Reliable and Interpretable Artificial Intelligence

ETH Zurich
Fall 2017
So Far

Foundations of probabilistic programming

Opportunity

Phrase practical problems in the framework of probabilistic programming

Next

Synthesis of probabilistic privacy enforcement
Motivation: Processing Private Data

Public output reveals information about confidential input. We want to restrict the amount of revealed information in some way.
Example: Genomic Data

Each patient has a pair of a red or green gene
Carol is a child of Alice and Bob
Example: Genomic Data

Each patient has a pair of a red or green gene

Carol is a child of Alice and Bob
Example: Genomic Data

Each patient has a pair of a red or green gene
Carol is a child of Alice and Bob

Eve (medical researcher)
Example: Censor Bob’s data

Input (confidential) → Program (censors data) → Output (public)
Example: Censor Bob’s data

Input (confidential)

Program (censors data)

Output (public)

Can Eve learn anything about Bob’s genes?
Example: Censor Bob’s data
Example: Censor Bob’s data

Carol inherits her genes from Alice and Bob
Example: Censor Bob’s data

Bob must have at least one red gene

Carol inherits her genes from Alice and Bob
Question:
How can we reason about how much Eve learns?
Bayesian Inference

Initial attacker belief

Revised attacker belief

Prior

$P(I = i)$

Query

$P(O = o | I = i)$

Joint Prior

$P(I = i, O = o)$

Output $o$
Privacy Policies and Verification

**Given**: Attacker belief $\delta$, program $\pi$, and privacy policy $\Phi$.

**Check**: Could running the program $\pi$ violate the policy $\Phi$?

\[ \Phi \equiv \forall o. P(I \in S | O = o) \in [a, b] \]

**Secret**: $S \subseteq I$: (An event)

**Belief bound**: $[a, b] \subseteq [0, 1]$

In general: Multiple policies $\Phi_1, \ldots, \Phi_k$. 
Privacy Policies and Verification

**Given**: Attacker belief $\delta$, program $\pi$ and privacy policy $\Phi$.

**Check**: Could running the program $\pi$ violate the policy $\Phi$?

$$\Phi \equiv \forall o. P(I \in S | O = o) \in [a, b]$$

- **Secret $S \subseteq I$**: An event
- **Belief bound**: $[a, b] \subseteq [0, 1]$

In general: Multiple policies $\Phi_1, \ldots, \Phi_k$. 

**Question**: Why do we need to check for all inputs?
Example: Counting Red Alleles

Input  
(confidential)

Program  
(counts red genes)

Policy: \( \forall i, o. P(\text{genes}(i) = [\text{red}, \text{red}]) | O = o \in [0, 0.75] \)

Question: Can Eve run this program?

Output (public)
Example: Counting Red Alleles

Policy: \( \forall i, o. P(\text{genes}(i) = [\text{red}, \text{red}] | O = o) \in [0, 0.75] \)

Question: Can Eve run this program?

NO. (e.g. \( o = 6 \) reveals all genes)
Example: Counting Red Alleles

Policy: $\forall i, o. P(\text{genes}(i) = [\text{\textcolor{red}{R}}, \text{\textcolor{red}{R}}] \mid O = o) \in [0, 0.75]$

Question: Can Eve run this program?

NO. (e.g. $o = 6$ reveals all genes)

How can Eve adapt her program?
Example: Counting Red Alleles

**Policy:** $\forall i, o. P(\text{genes}(i) = [\text{red}, \text{red}] \mid O = o) \in [0, 0.75]$  

**Question:** Can Eve run this program?  

**NO. (e.g. $o = 6$ reveals all genes)**

How can Eve adapt her program?

Use program synthesis to adapt the program automatically.
def query(patient: Patient[]):
    numRed := 0;
    for i in [0..3]:
        for j in [0..2]:
            if patient[i].red[j]:
                numRed += 1;
    return numRed;
Policy: $\forall i, o. P(\text{genes}(i) = [\text{red}, \text{red}] \mid O = o) \in [0, 0.75]$

```python
def query(patient: Patient[]):
    numRed := 0;
    for i in [0..3] {
        for j in [0..2] {
            if patient[i].red[j] {
                numRed += 1;
            }
        }
    }
    return numRed;
```

\[ P(\text{genes}(i) = (r, r) \mid o = 6) = 1 \]
Repairing the Program

Policy: \( \forall i, o. P(\text{genes}(i) = \text{[ ]}, \text{[ ]}) \mid O = o) \in [0, 0.75] \)

```python
def query(patient: Patient[]){
    numRed := 0;
    for i in [0..3]{
        for j in [0..2]{
            if patient[i].red[j]{
                numRed += 1;
            }
        }
    }
    if numRed in [5, 6] { 
        return pick([5, 6]);
    }
    return numRed;
}
```
Repairing the Program

