Probabilistic Learning from Big Code

Martin Vechev
Department of Computer Science
ETH Zurich
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

New kinds of techniques and tools that learn from Big Code

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 from Big Code

Probabilistically likely solutions to problems impossible to solve otherwise

Publications

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
statistical de-obfuscation

http://Nice2Predict.org
structured prediction framework

More information: http://www.srl.inf.ethz.ch/
Scene Completion
[Scene Completion Using Millions of Photographs, ACM SIGGRAPH 2007]
Camera camera = Camera.open();
camera.setDisplayOrientation(90);

SLANG

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
Natural Language Translation

What is the answer to the Question of Life, the Universe, and Everything?

What the answer is the question of life, the universe, and everything?
Programming Language Translation
[Phrase-based statistical translation of programming languages, ACM Onward 2014]
Image de-noisification
Program de-obfuscation

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

```
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;
}
```
JSNice.org
[Predicting program properties from Big Code, ACM POPL 2015]

✔ Every country ✔ 160,000 users ✔ Top ranked tool
Dimensions:
Probabilistic Learning from Big Code

- Applications
- Intermediate Representation
- Analyze Program (PL)
- Train Model (ML)
- Query Model (ML)
# Dimensions: Probabilistic Learning from Big Code

<table>
<thead>
<tr>
<th>Applications</th>
<th>Code completion</th>
<th>Deobfuscation</th>
<th>Program synthesis</th>
<th>Feedback generation</th>
<th>Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intermediate Representation</td>
<td>Sequences (sentences)</td>
<td>Translation Table</td>
<td>Trees</td>
<td>Graphical Models (CRFs)</td>
<td>Feature Vectors</td>
</tr>
<tr>
<td>Analyze Program (PL)</td>
<td>typestate analysis</td>
<td>control-flow analysis</td>
<td>alias analysis</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Train Model (ML)</td>
<td>Neural Networks</td>
<td>SVM</td>
<td>Structured SVM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Query Model (ML)</td>
<td>$\text{argmax } P(y</td>
<td>x)$</td>
<td>$y \in \Omega$</td>
<td>Greedy MAP inference</td>
<td></td>
</tr>
</tbody>
</table>
# Dimensions:
Probabilistic Learning from Big Code

<table>
<thead>
<tr>
<th>Applications</th>
<th>Code completion</th>
<th>Deobfuscation</th>
<th>Program synthesis</th>
<th>Feedback generation</th>
<th>Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intermediate Representation</td>
<td>Sequences (sentences)</td>
<td>Translation Table</td>
<td>Trees</td>
<td>Graphical Models (CRFs)</td>
<td>Feature Vectors</td>
</tr>
<tr>
<td>Analyze Program (PL)</td>
<td>typestate analysis</td>
<td>scope analysis</td>
<td>control-flow analysis</td>
<td>alias analysis</td>
<td></td>
</tr>
<tr>
<td>Train Model (ML)</td>
<td>Neural Networks</td>
<td>SVM</td>
<td>Structured SVM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Query Model (ML)</td>
<td>$\text{argmax } P(y \mid x)$</td>
<td>$y \in \Omega$</td>
<td>Greedy</td>
<td>MAP inference</td>
<td></td>
</tr>
</tbody>
</table>
Camera camera = Camera.open();
camera.setDisplayOrientation(90);

MediaRecorder rec = new MediaRecorder();

rec.setAudioSource(MediaRecorder.AudioSource.MIC);
rec.setVideoSource(MediaRecorder.VideoSource.DEFAULT);
rec.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);

rec.setOutputFile("file.mp4");
...

15
Camera camera = Camera.open();
camera.setDisplayOrientation(90);
camera.unlock();

MediaRecorder rec = new MediaRecorder();
rec.setCamera(camera);
rec.setAudioSource(MediaRecorder.AudioSource.MIC);
rec.setVideoSource(MediaRecorder.VideoSource.DEFAULT);
rec.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);
rec.setAudioEncoder(1);
rec.setVideoEncoder(3);
rec.setOutputFile("file.mp4");
...

