Synthesis for interpretable machine learning

Veselin Raychev

veselin@deepcode.ai

Example application: code completion

Task: write an android application that records a video

```
Camera camera = Camera.open();
camera.SetDisplayOrientation(90);
                ?
```

Completion based on existing programs

But this program is not in the training data

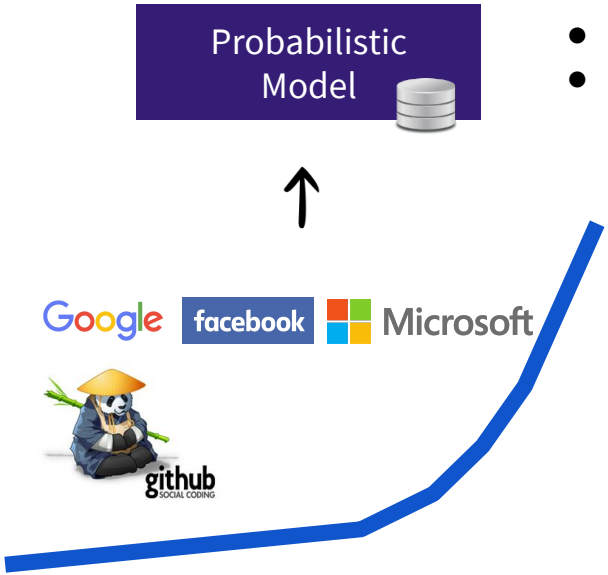
```
Camera camera = Camera.open();
camera.SetDisplayOrientation(90);
camera.unlock();

SurfaceHolder h = getHolder();
h.addCallback(this);
h.setType(SurfaceHolder.STP);

MediaRecorder r = new MediaRecorder();
r.setCamera(camera);
```

Need to generate and score solutions:

```
Camera camera = Camera.open();
camera.SetDisplayOrientation(90);
camera.release();
camera.release();
```

Probabilistic model for code or text

Can assign probabilities to **programs**, just like a model for **text** assigns probabilities to words and sentences



Can be used as a building block for a variety of systems:

- Code completion
- Translation
- Bug finding
- Change suggestion

Write new code:
Code Completion

```
Camera camera =
Camera.open();
camera.SetDisplayOrienta
tion(90);
     ?
```

Port code:
Programming Language Translation

C# | Java

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

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

Understand code/security:
Code Deobfuscation
Type Prediction

JS NICE | DEGUARD

Find issues:
Statistical Defect Prediction

DEEPCODE

Naive approach

N-gram model

Such models are shallow, but work well with huge datasets

→ English (most likely next word prediction)

```
Read  the  file  , please !
```

→ Code (most likely next token prediction)

```
file  .  read  ( ) ;
```

Example

query:

```
f.open(“file” , “r”);

f. ?
```

Training data ***D***

```
f.open(“f2” , “r”);
f.read();
```

```
f.open(“f2” , “rw”);
f.write(“c”);
```

```
f.open(“f1” , “r”);
f.read();
```

query:

```
f.open(“file” , “r”);

f. ?
```

Training data ***D***

```
f.open(“f2” , “r”);
f.read();
```

```
f.open(“f2” , “rw”);
f.write(“c”);
```

```
f.open(“f1” , “r”);
f.read();
```

→

3-gram Probabilistic Model P

```
P(open  | f. ) ~ 3/6
P(read  | f. ) ~ 2/6
P(write | f. ) ~ 1/6
```

context y

Prediction: open
Hindle et. al. [ICSE’12]

query:

```
f.open(“file” , “r”);

f. open
```

Training data ***D***

```
f.open("f2" , "r");
f.read();
```

```
f.open("f2" , "rw");
f.write("c");
```

```
f.open("f1" , "r");
f.read();
```

Prior API Probabilistic Model P

```
P(read  | open ) ~ 2/3
P(write | open ) ~ 1/3
```

context γ

3-gram Probabilistic Model P

```
P(open  | f. ) ~ 3/6
P(read  | f. ) ~ 2/6
P(write | f. ) ~ 1/6
```

context γ

Better prediction: `read`

Raychev et. al. [PLDI'14]

query:

```
f.open("file" , "r");

f. read
```

3-gram Probabilistic Model P

6
6
6

Different conditioning **contexts** lead to different probabilistic models

P(prediction | **context**)

Manually providing contexts is hard, error-prone and suboptimal !

How to find the best context?

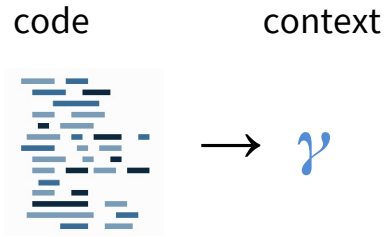
Prior AP

P(rea
P(wri

context

Synthesis approach

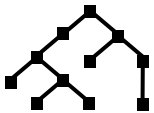
key idea: synthesize a function *f:*



Program synthesis: techniques for learning short programs from examples.

