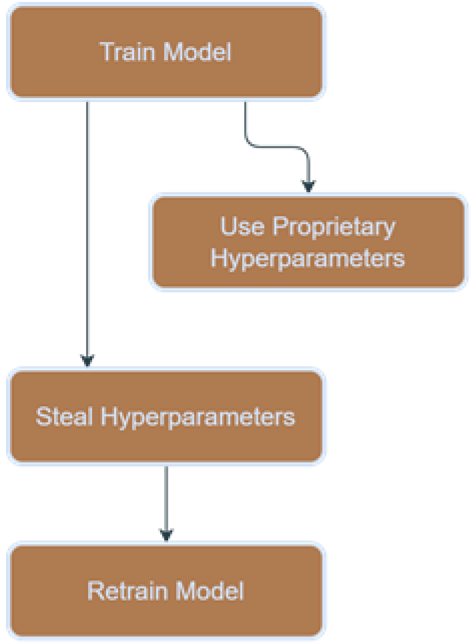
- Required Resources:
  - Training Dataset: The attacker has access to the training data used.
  - ML Algorithm: The attacker knows the type of model being used
  - Learned Model Parameters: The attacker may have access to the final model weights.
    - If not, model parameter stealing attacks can be used to approximate the weights.
- The goal of the attacker is to steal the hyperparameters without paying the full cost of optimizing them on the entire dataset, using an approach known as Train-Steal-Retrain.

When would this be the case?

- **MLaaS** platforms, like Amazon ML or Microsoft Azure ML, provide users with the ability to train models using proprietary techniques like hyperparameter tuning
- These platforms have optimized hyperparameters for specific models and use cases
- These platforms perform hyperparameter tuning for clients, which incurs significant computational costs and makes the client dependent on their service.

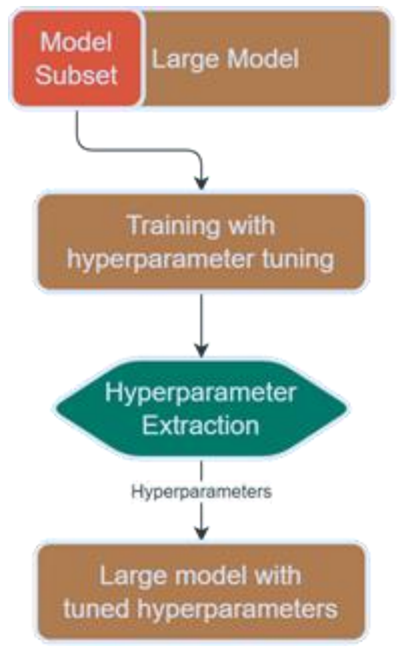
Train-Steal-Retrain

1. **Train on a Subset**: Use a small **subset of the data** to obtain **model parameters**. If the attacker has access to the fully trained original model this step can be skipped.
2. **Steal Hyperparameters**: Use the model parameters and training data to estimate the **hyperparameters**.
3. **Retrain on the Full Dataset**: Use the estimated hyperparameters to train a model on the **full dataset**, achieving high performance at a **reduced cost**.



Why is this feasible?

- Hyperparameters optimized on a small dataset can often generalize well to the full dataset
- This allows attackers to benefit from proprietary hyperparameters without incurring the full computational cost
- The attack exploits the relationship between model parameters, hyperparameters, and training data


Model Subset
Large Model
Training with hyperparameter tuning
Hyperparameter Extraction
Hyperparameters
Large model with tuned hyperparameters

Step 1: Training on a Subset

- The attacker first trains a model using a small subset of the data on the MLaaS platform.
- This subset is enough to get a reasonably accurate picture of the model's behavior and provides the weights.
- These weights are critical for the next step, where the attacker will attempt to steal the hyperparameters.

Step 2: Steal the Hyperparameters

Let’s consider a standard objective function

$$\mathcal{L}(w) = L(w; X, y) + \lambda \cdot \Omega(w)$$

The **weights** minimize the objective function, which means the gradient of the objective function with respect to w is zero or near it

$$\frac{\partial \mathcal{L}(w)}{\partial w} = \frac{\partial L(w; X, y)}{\partial w} + \lambda \frac{\partial \Omega(w)}{\partial w} = 0$$

The idea is to use this property to set up a system of equations that involve **λ**. The hyper parameter we are trying to solve for

Step 2: Steal the Hyperparameters (Contd.)