---

**Statistical Code Completion**

(Code Completion with Statistical Language Models, ACM PLDI 2014)

**Infers sequences not in training data**

**Handles multiple objects**

**Infers multiple statements**
**insight**: regularities in code similar to regularities in natural language

We want to learn that

```java
MediaRecorder rec = new MediaRecorder();
```

is before

```java
rec.setCamera(camera);
```

like in natural languages

```plaintext
Hello
is before
World!
```
Step 1: From Programs to Sentences

```java
he = new X();
me = new Y();
me.sleep();
if (random()) {
    me.eat();
}
he.enter();
me.talk(he);
```

for abstract object `me`:
- `Y_init` sleep talk
- `Y_init` sleep eat talk

for abstract object `he`:
- `X_init` enter talk param1

*analysis need not be sound*
Step 2: Learn Regularities in Sentences
Choice I: N-gram language model

conditional probability only on previous $n-1$ words

$$P( w_i \mid w_1 \ldots w_{i-1} ) \approx P( w_i \mid w_{i-n+1} \ldots w_{i-1} ) \approx \frac{\#(w_{i-n+1} \ldots w_{i-1} \ldots w_i)}{\#(w_{i-n+1} \ldots w_{i-1})}$$

- easy to implement
- fast to train (simply counting)
- sparseness is an issue (use smoothing or other methods)
- difficult to capture long distance relationships

Existing library implementing language models is SRILM:
http://www.speech.sri.com/projects/srilm/
Choice II: Recurrent Neural Networks (RNN)

Output vector:
- \( o_0 \)
- \( o_1 \)
- \( o_2 \)

Hidden layer:
1. \( W \) (weight matrix)
2. \( s_0 \) (hidden state)
3. \( V \) (input vector)

Input vector:
1. \( x_0 \) (input vector)
2. \( x_1 \) (input vector)
3. \( x_2 \) (input vector)

- open
- write
- read

Each entry is a probability of a word.

- captures “history”

A 1 denotes an input word.

Matrices \( U, V, W \) learned through backprop.

Matrix \( U \) is a low-dimensional continuous vector representation of input words, also referred to as “word embedding.”
Choice II: Recurrent Neural Networks (RNN)

+ can learn dependencies beyond the prior several words (e.g., LSTM)

+ learns continuous representation of vocabulary (helps avoid sparsity)

- slow and tricky to train

- predictions hard to explain


[https://www.tensorflow.org/](https://www.tensorflow.org/)
Statistical Code Completion

[Code Completion with Statistical Language Models, ACM PLDI 2014]
Importance of Static Analysis

precision vs. % of data used

- red: no alias analysis
- green: with alias analysis

*need to decide what to `compile` into a word

static analysis benefit = 10x more data
# Dimensions: Probabilistic Learning from Big Code

<table>
<thead>
<tr>
<th>Applications</th>
<th>Code completion</th>
<th>Deobfuscation</th>
<th>Program synthesis</th>
<th>Feedback generation</th>
<th>Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intermediate Representation</td>
<td>Sequences (sentences)</td>
<td>Translation Table</td>
<td>Trees</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analyze Program (PL)</td>
<td>typestate analysis</td>
<td>control-flow analysis</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Train Model (ML)</td>
<td>Neural Networks</td>
<td>SVM</td>
<td>N-gram/back-off</td>
<td>PCFG</td>
<td></td>
</tr>
<tr>
<td>Query Model (ML)</td>
<td>$\arg\max_y P(y \mid x)$</td>
<td>$y \in \Omega$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

function chunkData(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;

Key Challenges

Facts to be predicted are dependent

Prediction should be fast (real-time)
(hard, millions of possible choices)
Key Idea

