Machine Learning and Programming Languages

Martin Vechev
Software Reliability Lab
Department of Computer Science, ETH Zurich
@SRL: Two Directions

• Machine Learning based programming tools

• Probabilistic programming
Learning and probabilistic models based on Big Data have revolutionized entire fields

Natural Language Processing (e.g., machine translation)

Computer Vision (e.g., image captioning)

Medical Computing (e.g., disease prediction)

Can we bring this revolution to programmers?
Vision

Learning from large datasets of programs

Programming Task -> Statistical Programming Tool -> Solution

Probabilistic model

number of repositories

15 million repositories

Billions of lines of code

High quality, tested, maintained programs

last 5 years
Probabilistic Learning

Probabilistically likely solutions to problems impossible to solve otherwise

Publications

- Statistical Deobfuscation of Android Applications, ACM CCS’16
- Probabilistic Mode for Code with Decision Trees, ACM OOPSLA’16
- PHOG: Probabilistic Mode for Code, ACM ICML’16
- Learning Programs from Noisy Data, ACM POPL’16
- Predicting Program Properties from “Big Code”, ACM POPL’15
- Code Completion with Statistical Language Models, ACM PLDI’14
- Machine Translation for Programming Languages, ACM Onward’14

Publicly Available Tools

- [http://JSNice.org](http://JSNice.org) statistical de-obfuscation
- [http://Nice2Predict.org](http://Nice2Predict.org) structured prediction framework

Camera camera = Camera.open();
camera.setDisplayOrientation(90);
camera.unlock();
SurfaceHolder holder = getHolder();
holder.addCallback(this);
holder.setType(SurfaceHolder.STP);
MediaRecorder r = new MediaRecorder();
r.setCamera(camera);
r.setAudioSource(MediaRecorder.AS);
r.setVideoSource(MediaRecorder.VS);
r.setOutFormat(MediaRecorder.MPEG4);

Statistical language models
Recurrent neural networks
+ Typestate analysis
  Alias analysis
Programming Language Translation

[Phrase-based statistical translation of programming languages, ACM Onward 2014]

C#  Java  Translate

Console.WriteLine("Hi");
...

System.out.println("Hi");
...

Phrase-based Statistical Machine Translation

+  

Prefix Parsing of Context-Free Grammars
Program de-obfuscation

[Predicting program properties from Big Code, ACM POPL 2015]

```javascript
function FZ(e, t) {
    var n = [];
    var r = e.length; var i = 0;
    for (; i < r; i += t) if (i + t < r) n.push(e.substring(i, i + t)); else n.push(e.substring(i, r));
    return n;
}

function chunkData(str, step) {
    var colNames = [];
    var len = str.length;
    var i = 0;
    for (; i < len; i += step)
      if (i + step < len) colNames.push(str.substring(i, i + step));
      else colNames.push(str.substring(i, len));
    return colNames;
}
```
This Page Amsterdam @thispage AMS · Jul 16
Do you write ugly JavaScript code? Not to worry. JSNice will make it look like you are a Superstar coder. Yai! - buff.ly/1HR4JL7

Ingvar Stepanyan @RReverser · Aug 6
JSNice became my must-have tool for code deobfuscation.

Brevity @seekbrevity · Jul 23
JSNice is an amazing tool for de-minifying #javascript files. JSNice.org, it's great for #learning and reverse engineering.

Alvaro Sanchez @alvasvi · Jun 10
This is gold. Statistical renaming, Type inference and Deobfuscation. jsnice.org

Alex Vanston @mvdot · Jun 7
I've been looking for this for years: JS NICE buff.ly/1pQ5qfr #javascript #unminify #deobfuscate #makeItReadable

Kamil Tomšik @cztomsk · Jun 6
tell me how this works!
de-minify #jquery #javascript incl. args, vars & #jsdoc impressive! jsnice.org
# Dimensions: Probabilistic Learning from Big Code

<table>
<thead>
<tr>
<th>Applications</th>
<th>Code completion</th>
<th>Program synthesis</th>
<th>Feedback generation</th>
<th>Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Deobfuscation</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Intermediate Representation | Sequences (sentences) | Translation Table | Graphical Models (CRFs) | Feature Vectors |
|                            | Trees               |                   |                     |               |

| Analyze Program (PL) | typestate analysis | control-flow analysis | alias analysis |
|                      | scope analysis     |                        |               |

| Train Model (ML) | Neural Networks | SVM | Structured SVM | Pseudo-Likelihood |
|                 | N-gram/back-off |     |               |                   |
|                 |                  | PCFG|               |                   |

| Query Model (ML) | \[ \text{argmax} \ P(y \mid x) \]  
|                 | \[ y \in \Omega \] | Greedy MAP inference |