Express this in the form:

$$b + \lambda a = 0$$

Where

and

$$b = \frac{\partial L(w; X, y)}{\partial w} \qquad a = \frac{\partial \Omega(w)}{\partial w}$$

Since b and a are known, we can now solve for **λ** using the **least squares** method

$$\hat{\lambda} = -(a^T a)^{-1} a^T b$$

This provides an estimate of the hyperparameter **λ**

And thus by solving this equation, the attacker effectively **steals the hyperparameter**.

Step 3: Retrain the Full Dataset

- With the stolen hyperparameter, the attacker can now train the model on the entire dataset, achieving high performance without the cost of hyperparameter tuning.
- This allows the attacker to:
    1. Train independently without needing the MLaaS service.
    2. Use proprietary hyperparameters without incurring all the associated costs.

Does this Apply to Other Objective Functions?

- The approach we just used can be extended to other algorithms, including more complex models like support vector machines (SVMs) and neural networks.
- The general idea remains: use the gradient at the minimum to form equations involving hyperparameters.

Defenses Against this Attack?

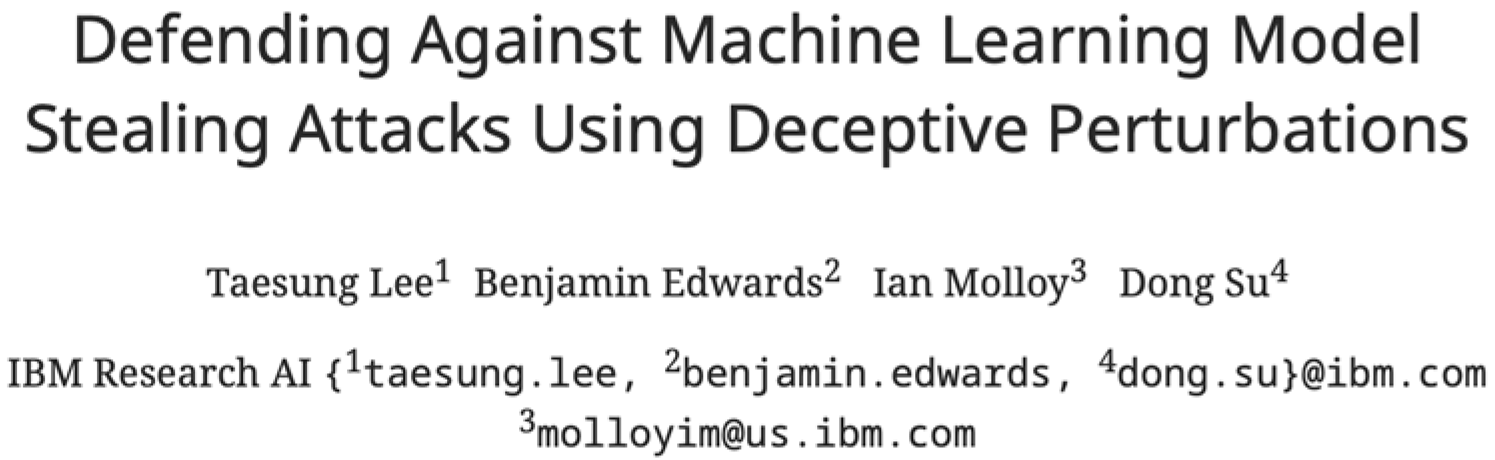
- Rounding the model parameters as a defense:
  - Rounding: Increases the estimation errors of the hyperparameters, but the attack remains effective in many cases.
- Effectiveness:
  - L2 regularization is more resistant to attacks compared to L1.
  - Loss Functions: Cross-entropy and square hinge loss provide better resistance compared to regular hinge loss

Summary

- **Hyperparameter stealing** exploits the relationship between model parameters, hyperparameters, and training data.
- The **Train-Steal-Retrain** approach allows attackers to achieve high-quality models at reduced costs.
- This attack is particularly relevant for MLaaS platforms, where hyperparameters are proprietary and expensive to compute.
- **Defenses** like rounding are partially effective, but there is a need for more robust countermeasures.