Phrase the problem of predicting program facts as

Structured Prediction for Programs
Structured Prediction for Programs

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

Conditional Random Field

Structured SVM

Program Analysis

MAP Inference

Undirected Probabilistic Graphical Model
Captures dependence between facts

framework open-sourced at:
http://nice2predict.org

JSNice: an application on top

First connection between Programs and Conditional Random Fields
Conditional Random Field

[J. Lafferty, A. McCallum, F. Pereira, ICML 2001]
Conditional Random Field

[J. Lafferty, A. McCallum, F. Pereira, ICML 2001]
Conditional Random Field
[J. Lafferty, A. McCallum, F. Pereira, ICML 2001]

Graph + functions determine a CRF

\[
P(\text{first}, \text{last} | \text{town}) = \frac{\exp(0.1 f_1 + 0.3 f_2 + 0.7 f_3 + 0.4 f_4)}{Z(\text{town})}
\]

\[
P(y | x) = \frac{1}{Z(x)} \exp(w^T f(y, x))
\]
Structured Prediction for Programs

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

Graph + functions determine a CRF

\[
P(\text{first}, \text{last} \mid \text{town}) = \frac{\exp (0.1 \cdot f_1 + 0.3 \cdot f_2 + 0.7 \cdot f_3 + 0.4 \cdot f_4)}{Z(\text{town})}
\]

\[
P(y \mid x) = \frac{1}{Z(x)} \exp(w^T f(y, x))
\]
Structured Prediction for Programs

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

Graph + functions determine a CRF

\[
P(\text{first, last} \mid \text{town}) = \frac{\exp (0.1 \cdot f_1 + 0.3 \cdot f_2 + 0.7 \cdot f_3 + 0.4 \cdot f_4)}{Z(\text{town})}
\]

\[
P(y \mid x) = \frac{1}{Z(x)} \exp(w^T f(y, x))
\]

weights learned via SSVM training

query the CRF via MAP Inference

\[
\arg \max_w w^T f(y, x)
\]

\[
y \in \Omega_x
\]

problem NP-Hard: designed an approximate algorithm
function chunkData(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(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(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(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(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;

function chunkData(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;

argmax $w^T f(i, t, r, length)$
var n = [];
var r = e.length;
var i = 0;
for (; i < r; i += t)
    if (i + t < r)
        n.push(e.subs(i, i + t));
    else
        n.push(e.subs(i, r));
return n;

var colNames = [];
var len = str.length;
var i = 0;
for (; i < len; i += step)
    if (i + step < len)
        colNames.push(str.subs(i, i + step));
    else
        colNames.push(str.subs(i, len));
return colNames;

var

150MB

Predicted program properties from Big Code, ACM POPL 2015

Prediction Phase

program analysis -> MAP inference -> transform

Learning Phase

program analysis -> SSVM learning

Conditional Random Field

P(y | x)

max-margin training

alias, call analysis

7M feature functions for names
70K feature functions for types

Atlassian
Bitbucket

GitHub
JSNice: over a year of usage

May 2015 – May 2016

Total: 12GB of code
# 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>

<table>
<thead>
<tr>
<th>Intermediate Representation</th>
<th>Sequences (sentences)</th>
<th>Translation Table</th>
<th>Graphical Models (CRFs)</th>
<th>Feature Vectors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Trees</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Analyze Program (PL)</th>
<th>typestate analysis</th>
<th>control-flow analysis</th>
<th>alias analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>scope analysis</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Train Model (ML)</th>
<th>Neural Networks</th>
<th>SVM</th>
<th>Structured SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N-gram/back-off</td>
<td>PCFG</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query Model (ML)</th>
<th>$\arg\max_y P(y \mid x)$</th>
<th>$y \in \Omega$</th>
<th>Greedy MAP inference</th>
</tr>
</thead>
</table>

More information and tutorials at:  
Future

Knowledge

Methods

Applications

Machine outperforms humans in programming tasks

Probabilistic model for code (icml’16)
Learning with noise (popl’16)

Frameworks

Nice2Predict.org

De-obfuscation (popl’15)