More information and tutorials at:

Key Challenge

Existing Programs  Learning  Model

Widely Applicable  Efficient Learning  High Precision  Explainable Predictions

Probabilistic Model
Representing Programs as Trees

**JavaScript expression:**

```javascript
elem.notify({
    position: 'top',
    autoHide: false,
    delay: 100
});
```

**Tree:**
Probabilistic Context Free Grammars (PCFGs)

\[ N: \text{non-terminal symbols} \quad \Sigma: \text{terminal symbols} \quad S \in N: \text{start symbol} \]

\[ R \text{ is a finite set of production rules } \quad \alpha \rightarrow \beta_1 \ldots \beta_n \]

\[ \alpha \in N \quad \beta_i \in (N \cup \Sigma) \]

\[ q: R \rightarrow \mathbb{R}^{(0,1)} \text{ is a conditional probability of choosing a production rule} \]

**Training**

\[ q(\alpha \rightarrow \beta_1 \ldots \beta_n) \approx \frac{\#(\alpha \rightarrow \beta_1 \ldots \beta_n)}{\#(\alpha)} \]

\[ \forall \alpha \sum_{\alpha \rightarrow \beta_1 \ldots \beta_n \in R} q(\alpha \rightarrow \beta_1 \ldots \beta_n) = 1 \]

valid probability distribution

Existing Dataset \[ \rightarrow \]

PCFG
PCFG

\[ \alpha \rightarrow \beta_1 \cdots \beta_n \]

\[ P \]

Property \rightarrow x \quad 0.05

Property \rightarrow y \quad 0.03

Property \rightarrow \text{promise} \quad 0.001
PHOG: Generalizing PCFG

PCFG
\( \alpha \rightarrow \beta_1 \ldots \beta_n \)

PHOG
\( \alpha[\gamma] \rightarrow \beta_1 \ldots \beta_n \)

<table>
<thead>
<tr>
<th>Property</th>
<th>( P )</th>
<th>Property[\text{reject, promise}]</th>
<th>( P )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \rightarrow x )</td>
<td>0.05</td>
<td>( \rightarrow \text{promise} )</td>
<td>0.67</td>
</tr>
<tr>
<td>( \rightarrow y )</td>
<td>0.03</td>
<td>( \rightarrow \text{notify} )</td>
<td>0.12</td>
</tr>
<tr>
<td>( \rightarrow \text{promise} )</td>
<td>0.001</td>
<td>( \rightarrow \text{resolve} )</td>
<td>0.11</td>
</tr>
</tbody>
</table>
PHOG

Production Rules:
\[ \alpha[\gamma] \rightarrow \beta_1 \ldots \beta_n \]

Function:
\[ f: \text{AST} \rightarrow \gamma \]

Key Idea:
Parametrize the grammar by a function used to dynamically obtain the context

“...All problems in computer science can be solved by another level of indirection...”
-- David Wheeler
PHOG

Production Rules:
\[ \alpha[\gamma] \rightarrow \beta_1 \ldots \beta_n \]

Function:
\[ f: \text{AST} \rightarrow \gamma \]
What are these functions?

in general: can be unrestricted programs (Turing complete)

so far: language to iterate over trees, accumulate context

\[
T\text{Cond} ::= \varepsilon \mid \text{WriteOp } T\text{Cond} \mid \text{MoveOp } T\text{Cond}
\]

\[
\text{MoveOp} ::= \text{Up, Left, Right, DownFirst, DownLast, NextDFS, PrevDFS, NextLeaf, PrevLeaf,PrevNodeType, PrevNodeValue, PrevNodeContext}
\]

\[
\text{WriteOp} ::= \text{WriteValue, WriteType, WritePos}
\]
Expressing functions

$$\varphi \leftarrow \varphi \cdot 19$$
Function Evaluation: Example

Query

```
elem.notify(
  ...,
  ...
  {
    position: 'top',
    hide: false,
    ?
  }
);
```

\[ F \]

- Left
- WriteValue
- Up
- WritePos
- Up
- DownFirst
- DownLast
- WriteValue

\[ \gamma \]

- {}  
- {hide}
- {hide}
- {hide, 3}
- {hide, 3}
- {hide, 3}
- {hide, 3, notify}

\{ Previous Property, Parameter Position, API name \}
Learning a PHOG

Existing Dataset $D$

### TCond Language

- **TCond** ::= $\varepsilon$ | WriteOp TCond | MoveOp TCond
- **MoveOp** ::= Up, Left, Right, ...
- **WriteOp** ::= WriteValue, WriteType, ...