Model Stealing Defense

Defensive Perturbations



Defending Against Machine Learning Model Stealing Attacks Using Deceptive Perturbations

Taesung Lee[1] Benjamin Edwards[2] Ian Molloy[3] Dong Su[4]

IBM Research AI {[1]taesung.lee, [2]benjamin.edwards, [4]dong.su}@ibm.com
[3]molloyim@us.ibm.com

Problem: Probability Values

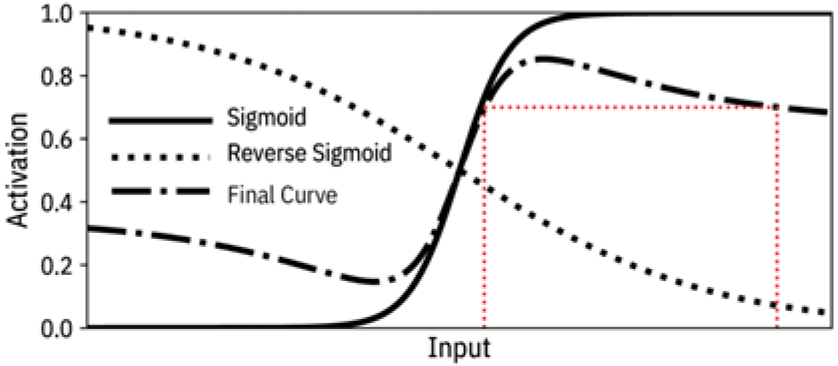
- There are many web services relating to machine learning which provide labeling for datasets.
  - These services generally provide probability values, which add to their utility for consumers.
- Introducing probability values into a black box classification model leaks valuable information which can massively increase the effectiveness of model stealing.
  - While eliminating this feature may reduce the effectiveness or increase the cost of model stealing, it also decreases the consumer value of the service.
- A proposed solution to this dilemma is to perturb these probabilities in a way that maintains functionality and accuracy for a user, but makes attackers unable to use this information to steal the model accurately or quickly.

Method: Reverse Sigmoid

$$r(y_j^i) \approx \beta(s(\gamma s^{-1}(y_j^i)) - 1/2)$$

$$\tilde{y}_j^i = \alpha^i\left(y_j^i - \beta(s(\gamma s^{-1}(y_j^i)) - 1/2)\right)$$

- The researchers here propose adding a layer which returns probabilities with high loss while preserving accuracy and maintaining the ranking of labels.
- This is done by creating a reverse sigmoid perturbation, r(y), which is normalized to produce the final probabilities.
- This function prevents simple model inversion while retaining the meaning of the probabilities.

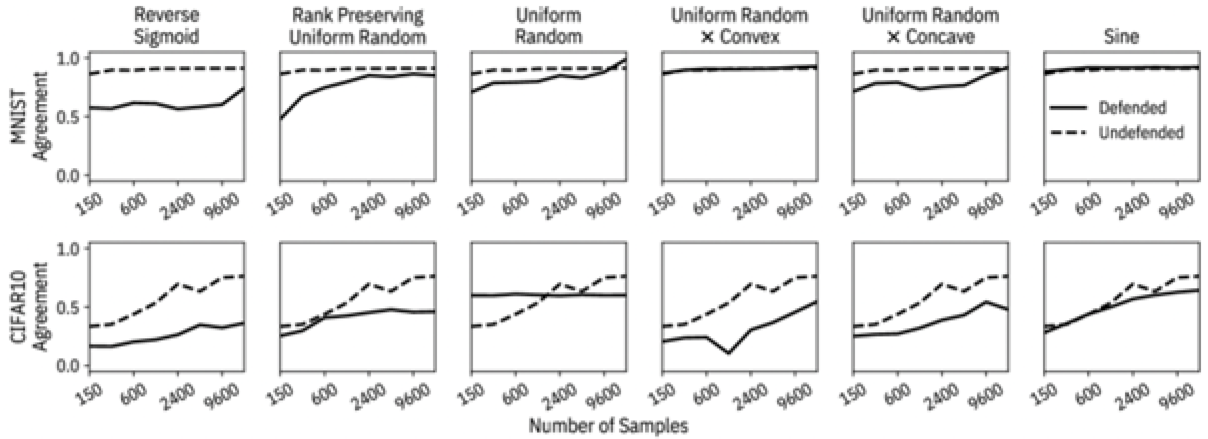


Experiments Overview

- This method was tested under two different scenarios, first, a situation in which the attacker is unaware of the perturbation, and second, under the condition of an informed attacker.
- In the first case, they compare the effectiveness of various attacks in their ability to steal the base model in both a defended and undefended case, and compare the base model with the stolen model.
- In the latter case, they attempt these same attacks in a way that is aware of the defense, and evaluate whether there is any improvement in using the probabilities at all over just using labels, which would constitute an ideal defense.
- The metrics used to evaluate this defense were agreement between models, cosine similarity, mean absolute error, KL-divergence, and accuracy.

