Probabilistic Model for Code with Decision Trees

Veselin Raychev, Pavol Bielik, Martin Vechev
Software Reliability Lab
Department of Computer Science
ETH Zurich
## Big Data Revolution

<table>
<thead>
<tr>
<th>Research area</th>
<th>Big Data</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer vision</td>
<td><img src="image1.png" alt="IMAGENET" /></td>
<td><img src="image2.png" alt="Image labeling" /></td>
</tr>
<tr>
<td></td>
<td><img src="image3.png" alt="COCO" /></td>
<td>A group of people shopping at an outdoor market. There are many vegetables at the fruit stand.</td>
</tr>
<tr>
<td>Programming languages</td>
<td><img src="image4.png" alt="GitHub" /></td>
<td><img src="image5.png" alt="Bitbucket" /></td>
</tr>
<tr>
<td></td>
<td><img src="image5.png" alt="Bitbucket" /></td>
<td><img src="image6.png" alt="GitLab" /></td>
</tr>
</tbody>
</table>
All of these benefit from a good probabilistic model for code.
Probabilistic model for code

Model is a key part of the Statistical Programming Tools

Goal: score programs
Select best among several candidates

Example: Which function is more likely?

```javascript
function area(a) {
    return a.width * a.height
}
```

```javascript
function area(a) {
    return a.width * a.close()
}
```
Model is a key part of Statistical Programming Tools

Example:

```javascript
function area(a) {
    return a.width * a.
}
```

Goal: score programs
Select best among several candidates

- Very likely
- Less likely
- Impossible
Directly applicable to code completion, but is a key statistical component for many others tasks: e.g. natural language to code, statistical bug localization
Existing works: naive models

Most common model: n-gram

Training (3-gram model):

```plaintext
return a.width * a.
+ a.open (mode);
+ a.close ();
* a.height;
* a.close ();
```

Count 3-grams:

<table>
<thead>
<tr>
<th>e.g a.close</th>
<th>Count 3-grams</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2 times)</td>
<td></td>
</tr>
</tbody>
</table>

Prediction:

```plaintext
width + a.
```

Probability:

- close: 0.4
- open: 0.2
- width: 0.2
- height: 0.2

3-gram:

- a.close - 2 times
- a.open - 1 times
- a.width - 1 times
- a.height - 1 times
Existing works: naive models

Most common model: n-gram

Training (3-gram model):

```
<table>
<thead>
<tr>
<th>Word</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>close</td>
<td>2</td>
</tr>
<tr>
<td>open</td>
<td>1</td>
</tr>
<tr>
<td>width</td>
<td>1</td>
</tr>
<tr>
<td>height</td>
<td>1</td>
</tr>
</tbody>
</table>
```

Prediction:

```
width + a.
```

Probability

- close: 0.4
- open: 0.2
- width: 0.2
- height: 0.2

Main problem:

**Bad context** leads to bad probability estimates
1. Richer conditioning context

```
return a.width * a.?
```

2. Domain Specific Language encoding Decision Trees

```
Lang ::= BasicPart | BranchPart

BranchPart ::= if pred(x) then Lang else Lang
```
Conditioning context

Training (3-gram model):

\[
\text{return } x \cdot \text{width} \times x \cdot \text{height}
\]

\[
\text{return } y \cdot \text{width} \times y \cdot \text{height}
\]

\[
\text{area} = s \cdot \text{width} \times s \cdot \text{height}
\]

\[
\text{s.width} = \text{s.width} + 10
\]

\[
\text{q.depth} \times \text{q.width} \times \text{q.height}
\]

Prediction:

\[
\text{return } a \cdot \text{width} \times a
\]

Context: relevant for this prediction

Height | Width | 0.8 | 0.2

- width
- width 4 times
- width 1 time
- width 0 times
- close

10
Indirection

Query: \( x \)

\[
\text{return } a.\text{width} \times a.\text{height}
\]

Completion: \( y \)

\[
? = \text{height}
\]

Context: \( \text{ctx} = f(x) \)

\[
\text{return } a.\text{width} \times a.\text{height}
\]

Model: \( P(y|f(x)) \)

- width
- height

- width
- width

4 times
1 time
Example contexts

Query: \( x \)

return \( a.\text{width} \times a. \) ?