### Synthesis:
- **practically intractable**

### Key Idea:
- **iterative synthesis on fraction of examples**, [POPL’16]

\[ f_{best} = \arg \min_{f \in TCond} \text{cost}(D, f) \]
PHOG: Key Ideas

PHOG: Probabilistic Mode for Code, ACM ICML’16
P.Bielik, V.Raychev, M.V.

Parametrize grammar by a function $F : \rightarrow \gamma$

Synthesize $F$ from Dataset $D$

Apply $F$ to a new program

$\alpha \left[ \gamma \right] \xrightarrow{0.02} \beta_1 \ldots \beta_n$
Current and Future Research

Learning Reasoning Engines

Beyond code (e.g., NLP)

New Models of Code: Neural + PHOG

New AI Programming Assistants
Learning Points-To Analysis for JavaScript

\[ v1 = v2 \quad \text{[Assignment]} \]

\[ \text{function isBig(v) \{} \]
\[ \quad \text{return } v < \text{this}.length \quad \text{[This]} \]
\[ \text{\}} \]
\[ [12, 5].\text{filter(isBig)}; \]

\[ \text{VarPtsTo(v2, h)} \]
\[ \text{Assign(v1, v2)} \]
\[ \text{VarPtsTo(v1, h)} \]

\[ \text{VarPtsTo(“global”, h)} \]
\[ \text{checkIfInsideMethodCall} \]
\[ \text{checkMethodCallName} \]
\[ \text{checkReceiverType} \]
\[ \text{checkNumberOfArguments} \ldots \]
\[ \text{VarPtsTo(this, h)} \]

\textbf{does not handle these:}

- Function.prototype
  - call()
  - apply()
- Array.prototype
  - map()
  - some()
  - forEach()
  - every()
  - filter()
- JSON
  - stringify()

\textbf{Goal: Can we learn the production rules?}

\[ \text{VarPtsTo(v2, h)} \]
\[ v2 = f(v1) \]
\[ \text{VarPtsTo(v1, h)} \]
How do I get started?

Reading:

**Learning from Large Codebases, PhD Thesis, ETH Zurich, 2016**

Dagstuhl Seminar on Big Code Analytics, Nov 2015
- ML, NLP, PL, SE
- Link to materials, people in the general area

http://learningfrombigcode.org
- data sets, tools, challenges

http://nice2predict.org
- open source, online demo, build your own tool
@SRL: Two Directions

- Machine Learning based programming tools
- Probabilistic programming
Probabilistic Programming

**Promise:** simplifies the construction and querying of probabilistic models, even by non-experts.

**Applications:** reliable machine learning, vision, robotics, networks, security, cyber-physical systems, etc.
Probabilistic Programs
Express a probabilistic model

```python
def main() {
p := Uniform(0,1);
r := [1,1,0,1,0];

for i in [0..r.length] {
    observe(Bernoulli(p) == r[i]);
}
return p;
}
```

PDF(p) = (60p^2 -120p + 60) *
[-1 + p ≤ 0] * [-p ≤ 0] * p^3

Can contain a mixture of Discrete and Continuous Distributions
Ultimate Goal

Automatically and exactly answer queries on the distribution represented by the program

Example queries: probabilities, expectations

Hard problem: complex programs, mixture of discrete and continuous distributions, etc.
Most Existing Work: Approximate

Exact (solid) and Infer.NET (dashed)

Heuristic solutions, no guarantees
Our Work: **PSI**

PSI: Exact Symbolic Inference for Probabilistic Programs, CAV’16

**PSI** \(\approx\) **SAT/SMT** for probabilistic programming

http://psisolver.org/
def max(a,b) {
    r := a;
    if b > r { r = b; }
    return r;
}
def main() {
    x := Uniform(0,1);
    y := Gauss(0,1);
    z := Uniform(0,1);
    r := max(x,max(y,z));
    observe(x < 0.75);
    if Bernoulli(1/2)
        { assert(r < 0.9); }
    return r + UniformInt(1,3);
}
def max(a, b):
    r := a;
    if b > r:
        r := b;
    return r;

def main():
    x := Uniform(0, 1);
    y := Gauss(0, 1);
    z := Uniform(0, 1);
    r := max(x, max(y, z));
    observe(x < 0.75);
    if Bernoulli(1/2):
        assert r < 0.9;
    return r + UniformInt(1, 3);
def max(a,b) {
    r := a;
    if b > r { r = b; }
    return r;
}
def main() {
    x := Uniform(0,1);
    y := Gauss(0,1);
    z := Uniform(0,1);
    r := max(x,max(y,z));
    observe(x < 0.75);
    if Bernoulli(1/2) {
        assert(r < 0.9);
    }
    return r + UniformInt(1,3);
}