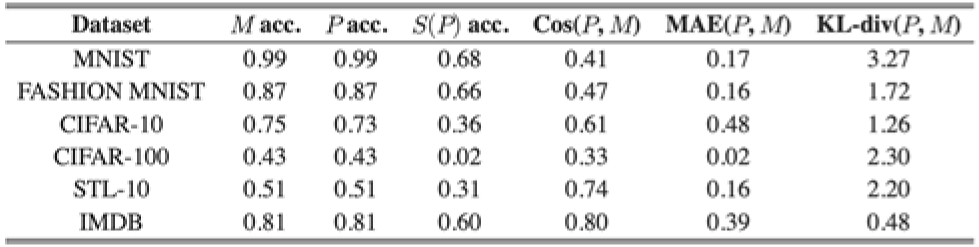
Perturbation Effectiveness

- Below are the performances of agreement under the first experiment with various perturbations as a defense, with reverse sigmoid showing substantial results compared to alternatives.
- Within an attacker budget of 19200 samples, this produced an accuracy drop greater than 20% on all datasets.

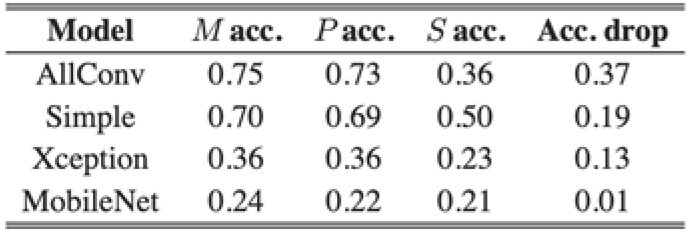


Reverse Sigmoid Effects

- The following charts represent the performance of the reverse sigmoid as a perturbation (P) in comparison to the base model (M) and the stolen model (S(P)).
- As demonstrated in the first chart, the defense provides consistent accuracy with the base model in all datasets, as the ordering between the probabilities of labels remains largely unchanged.
- In the second chart, which compares the accuracy of the base model to the effectiveness of the defense, it is demonstrated that the defense's performance is best in scenarios where the base model is more accurate, which makes it especially useful for high likelihood targets.



Dataset	M acc.	P acc.	$S(P)$ acc.	Cos(P, M)	MAE(P, M)	KL-div(P, M)
MNIST	0.99	0.99	0.68	0.41	0.17	3.27
FASHION MNIST	0.87	0.87	0.66	0.47	0.16	1.72
CIFAR-10	0.75	0.73	0.36	0.61	0.48	1.26
CIFAR-100	0.43	0.43	0.02	0.33	0.02	2.30
STL-10	0.51	0.51	0.31	0.74	0.16	2.20
IMDB	0.81	0.81	0.60	0.80	0.39	0.48

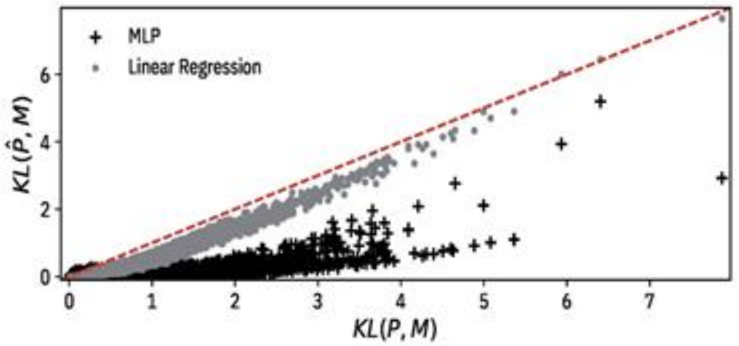


Model	M acc.	P acc.	S acc.	Acc. drop
AllConv	0.75	0.73	0.36	0.37
Simple	0.70	0.69	0.50	0.19
Xception	0.36	0.36	0.23	0.13
MobileNet	0.24	0.22	0.21	0.01

Defense Robustness

- When we assume that the attacker has information about the defense, there are various potential ways that they can adapt, such as assuming the same defensive layer in their model, using a different loss function, or inversion mapping.
- Since the defense adds ambiguity to the probabilities, regardless of the knowledge of the attacker, these methods all fall short with respect to agreement.
- In addition, the inversion mapping, while recovering some information from the base model when the magnitudes of the perturbation are low as seen in the chart, still maintains weak agreement.