\( \text{ctx} = f(x): \) last two tokens
3-gram language model

return \( a.\text{width} \times a. \) ?

\( \text{ctx} = f(x): \) previous actions
on the same object [PLDI'14]

return \( a.\text{width} \times a. \) ?

\( \text{ctx} = f(x): \) the third token
before the completion

Question:
Which context is the best?

Try many of them and evaluate
### Evaluation metric: entropy

<table>
<thead>
<tr>
<th>Expression</th>
<th>Conditional Probability</th>
<th>Entropy (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>console.info</code></td>
<td>$P(\text{info}</td>
<td>\text{console.}) = 1$</td>
</tr>
<tr>
<td><code>console.info</code></td>
<td>$P(\text{info}</td>
<td>\text{console.}) = 1$</td>
</tr>
<tr>
<td><code>a.width * a.height</code></td>
<td>$P(\text{height}</td>
<td>a.) = 0.5$</td>
</tr>
<tr>
<td><code>b.width * b.height</code></td>
<td>$P(\text{height}</td>
<td>b.) = 0.5$</td>
</tr>
<tr>
<td><code>a.right - a.left</code></td>
<td>$P(\text{left}</td>
<td>a.) = 0.5$</td>
</tr>
<tr>
<td><code>b.bottom - b.top</code></td>
<td>$P(\text{top}</td>
<td>b.) = 0.5$</td>
</tr>
</tbody>
</table>

Average entropy: ~0.66667 bits
Evaluation metric: entropy

console.info

P(info | .) = 1/3
~1.58

a.width * a.height

P(height | .) = 1/3
~1.58

b.width * b.height

P(height | .) = 1/3
~1.58

a.right - a.left

P(left | .) = 1/6
~2.58

b.bottom - b.top

P(top | .) = 1/6
~2.58

Average entropy: ~1.91 bits
## Unconditional model

No conditioning in the probability distribution: 
\[ \text{ctx} = \bot \]

<table>
<thead>
<tr>
<th>Expression</th>
<th>( P(a) )</th>
<th>( \text{Entropy (bits)} )</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a.width * a.height</code></td>
<td>( P(a) = 1/3 )</td>
<td>( \sim 1.58 )</td>
</tr>
<tr>
<td><code>b.width * b.height</code></td>
<td>( P(a) = 1/3 )</td>
<td>( \sim 1.58 )</td>
</tr>
<tr>
<td><code>a.right - a.left</code></td>
<td>( P(a) = 1/6 )</td>
<td>( \sim 2.58 )</td>
</tr>
<tr>
<td><code>b.bottom - b.top</code></td>
<td>( P(a) = 1/6 )</td>
<td>( \sim 2.58 )</td>
</tr>
</tbody>
</table>

Average entropy: \( \sim 1.91 \) bits
1. Richer conditioning context

return a.width * a.height

2. Domain Specific Language encoding Decision Trees

3. Evaluation

Lang ::= BasicPart | BranchPart

BranchPart ::= if pred(x) then Lang else Lang
Synthesize the best model

Query: $x$

```
return a.width * a.?
```

Completion: $y$

```
? = height
```

Context: $\text{ctx} = f(x)$

```
return a.width * a.?
```

Model: $P(y|\text{f}(x))$

```
  4 x
  1 x
```

Find function $f$

From a domain specific language

Same basic idea in
Learning Programs from Noisy Data [POPL'16]
Main DSL requirement

**Program** ::= **BasicPart | BranchPart**

**BranchPart** ::= if \( \text{pred}(x) \) then **Program** else **Program**

**Basic part**

Includes models from a simple DSL

- return \( a \cdot \text{width} \times a \cdot \text{height} \)
- ctx = \( f(x) \): last two tokens
- 3-gram language model

- return \( a \cdot \text{width} \times a \cdot \text{height} \)
- ctx = \( f(x) \): previous actions on the same object [PLDi’14]

**Branch part**

Combines programs based on predicates. Language of predicates \( \text{pred} \)

- e.g. are we completing a field name?
- Is there another operation on the same object?