Applications

Code translation (onward’14)

Code synthesis (pldi’14)

2014  2017  2020  2024

Time
Probabilistic Higher Order Grammar (PHOG)

[PHOG: Probabilistic Model for Code, ICML’16]

PCFG rule:

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

PHOG rule:

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

conditioning context

PHOG generalizes PCFG
**PHOG**
[PHOG: Probabilistic Model for Code, ICML’16]

New **probabilistic model**, cleanly generalizes prior work

**Widely applicable**: synthesis, sampling, fuzzing, any PL

New procedure to automatically **learn feature functions**

**More:** Learning Programs from Noisy Data, ACM POPL’16
PHOG: Probabilistic Model for Code, ACM ICML’16
Evaluation

Learn a Probabilistic Model of JavaScript
## Predict JavaScript APIs

<table>
<thead>
<tr>
<th>Function F</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Last two tokens, Hindle et. al. [ICSE’12]</td>
<td>22.2%</td>
</tr>
<tr>
<td>Last two APIs, Raychev et. al. [PLDI’14]</td>
<td>30.4%</td>
</tr>
<tr>
<td><strong>Our Model</strong></td>
<td><strong>66.6%</strong></td>
</tr>
</tbody>
</table>
## Predict JavaScript Values

<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifier</td>
<td>62%</td>
<td><code>contains = jQuery ...</code></td>
</tr>
<tr>
<td>Property</td>
<td>65%</td>
<td><code>start = list.length;</code></td>
</tr>
<tr>
<td>String</td>
<td>52%</td>
<td><code>‘[‘ + attrs + ‘]’</code></td>
</tr>
<tr>
<td>Number</td>
<td>64%</td>
<td><code>canvas(xy[0], xy[1], ...)</code></td>
</tr>
<tr>
<td>RegExp</td>
<td>66%</td>
<td>`line.replace(/(&amp;nbsp;</td>
</tr>
<tr>
<td>UnaryExpr</td>
<td>97%</td>
<td>`if (!events</td>
</tr>
<tr>
<td>BinaryExpr</td>
<td>74%</td>
<td><code>while (++index &lt; ...)</code></td>
</tr>
<tr>
<td>LogicalExpr</td>
<td>92%</td>
<td>`frame = frame</td>
</tr>
</tbody>
</table>
How do I get started?

Reading: **Learning from Large Codebases, PhD Thesis, 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
Probabilistic Learning from Big Code

**TREND**

Google  facebook  Microsoft

**NEW PROBABILISTIC MODELS**

\[ \alpha [\gamma] \rightarrow \beta_1 \ldots \beta_n \]

- **PARAMETRIZE** grammar by a function \( F : \chi \rightarrow \gamma \)
- **SYthesize** \( F \) from \( D \) \( \rightarrow \chi \rightarrow F (\chi) \rightarrow \gamma \)
- **APPLY** \( F \) to program

**NEW APPLICATIONS**


**RICH PROBLEM SPACE**

<table>
<thead>
<tr>
<th>Applications</th>
<th>Code completion</th>
<th>Deobfuscation</th>
<th>Program synthesis</th>
<th>Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intermediate Representation</td>
<td>Sequences (sentences)</td>
<td>Translation Table</td>
<td>Trees</td>
<td>Graphical Models (CRFs)</td>
</tr>
<tr>
<td>Analyze Program (PL)</td>
<td>typestate analysis</td>
<td>scope analysis</td>
<td>control-flow analysis</td>
<td>alias analysis</td>
</tr>
<tr>
<td>Train Model (ML)</td>
<td>Neural Networks</td>
<td>PCFGs</td>
<td>SVM</td>
<td>Structured SVM</td>
</tr>
<tr>
<td>Query Model (ML)</td>
<td>( \operatorname{argmax} { F(y) \mid y \in D } )</td>
<td>Greedy</td>
<td>MAP inference</td>
<td></td>
</tr>
</tbody>
</table>