Learning and inference in the presence of corrupted inputs

Uriel Feige, Weizmann
Yishay Mansour, MSR & Tel Aviv University
Robert Schapire, MSR
Talk Overview

• Motivating examples
• Model
• Our main results
• Sample of some results
Spam Detection

The "From:" email address is not an individual Parkland staff email address (i.e. firstInitialLastName@prrlab.ca).

Notice how there is no "To:" address. Emails sent by Parkland staff are always sent to "pripublics@lists.prrlab.ca".

Parkland IT staff will never request your password via email. As for the rest of the information requested, if we had to ask you which city you live in and what your country of residence is, then we’re in real trouble. :(

Hovering over the "From:" address with your mouse cursor shows the actual address. Emails sent by Parkland staff never come from an external email address.

In general, when you read how the email was written, think to yourself, does this read like an email that a Parkland staff person would write?

It’s also important to be aware of any click-able links within the body of the email. Never click on a link unless you’re sure the email is from a legitimate source.

Emails from Parkland staff are always signed by an individual. There is no "PRL.AB.CA Web-mail Services" department.

Reply - Reply to All - Forward - More Actions
Robust Spam Detection

• It is all a game!
  – Spammers versus detectors
Robust Spam Detection

• **It is all a game!**
  – Spammers versus detectors

• **Spammer**
  – adjust content to the detector
  – learns to fool a few detectors
Robust Spam Detection

• It is all a game!
  – Spammers versus detectors

• Spammer
  – adjust content to the detector
  – learns to fool a few detectors

• Goal:
  – classify the e-mail correctly even if the spammer can (adversarially) corrupt a few detector’s output
Network Failure
Robust Network Failure Detection
Robust Network Failure Detection

• What happens when the detectors fail?
Robust Network Failure Detection

• What happens when the detectors fail?
• How to model failures?
  – Bayesian versus worst-case
Robust Network Failure Detection

• What happens when the detectors fail?
• How to model failures?
  – Bayesian versus worst-case

• Goal:
  – Perform good failure detection
  – Overcome k-point failure
    • adversarial behavior
Adaptive model
Adaptive model

Modify to $Z \in \rho(y, x)$
Adaptive Model

- **Label**
  - $y \in \{0, 1\}$

- **Signals**
  - $x \in X$

- **Distribution**
  - $D(y, x_1, ..., x_n)$
  - known prior,
    - computable & samplable

- **Observed signals**
  - $z \in Z$
Adaptive Model

- **Label**
  - $y \in \{0,1\}$
- **Signals**
  - $x \in X$
- **Distribution**
  - $D(y, x_1, \ldots, x_n)$
  - known prior,
    - computable & samplable
- **Observed signals**
  - $z \in Z$

- **Corruption:**
  - $\rho(y,x)$: possible corruptions
    - size $m$
    - computed in poly time
  - Adversary selects $z \in \rho(y,x)$
Adaptive Model

- **Label**
  - \( y \in \{0, 1\} \)
- **Signals**
  - \( x \in X \)
- **Distribution**
  - \( D(y, x_1, \ldots, x_n) \)
  - known prior,
    - computable & samplable
- **Observed signals**
  - \( z \in Z \)

- **Corruption:**
  - \( \rho(y,x) \): possible corruptions
    - size \( m \)
    - computed in poly time
  - Adversary selects \( z \in \rho(y,x) \)

- **Example:**
  - flips one signal
    - Hamming distance 1
Adaptive Model

- **Label**
  - $y \in \{0,1\}$

- **Signals**
  - $x \in X$

- **Distribution**
  - $D(y, x_1, \ldots, x_n)$
  - known prior,
    - computable & samplable

- **Observed signals**
  - $z \in Z$

- **Corruption:**
  - $\rho(y,x)$: possible corruptions
    - size m
    - computed in poly time
  - Adversary selects $z \in \rho(y,x)$

- **Example:**
  - flips one signal
    - Hamming distance 1

- **Goal**
  - given $z$ predict $y$
Adaptive Model

• **Predictor:**
  – Given z
    • observed signals
  – Predicts y
    • label
  – Defines a policy \( \pi(z) \)
    • probabilistic

• **Adversary:**
  – Given \( x,y \) selects z
    • Corruption rule \( c(y,x) \)
Adaptive Model

• **Predictor:**
  – Given z
    • observed signals
  – Predicts y
    • label
  – Defines a policy $\pi(z)$
    • probabilistic

• **Zero sum game**

• **error**
  – fix policy $\pi$ and corruption $c$

  $$\text{error}(\pi,c) = E_{y,x}[\Pr[\pi(c(y,x)) \neq y]]$$

• **Adversary:**
  – Given $x,y$ selects $z$
    • Corruption rule $c(y,x)$
Adaptive Model

• **Predictor:**
  – Given $z_{\text{observed signals}}$
  – Predicts $y_{\text{label}}$
  – Defines a policy $\pi(z)$
    • probabilistic

• **Adversary:**
  – Given $x,y$ selects $z$
    • Corruption rule $c(y,x)$

• **Zero sum game**

• **error**
  – fix policy $\pi$ and corruption $c$
  
\[
\text{error}(\pi,c) = E_{y,x}[\Pr[\pi(c(y,x)) \neq y]]
\]