**Special empty program** \( \varepsilon \)
If there is no prior field access, then use 3-gram model, otherwise prior field access

- `console.info`  \( P(\text{info} | \text{console.}) = 1 \)  \( 0 \)
- `console.info`  \( P(\text{info} | \text{console.}) = 1 \)  \( 0 \)
- `a.width * a.height`  \( P(\text{height} | a.) = 1 \)  \( 0 \)
- `b.width * b.height`  \( P(\text{height} | b.) = 1 \)  \( 0 \)
- `a.right - a.left`  \( P(\text{left} | a.) = 1 \)  \( 0 \)
- `b.bottom - b.top`  \( P(\text{top} | b.) = 1 \)  \( 0 \)
Synthesis using basic part of DSL

Includes models from a simple DSL

At high level: Search through **thousands** of candidate programs that describe conditioning.

Involved See POPL'16
Synthesis of branch part

Space of possible programs:

\[ \text{BranchPart} := \text{if pred(x) then Program else Program} \]

- Search thousands
- Search thousands
- Search thousands

Billions even without nesting

Infinite with nesting

Goal: find program with best entropy
Idea 1: synthesis in parts

1. Synthesize branch with empty programs in leaves

**Synthesize** : if $\text{pred}(x)$ then $\varepsilon$ else $\varepsilon$

Search thousands

Goal: find program with approximately best entropy

2. Synthesize leaves separately

**Synthesize** : if $\text{pred}(x)$ then Program else Program

Then synthesize basic part

Until no predicate improves entropy over basic $\varepsilon$

Recursively call 1 for branch

Recursively call 1 for branch

This is decision tree learning
Synthesis of branch part

Space of possible programs:

\[
\text{BranchPart} ::= \text{if } \text{pred}(x) \text{ then Program else Program}
\]

- Search thousands
- Search thousands
- Search thousands

Search for each component separately

Performance at expense of possible non-optimality

Goal: find program with approximately best entropy
Idea 2: synthesis in parts

1. Synthesize basic program
   \[ f' \in \text{BasicPart} \]

2. Synthesize branch
   \[ \text{Synthesize} : \text{if } \text{pred}(x) \text{ then } f' \text{ else } f' \]

3. Synthesize leaves separately
   \[ \text{Synthesize} : \text{if } \text{pred}(x) \text{ then Program else Program} \]

Goal: find program with approximately best entropy
Synthesis procedure 1 is a new formulation of a known and popular algorithm for decision tree learning: ID3

In fact, we extended ID3 to support programs in decision tree leaves from the BasicPart DSL fragment

Synthesis procedure 2 is new
And also applicable to decision trees

New name: E13
1. Richer conditioning context

```python
return a.\texttt{width} * a.\texttt{?}
```

2. Domain Specific Language encoding Decision Trees

3. Evaluation

\[
\text{Lang} ::= \text{BasicPart} | \text{BranchPart}
\]

\[
\text{BranchPart} ::= \text{if pred}(x) \text{ then Lang else Lang}
\]
Evaluation

150’000 JavaScript files, from GitHub.com, parsed into ASTs. Public datasets.

100’000 files Training data for synthesis

50’000 files Evaluation data

Evaluation files are not on the same projects as training

Synthesis time: ~100 hours

Question: How well can we predict program elements?
Query time for all models is basically the same (>10K queries per second)

<table>
<thead>
<tr>
<th>Task</th>
<th>PCFG</th>
<th>3-gram</th>
<th>ID3+</th>
<th>E13</th>
</tr>
</thead>
<tbody>
<tr>
<td>API completion</td>
<td>0.04%</td>
<td>30.0%</td>
<td>54%</td>
<td>66.6%</td>
</tr>
<tr>
<td>Field access completion</td>
<td>3.2%</td>
<td>32.9%</td>
<td>52.5%</td>
<td>67.0%</td>
</tr>
<tr>
<td>Predicting loops</td>
<td>0%</td>
<td>37.5%</td>
<td>65.0%</td>
<td>28.3%</td>
</tr>
</tbody>
</table>

Many more evaluation results in the paper.... Also for Python. Easily applicable to all languages.

Big improvement over prior methods

Both ID3+ and E13 learn useful probabilistic models