Attack	Agree.
Same Defense Layer	0.10
MSE Loss	0.18
Inversion (MLP)	0.22
Argmax	0.78

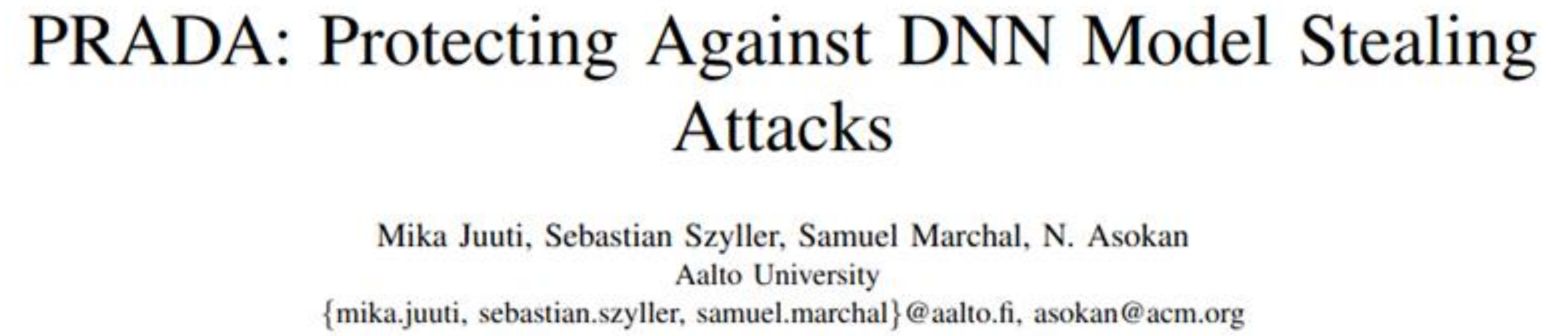


Defense Conclusion

- A successful approach for preventing attackers from abusing probability values is to perturb the outputs using a reverse sigmoid function.
- Under the case of an informed attack, the best results were actually achieved by deploying Argmax rather than an adaptation to the perturbations themselves, thus leading them to discard the defended probabilities altogether, which is a complete success for the defenders.
- This approach drastically reduces the quality of the stolen model or massively increases the costs by forcing the attacker to abandon these values and revert to using the labels exclusively.

Attack Detection

Introducing PRADA

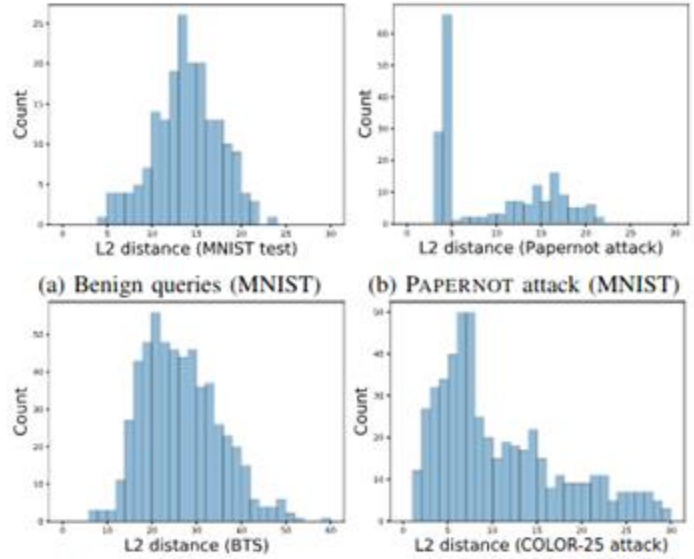


PRADA: Protecting Against DNN Model Stealing Attacks

Mika Juuti, Sebastian Szyller, Samuel Marchal, N. Asokan
Aalto University
{mika.juuti, sebastian.szyller, samuel.marchal}@aalto.fi, asokan@acm.org

Basis

- Model stealing attacks rely on making…
  - Multiple queries to a target model
  - Using samples designed to extract maximal information.
- Legitimate users will make queries that are much more random
  - The distances between them are roughly normally distributed



(a) Benign queries (MNIST) (b) PAPERNOT attack (MNIST)

(c) Benign queries (GTSRB) (d) COLOR attack (GTSRB)