• optimal min-max error
\[
\text{error}^* = \min_{\pi} \max_c \text{error}(\pi,c)
\]
Adaptive Model

- **Predictor:**
  - Given $z$
    - observed signals
  - Predicts $y$
    - label
  - Defines a policy $\pi(z)$
    - probabilistic

- **Adversary:**
  - Given $x,y$ selects $z$
    - Corruption rule $c(y,x)$

- **Zero sum game**
  - error
    - fix policy $\pi$ and corruption $c$
      
      $$
      \text{error}(\pi,c) = E_{y,x}[Pr[\pi(c(y,x)) \neq y]]
      $$
    - optimal min-max error
      
      $$
      \text{error}^* = \min_{\pi} \max_c \text{error}(\pi,c)
      $$
    - $\varepsilon$-optimal: $\text{error}^* + \varepsilon$
Our Main Results: Inference

Setting

- Distribution D
  - Known and computable
- Possible corruptions
  - Given only implicitly
Our Main Results: Inference

**Setting**
- Distribution D
  - Known and computable
- Possible corruptions
  - Given only implicitly

**Main result**
- Theorem
  - For every distribution D
  - Given observable signal
    - Z
  - Compute a prediction for y
  - Probability of error
    - error* + ε
  - Efficient algorithm
    - poly(n) and exp(m)
Our Main Results: Inference

Setting

• Distribution D
  – Known and computable

• Possible corruptions
  – Given only implicitly

• Previous work [MTR]
  – Static setting
  – Efficient algorithm

Main result

• Theorem
  – For every distribution D
  – Given observable signal
    • Z
  – Compute a prediction for y
  – Probability of error
    • error* + ε
  – Efficient algorithm
    • poly(n) and exp(m)
Our Main Result: Learning

Setting

- Distribution D
  - Unknown and arbitrary
- Training set
  - Examples are uncorrupted
- Test set
  - Possibly corrupted by adversary
- Hypothesis class: H
  - Given an oracle for ERM for H
- Goal:
  - Select a mixture of hypothesis from H that minimize the error
Our Main Result: Learning

Main Results

Setting

• Distribution D
  – Unknown and arbitrary

• Training set
  – Examples are uncorrupted

• Test set
  – Possibly corrupted by adversary

• Hypothesis class: H
  – Given an oracle for ERM for H

• Goal:
  – Select a mixture of hypothesis from H that minimize the error

Theorem 1:

– Given a sample $S$
– Efficiently compute a mixture $h$ which is $\varepsilon$-optimal on $S$
Our Main Result: Learning

Setting

• Distribution D
  – Unknown and arbitrary
• Training set
  – Examples are uncorrupted
• Test set
  – Possibly corrupted by adversary
• Hypothesis class: H
  – Given an oracle for ERM for H
• Goal:
  – Select a mixture of hypothesis from H that minimize the error

Main Results

• Theorem 1:
  – Given a sample S
  – Efficiently compute a mixture h which is $\varepsilon$-optimal on S

• Theorem 2
  – For S: $|S| \geq \log(H/\delta)/\varepsilon^4$
  – With prob 1 - $\delta$
  – Error(h) – obsError(h) $\leq \varepsilon$
Where does it make a difference?

• Learning homogeneous hyperplanes
  – \( \text{Sign}(w^T x) \)

• Regular learning
  – \( \min_w \Pr[(w^T x)y < 0] \)

• Large Margin (SVM)
  – \( \min_w \Pr[(w^T x)y < \lambda] \)

• Adversary
  – Can set one \( x_i \) to zero
Where does it make a difference?

- Learning homogenous hyperplanes
  - \( \text{Sign}(w^T x) \)
- Regular learning
  - \( \min_w \Pr[(w^T x)y < 0] \)
- Large Margin (SVM)
  - \( \min_w \Pr[(w^T x)y < \lambda] \)
- Adversary
  - Can set one \( x_i \) to zero
- For simplicity assume
  - uniform distribution
Where does it make a difference?

• Learning homogenous hyperplanes
  – \( \text{Sign}( w^T x ) \)

• Regular learning
  – \( \min_w \Pr[(w^T x)y < 0] \)

• Large Margin (SVM)
  – \( \min_w \Pr[(w^T x)y < \lambda] \)

• Adversary
  – Can set one \( x_i \) to zero

• For simplicity assume
  – uniform distribution

• Static adversary
  \( \min_w \Pr[(w^T x)y < \max_i |w_i|] \)
Where does it make a difference?

- **Learning homogenous hyperplanes**
  - \( \text{Sign}(w^T x) \)
- **Regular learning**
  - \( \min_w \Pr[(w^T x)y < 0] \)
- **Large Margin (SVM)**
  - \( \min_w \Pr[(w^T x)y < \lambda] \)
- **Adversary**
  - Can set one \( x_i \) to zero

- **For simplicity assume**
  - uniform distribution
- **Static adversary**
  - \( \min_w \Pr[(w^T x)y < \max_i |w_i|] \)
- **Adaptive Adversary**
  - Observe \( x \) and \( w \)
    - \( \text{Sign}(w^T x_i) = y \)
    - \( \min_w \Pr[(w^T x)y < \max_i(x_i w_i)] \)
Where does it make a difference?