Policy: ∀i, o. P(genes(i) = \([\text{red}, \text{red}]\) | O = o) ∈ [0, 0.75]

```python
def query(patient: Patient[]):
    numRed := 0;
    for i in [0..3]:
        for j in [0..2]:
            if patient[i].red[j]:
                numRed += 1;
    if numRed in [5, 6]:
        return pick([5, 6]);
    return numRed;
```

Eve

\[ P(\text{genes}(c) = (r, r) | o = 5) = 1 \]
Repairing the Program

Policy: \( \forall i, o. P(\text{genes}(i) = [\text{\red{}}, \text{\green{}}] | O = o) \in [0, 0.75] \)

```python
def query(patient: Patient[]):
    numRed := 0;
    for i in [0..3]{
        for j in [0..2]{
            if patient[i].red[j]{
                numRed += 1;
            }
        }
    }
    if numRed in [4, 5, 6] {  
        return pick([4, 5, 6]);
    }
    return numRed;
```
def query(patient: Patient[]):
    numRed := 0;
    for i in [0..3):
        for j in [0..2):
            if patient[i].red[j]:
                numRed += 1;
    if numRed in [4, 5, 6] {
        return pick([4, 5, 6]);
    } else {
        return numRed;
    }

Policy: $\forall i, o. P(\text{genes}(i) = [\text{red}, \text{red}] | O = o) \in [0, 0.75]$
Our Approach: Synthesis of Enforcement

Program $\pi$ → SPIRE → Policy-compliant program $\pi'$

Attacker belief $\delta$

Policies $\Psi$
Implementation using Probabilistic Programs

Attacker belief $\delta$

```python
1 def patient():
2     return Patient([Bern(0.77), Bern(0.77)])
3 }
4 def child(a: Patient, b: Patient):
5     allele := shuffle([a.allele[Bern(1/2)], b.allele[Bern(1/2)]]);
6     return Patient(allele);
7 }
8 def prior():
9     (alice, bob) := (patient(), patient());
10    carol := child(alice, bob);
11    patients := [alice, bob, carol];
12    return patients;
```
Implementation using Probabilistic Programs

**Query $\pi$**

```python
1 def query(patient: Patient[]){
2     numRed := 0;
3     for i in [0..3]{
4         for j in [0..2]{
5             if patient[i].red[j]{
6                 numRed += 1;
7             }
8         }
9     return numRed;
10 }
```

12/25
Implementation using Probabilistic Programs

Secrets $\Psi$

```python
1 def secret(i, patients: Patient[]): // interval: [0, 0.75]
2     patient := patients[i];
3     return patient.allele[0] == RED &&
4         patient.allele[1] == RED;
5 }
```
Implementation using Probabilistic Programs

Inference Query: \( P(O = o) \)

```python
1 def outputProb(o):
2     input := prior();
3     output := query(input);
4     return output == o;
5 }
```

Inference Query: \( P(I \in S | O = o) \)

```python
1 def secretProb(i,o):
2     input := prior();
3     output := query(input);
4     observe(output == o);
5     return secret(i,input);
6 }
```
Privacy Enforcement

Enforcement $\xi$ is an equivalence relation over $O$ such that

$$\forall o. P(I \in S \mid O \in [o]_\xi) \in [a, b]$$

Intuition: Only report $[o]_\xi$ instead of $o$. Conflate outputs.
Notions of Optimality

Permissiveness
Permissiveness of enforcement $\xi$ is $|O/\xi|$ (Number of equivalence classes.)

Precision
Precision of enforcement $\xi$ is $|\{o \in O \mid |[o]_{\xi}| = 1\}|$ (Number of equivalence classes of size 1.)
Complexity Results

**Given:** Probabilities $P(O = o), P(I \in S \mid O = o)$ for all $o$

**Want:** Enforcement $\xi$ ($\forall o. P(I \in S \mid O \in [o]_{\xi}) \in [a,b]$)

**Permissiveness**

**Theorem:** Synthesis of optimally permissive enforcement $\xi$ is NP-equivalent (NP-hard and NP-easy).

**Precision**

**Theorem:** Synthesis of optimally precise enforcement $\xi$ of a single policy is possible in $O(n \log n)$ time ($n = |O|$).
Optimally Permissive Enforcement with SMT

Probabilistic program $\pi$: 

Attacker belief $\delta$: 

Privacy policy $\Phi$: 

Probabilities $P^\pi_\delta(\cdot)$ and $P_\delta^\pi(I \in S_i | O = \cdot)$:

$P_\delta^\pi(O = 0) = \frac{29}{64}$  
$P_\delta^\pi(I \in S_1 | O = 0) = \frac{7}{29}$  
$P_\delta^\pi(I \in S_2 | O = 0) = \frac{28}{29}$  
$P_\delta^\pi(O = 1) = \frac{1}{16}$  
$P_\delta^\pi(I \in S_1 | O = 1) = \frac{1}{4}$  
$P_\delta^\pi(I \in S_2 | O = 1) = \frac{5}{16}$  
$P_\delta^\pi(O = 2) = \frac{11}{64}$  
$P_\delta^\pi(I \in S_1 | O = 2) = \frac{6}{11}$  
$P_\delta^\pi(I \in S_2 | O = 2) = \frac{4}{11}$  
$P_\delta^\pi(O = 3) = \frac{1}{16}$  
$P_\delta^\pi(I \in S_1 | O = 3) = \frac{3}{4}$  
$P_\delta^\pi(I \in S_2 | O = 3) = 0$