Technique

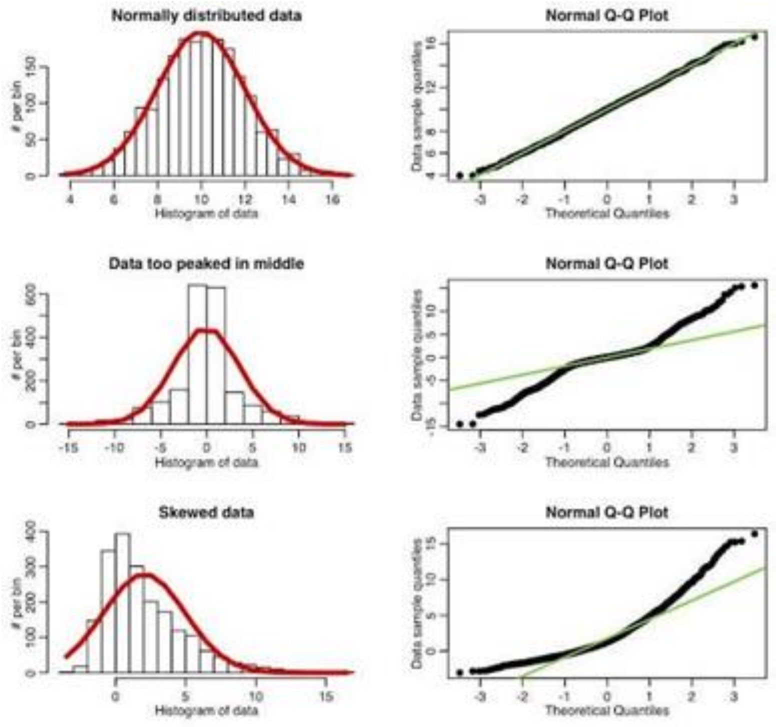
- Maintain a set of each query for each class.
- Calculate the pairwise distance for samples in the set.
- Once the number of queries is large enough, apply the Shapiro-Wilk test to measure how normally distributed the data are.
- If they are suspicious, block or rate limit the user.

$$W = \frac{\left(\sum_{i=1}^{n} a_i x_{(i)}\right)^2}{\sum_{i=1}^{n} (x_i - \overline{x})^2},$$

Shapiro-Wilk Test

Shapiro Wilk Test

- **Q-Q Plot**: A cumulative frequency plot with the x-axis scaled so that a normal distribution appears as a straight line.
  - Y-Axis: Training data value
  - X-Axis: Scaled percentile
- The Shapiro Wilk Test checks how closely the data fit the straight line.



Experiment - Attacks

The author’s measure their detector against four model extraction methods.

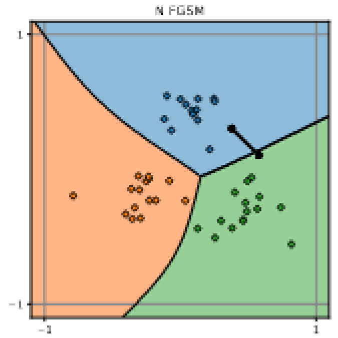
Two are pre-existing:

- Tramer et. al. - “Stealing Machine Learning Models via Prediction APIs”
- Papernot et. al. - Jacobian-based Dataset Augmentation
  - The attacker trains a candidate model on a small set of training samples classified by the target model.
  - He applies FGSM on all the training samples to generate new training examples that are closer to classification boundaries. These are the new queries.

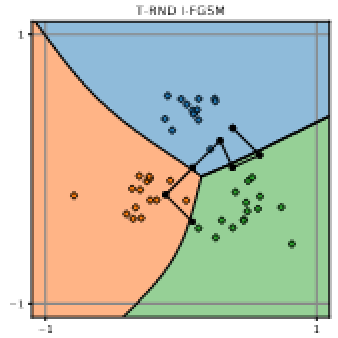
Experiment - Attacks

Two are novel:

- T-RND
  - Similar to the Papernot method, T-RND uses FGSM to generate synthetic samples.
  - However, instead of generating one un-targeted FGSM sample, it iteratively generates targeted FGSM with random targets.
  - This promotes sample variety.
- Color
  - Randomly generate synthetic samples by perturbing each color channel.