• Learning homogenous hyperplanes
  – $\text{Sign}(w^T x)$

• Regular learning
  – $\min_w \Pr[(w^T x)y < 0]$ 

• Large Margin (SVM)
  – $\min_w \Pr[(w^T x)y < \lambda]$ 

• Adversary
  – Can set one $x_i$ to zero

• For simplicity assume
  – uniform distribution

• Static adversary
  $\min_w \Pr[(w^T x)y < \max_i |w_i|]$

• Adaptive Adversary
  – Observe $x$ and $w$
    • $\text{Sign}(w^T_i x_i) = y$
      $\min_w \Pr[(w^T x)y < \max_i (x_i w_i)]$

• “Adaptive margin”
Adaptive Adversary

Quick overview
Modeling the setting

Corruption Graph

Interference Graph

X

Z
Modeling the setting

Corruption Graph

Interference Graph

X  Z
Modeling the setting

Corruption Graph

Interference Graph

X
Z
Modeling the setting

**Corruption Graph**

**Interference Graph**

Weighted By Dist.
Modeling the setting

Corruption Graph

Interference Graph

\[ X_b = \{ x : f(x) = b \} \]
Modeling the setting

Corruption Graph

Interference Graph

Weighted by Dist.

$X_0 = \{x : f(x) = b\}$
Modeling the setting

**Corruption Graph**

- Weighted by Dist.

**Interference Graph**

\[ X_b = \{ x : f(x) = b \} \]
Modeling the setting

Corruption Graph

Interference Graph

Weighted by distribution

\[ X_0 = \{ x : f(x) = b \} \]
Modeling the setting

Corruption Graph

Interference Graph

For the talk: Assume uniform distribution

\[ X_b = \{ x : f(x) = b \} \]
Optimal policies

Adversary
(maximum) matching

Predictor
(minimum) vertex cover

Errors $\geq |\text{Matching}|$

Errors $\leq |\text{Vertex Cover}|$
Optimal policies

• Optimal Predictor policy deterministic

• Can be computed in poly time
  – In the size of $|X|$

• What about poly in dim
  – Say $X=\{0,1\}^n$
    – cannot hope to inspect all the graph!

• Local computation algorithms.
Local Computation Algorithms
Local computation algorithms

Setting:

• **Input**
  – A graph $G(V,E)$

• **Output (implicit)**
  – A matching

• **Correctness:**
  – Feasibility
  – Defines a matching

• **Performance**
  – $(1-\epsilon)$-optimal

Local computation Algorithm (LCA)
Local computation algorithms

Setting:

• Input
  – A graph G(V,E)

• Output (implicit)
  – A matching

• Correctness:
  – Feasibility
  – Defines a matching

• Performance
  – (1-\(\varepsilon\))-optimal

Local computation Algorithm (LCA)

• User query:
  – Is edge \(e\) in the matching
Local computation algorithms

Setting:

- **Input**
  - A graph $G(V,E)$
- **Output (implicit)**
  - A matching
- **Correctness:**
  - Feasibility
  - Defines a matching
- **Performance**
  - $(1-\varepsilon)$-optimal

Local computation Algorithm (LCA)

- **User query:**
  - Is edge $e$ in the matching
- **LCA graph access**
  - Edge probes
Local computation algorithms

Setting:

• Input
  – A graph G(V,E)

• Output (implicit)
  – A matching

• Correctness:
  – Feasibility
  – Defines a matching

• Performance
  – (1-ε)-optimal

Local computation Algorithm (LCA)

• User query:
  – Is edge e in the matching

• LCA graph access
  – Edge probes

• LCA Replies to queries
  – Logarithmic time
  – No pre-computation/storage
Local computation algorithms

Setting:

• Input
  – A graph G(V,E)
• Output (implicit)
  – A matching
• Correctness:
  – Feasibility
  – Defines a matching
• Performance
  – (1-\(\varepsilon\))-optimal

Local computation Algorithm (LCA)

• User query:
  – Is edge \(e\) in the matching
• LCA graph access
  – Edge probes
• LCA Replies to queries
  – Logarithmic time
  – No pre-computation/storage
• Known LCA Algo for matching
  – [MV13, EMR14]
  – Poly-log time
From Matching to Vertex Cover

• Classical:
  – map maximum matching to minimum vertex cover

• Challenge:
  – translate approximate maximum matching to approximate minimum vertex cover
    • Using a local computation algorithm.

• THEOREM: There is a randomized algorithm that outputs a vertex cover of size at most $(1+\epsilon)\OPT$
Conclusion

• Robust probabilistic inference
  – Static adversary
  – Adaptive adversary
  – efficient algo.
  – connections to graph properties.

• local algorithms
  – surprising application
  – Natural domain

• Learning
  – Efficient learning
    • Given clean input
  – Many challenges
Thank You