Can compute various queries on the PDF: e.g., expectations, marginal probabilities

\[ E[\text{result}] \approx 2.6929 \]
\[ Pr[\text{error}] \approx 0.132827 \]
PSI: Ingredients and Flow

Source Language

Ψ

Symbolic Domain

Formula χ

Program P

interpret

Symbolic Simplification

Formula χ'
**PSI: Symbolic Domain**

\[ e \in E ::= \quad x \mid e \mid \pi \mid 0 \mid 1 \mid 2 \mid \ldots \]

\[ \mid \log(e) \mid -e \mid e_1 + \ldots + e_n \mid e_1 \ldots \ldots e_n \mid e_1 e_2 \]

\[ \mid \delta(e) \mid [e_1 = e_2] \mid [e_1 \leq e_2] \mid [e_1 \neq e_2] \mid [e_1 < e_2] \]

\[ \mid \int_{\mathbb{R}} \! dx \; e[x] \mid \sum_{x \in \mathbb{Z}} e[x] \mid \varphi(e_1, \ldots, e_n) \]

\[ \mid (d/dx)^{-1}[e^{-x^2}](e) \quad \text{(Error function)} \]

**Encodes probability density functions**

- **Bernoulli** \( x; p \) = \( p \cdot \delta(1 - x) + (1 - p) \cdot \delta(x) \)

- **Gauss** \( x; \mu, \nu \) = \( [\nu = 0] \cdot \delta(x - \mu) + [\nu \neq 0] \cdot \frac{e^{-(x-\mu)^2/(2\nu)}}{(2\pi\nu)^{1/2}} \)

- **UniformInt** \( x; a, b \) = \( \frac{\sum_{x' \in \mathbb{Z}} \delta(x - x') \cdot [a \leq x'] \cdot [x' \leq b]}{\sum_{x' \in \mathbb{Z}}[a \leq x'] \cdot [x' \leq b]} \)
def main() {
    x := Uniform(0,1);
    y := Uniform(0,1);
    z := x * y
    return z;
}

\[ p() = 1 \]

\[ p(x) = [0 \leq x] * [x \leq 1] \]

\[ p(x,y) = [0 \leq x] * [x \leq 1] * [0 \leq y] * [y \leq 1] \]

\[ p(x,y,z) = [0 \leq x] * [x \leq 1] * [0 \leq y] * [y \leq 1] * \delta (z - x * y) \]

\[ p(z) = \int dx \int dy [0 \leq x] * [x \leq 1] * [0 \leq y] * [y \leq 1] * \delta (z - x * y) \]

\[ p(z) = - [z - 1 \leq 0] * [z \neq 0] * [-z \leq 0] * \log(z) \]
PSI: Symbolic Simplification

- Basic algebraic simplifications
  \[ x + x \to 2 \cdot x, \quad x \cdot x \to x^2, \ldots \]

- Simplifications on constraints
  \[ [x = 0] + [x \neq 0] \to 1, \quad [x \leq 0] \cdot [0 \leq x] \to [x = 0], \]
  \[ \delta(x) \cdot [1 \leq x] \to 0, \ldots \]

- Linearize Guards
  \[ \delta(y - x^2) \to \]
  \[ [-y \leq 0] \cdot ([x = 0] \cdot \delta(y) + [x \neq 0] \cdot \frac{1}{2\sqrt{y}}(\delta(x - \sqrt{y}) + \delta(x + \sqrt{y}))), \]
  \[ \ldots \]

- Symbolic Integration
  \[ \int_{\mathbb{R}} dx \int_{\mathbb{R}} dy \, [0 \leq x] \cdot [x \leq 1] \cdot [0 \leq y] \cdot [y \leq 1] \cdot \delta(z - x \cdot y) \to \]
  \[ -[0 < z] \cdot [z \leq 1] \cdot \log(z), \ldots \]
PSI: Current and Future Research

Applications using PSI:
- Security
- Networks
- CPS

Programming Languages:
- Synthesis
- Abstraction
- Dual interpretation

Scalability:
- Specializing
- Exact
- Lazy Approximation
Machine learning based programming tools

\[
\alpha[y] \rightarrow \beta_1, \ldots, \beta_n
\]

Parametrize grammar by a function \( F : \text{grammar} \rightarrow \gamma \)

Synthesize \( F \) from \( D \)

Apply \( F \) to program

Probabilistic Programming

```python
def main() {
p := Uniform(0,1);
r := [1,1,0,1,0];

for i in [0..r.length] {
    observe(Bernoulli(p) == r[i]);
}
return p;
}
```