(a) Non-targeted FGSM



(b) T-RND I-FGSM

Experiment - Models

Model: CNN

Training Datasets

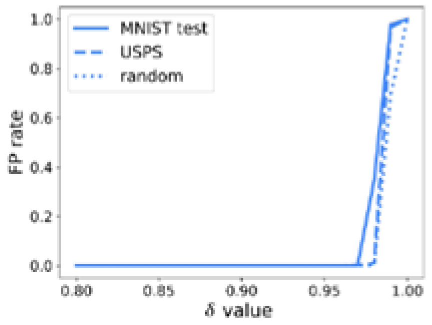
- MNIST - Handwritten digit classification
- GTSRB - German traffic sign classification

Benign Datasets

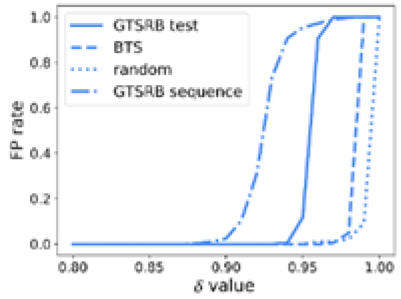
- MNIST
  - USPS Handwritten Digits
- GTSRB
  - BTS - Belgian traffic signs
  - GTSRB Sequential - Belgian traffic signs including pictures taken right after each other

Experiment - Models

- δ is the Shapiro-Wilks value above which queries are classified as suspicious.
- The sequence data are not normally distributed.
- A loose normalcy-test threshold will result in a high false positive rate.



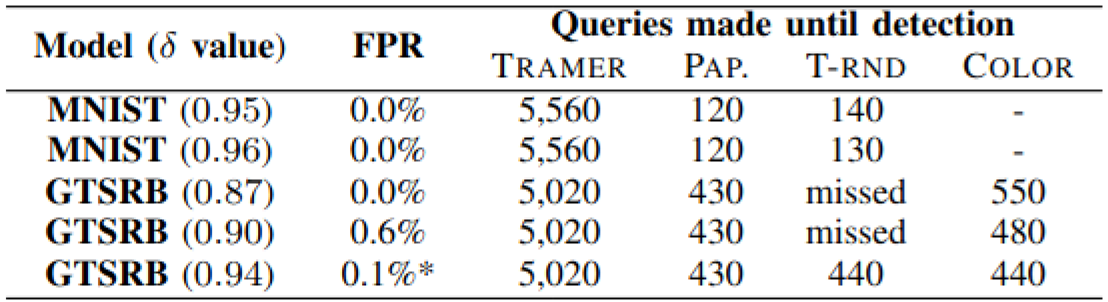
(a) MNIST model



(b) GTSRB model

Experiment - Results

- All techniques, besides T-RND, were detected shortly after the attacks switched from natural queries to synthetic queries
- They achieved this while maintaining a low false positive rate.
    - Except for row five which omitted the BTS Sequence data



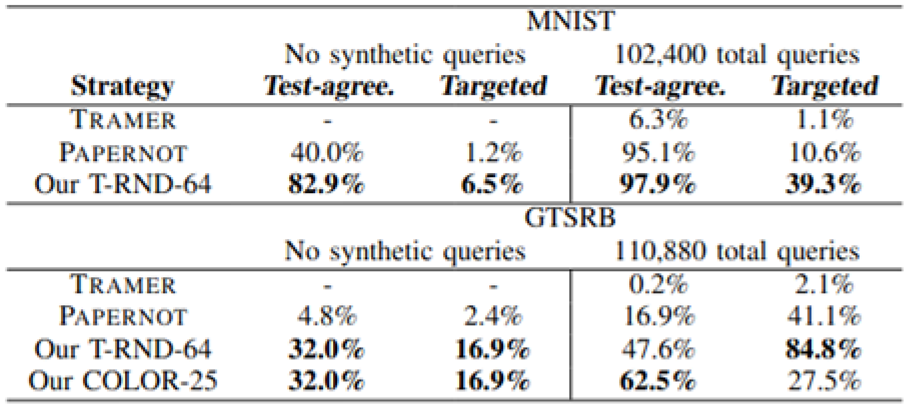
Model (δ value)	FPR	Queries made until detection			
		TRAMER	PAP.	T-RND	COLOR
MNIST (0.95)	0.0%	5,560	120	140	-
MNIST (0.96)	0.0%	5,560	120	130	-
GTSRB (0.87)	0.0%	5,020	430	missed	550
GTSRB (0.90)	0.6%	5,020	430	missed	480
GTSRB (0.94)	0.1%*	5,020	430	440	440

δ - Cutoff value for normal distribution test

FPR - False Positive Rate

*** Discarding sequence data**

Experiment -Results



	MNIST			
	No synthetic queries		102,400 total queries	
Strategy	***Test-agree.***	***Targeted***	***Test-agree.***	***Targeted***
TRAMER	-	-	6.3%	1.1%
PAPERNOT	40.0%	1.2%	95.1%	10.6%
Our T-RND-64	**82.9%**	**6.5%**	**97.9%**	**39.3%**
	GTSRB			
	No synthetic queries		110,880 total queries	
TRAMER	-	-	0.2%	2.1%
PAPERNOT	4.8%	2.4%	16.9%	41.1%
Our T-RND-64	**32.0%**	**16.9%**	47.6%	**84.8%**
Our COLOR-25	**32.0%**	**16.9%**	**62.5%**	27.5%

TABLE V: Comparative evaluation of model extraction attacks on our two datasets. Our techniques achieve significantly improved performance on both *Test-agreement* and *Targeted*.