SMT constraints / Objective function:

$\psi_{\text{assert}} \equiv \psi_{\text{range}} \land \psi_{\text{bounds}}$

$\psi_{\text{range}} \equiv \bigwedge_{i=1}^4 C_i \geq 1 \land C_i \leq 4$

$\psi_{\text{bounds}} \equiv (\bigwedge_{i=1}^4 p_1^i \in [0.1, 0.5]) \land (\bigwedge_{i=1}^4 p_2^i \in [0.5, 0.9])$

$p_1^i = \frac{[C_1 = i] \cdot \frac{1}{16} + [C_2 = i] \cdot \frac{5}{64} + [C_3 = i] \cdot \frac{3}{32} + [C_4 = i] \cdot \frac{3}{64}}{[C_1 = i] \cdot \frac{29}{64} + [C_2 = i] \cdot \frac{5}{64} + [C_3 = i] \cdot \frac{11}{64} + [C_4 = i] \cdot \frac{1}{16}}$

$p_2^i = \frac{[C_1 = i] \cdot \frac{7}{16} + [C_2 = i] \cdot \frac{1}{4} + [C_3 = i] \cdot \frac{1}{16} + [C_4 = i] \cdot 0}{[C_1 = i] \cdot \frac{29}{64} + [C_2 = i] \cdot \frac{5}{64} + [C_3 = i] \cdot \frac{11}{64} + [C_4 = i] \cdot \frac{1}{16}}$

$\psi_{\text{obj}} = \maximize((C_1 = 1 \lor C_2 = 1 \lor C_3 = 1 \lor C_4 = 1) 
+ \cdots + [C_1 = 4 \lor C_2 = 4 \lor C_3 = 4 \lor C_4 = 4])$

$M := \text{MAX}(\psi_{\text{assert}}, \psi_{\text{obj}})$

Model: $M = \{C_1 \mapsto 1, C_2 \mapsto 2, C_3 \mapsto 1, C_4 \mapsto 2\}$

$\xi := \ker(M)$

Equivalence classes: $O/\xi = \{(0, 2), \{1, 3\}\}$
Greedy Heuristic for Permissive Enforcement

Pick most violating class.
Select candidate to merge.
Merge, repeat.
Greedy Heuristic for Permissive Enforcement

Pick most violating class.
Select candidate to merge.
Merge, repeat.
Greedy Heuristic for Permissive Enforcement

Pick most violating class.
Select candidate to merge.
Merge, repeat.
Greedy Heuristic for Permissive Enforcement

Pick most violating class.
Select candidate to merge.
Merge, repeat.
Optimal Algorithm for Precise Enforcement

Recall: Want to maximize number of singleton classes.
Optimal Algorithm for Precise Enforcement

Recall: Want to maximize number of singleton classes.

**Algorithm**

- Join all violating classes into class $C$
Optimal Algorithm for Precise Enforcement

Recall: Want to maximize number of singleton classes.

**Algorithm**

- Join all violating classes into class $C$

- If non-violating, done. Otherwise wlog, $P(S \mid o \in C) > b$
Optimal Algorithm for Precise Enforcement

Recall: Want to maximize number of singleton classes.

**Algorithm**

- Join all violating classes into class $C$

- If non-violating, done. Otherwise wlog, $P(S \mid o \in C) > b$

- Need to merge more outputs into $C$ such that

$$P(I \in S \mid o \in C) = \frac{\sum_{o \in C} P(I \in S \mid O = o) \cdot P(O = o)}{\sum_{o \in C} P(O = o)} \leq b$$
Optimal Algorithm for Precise Enforcement

Recall: Want to maximize number of singleton classes.

**Algorithm**

- Join all violating classes into class $C$

- If non-violating, done. Otherwise wlog, $P(S \mid o \in C) > b$

- Need to merge more outputs into $C$ such that

  $$P(I \in S \mid o \in C) = \frac{\sum_{o \in C} P(I \in S \mid O = o) \cdot P(O = o)}{\sum_{o \in C} P(O = o)} \leq b$$

- Sort by contribution, pick smallest first, merge into $C$
System: SPIRE

Implementation: http://www.srl.inf.ethz.ch/probabilistic_security

Approach requires exact probabilistic inference. Tools used:

PSI SOLVER

(Exact probabilistic inference.)

z3

(NP oracle.)
Summary

Attacker belief $\delta$

Policies $\Psi$

Program $\pi$

SPIRE

Policy-compliant program $\pi'$
Next Time

Details on the PSI Solver

Useful Links

Synthesis of Probabilistic Privacy Enforcement

Dynamic Enforcement of Knowledge-based Security Policies using Probabilistic Abstract Interpretation