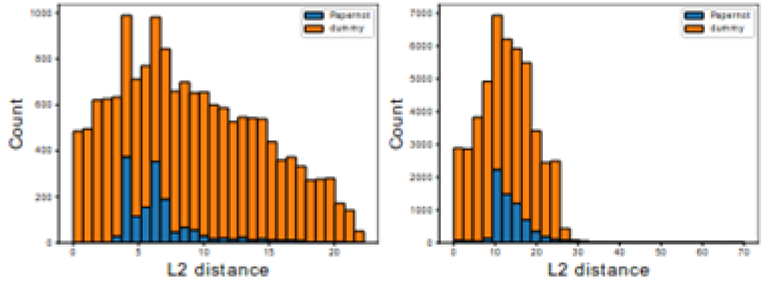
PRADA's Pros

- It makes no assumptions about the model
- It makes weak assumptions about the attack method.
- The paper also provided an attack that beat it.

Does anyone see any problems with this?

Problems with PRADA

- Threat agents can mix their malicious queries with benign ones
  - Attackers know the distribution of the data they send.
  - This requires 3x - 10x more queries.
- User queries are not always normally distributed
  - *Can anyone think of examples?*
- A blocked attacker can create a new account and query until they are blocked.
- They propose continually measuring benign samples. As sample size gets very large, Shapiro-Wilks begins to over-classify distributions as abnormal.
- The paper also provided an attack that beat it.



(a) MNIST model (b) GTSRB model

Papernot attack with λ chosen to produce a normal distribution,

How else can we detect suspicious queries?

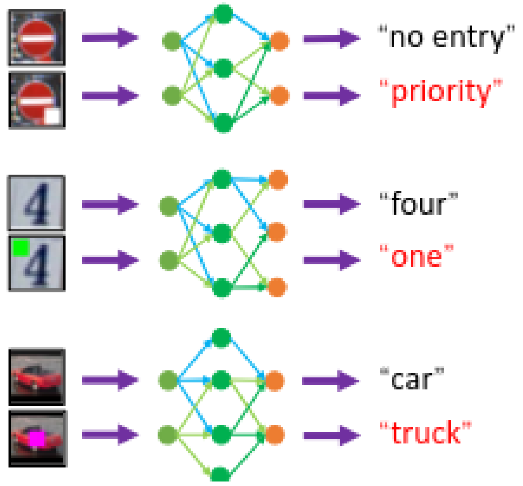
Possible Remedies

- Train a NN to detect adversarial queries (Yu et al.)
- Detect if individual queries are near decision boundaries.
- Detect if individual queries are out-of-distribution from reference data in a feature space (that the attacker has no access to).

Other Topics in Model Stealing

Watermarking

- **Goal:** Embed unique information in your model to detect if another model was stolen from yours.
- *How can we do this?*
  - Using a Backdoor Attack
    - Embed a stealable trigger in your model.
    - Check if another model has the same trigger.



BadEncoder: Backdoor Attacks to Pre-trained Encoders in Self-Supervised Learning

Side-Channel Attacks

- **Side Channel Attacks** - Using the side-effects of a machine learning model to infer details about it
- What are some possible side effects?

Model Architecture Stealing

- Many attack methods rely on knowing details about the model architecture beforehand. This is not always public.
- Prior research has demonstrated model architecture stealing attacks using
    - Model architecture prediction models
    - Side-channel attacks
- More research is needed.

Bibliography

[1] https://arxiv.org/pdf/1609.02943

[2]https://arxiv.org/pdf/1802.05351

[3] https://ar5iv.labs.arxiv.org/html/1806.00054v2

[4] https://arxiv.org/pdf/1805.02628#cite.tramer%3A2016%3Astealing

[5] https://arxiv.org/abs/1812.11720

[6] https://cs-people.bu.edu/tromer/papers/cache-joc-official.pdf

[7] https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9000972

[8] https://ieeexplore.ieee.org/document/8735505