Creating a probabilistic model

1. Pick data representation:



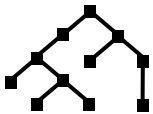
2. Define a domain-specific language for expressing functions:

DSL

3. Synthesize $f_{best} \in$ DSL from Dataset $\boldsymbol{D}$:

$$f_{best} = \underset{f \in \text{DSL}}{\text{argmin}}\ \text{cost}(\boldsymbol{D}, f)$$

4. Use f_{best} to compute contexts, learn and predict:

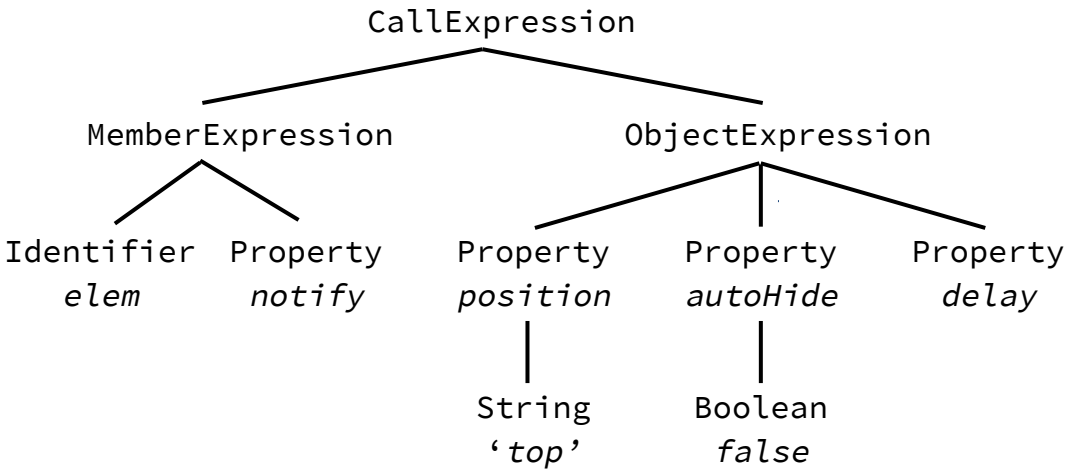
f_{best} () → y

Step 1: Pick data representation (for code)

JavaScript program

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

AST

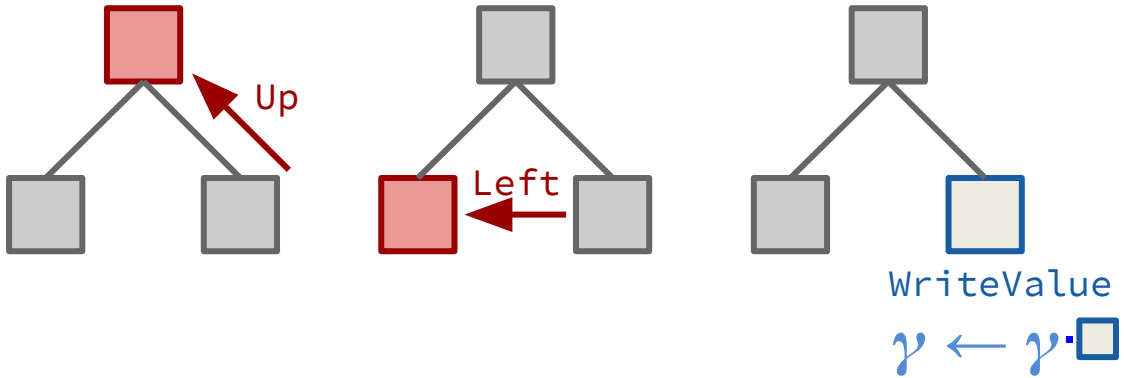


Step 2: Define domain-specific language

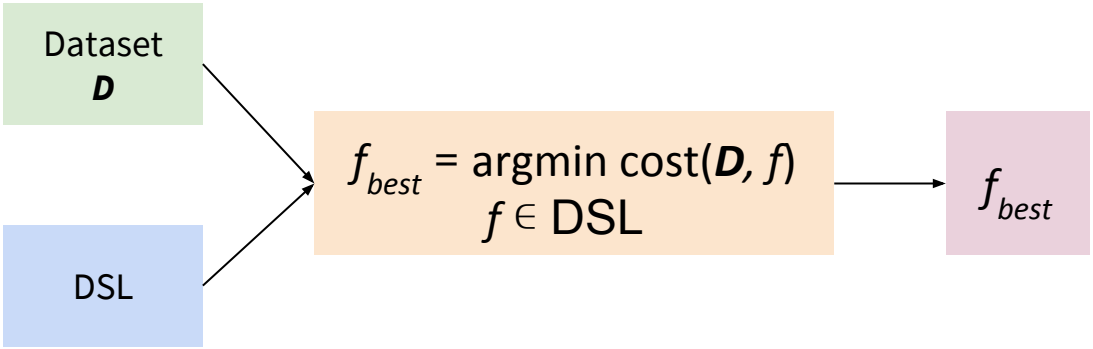
Syntax

```
TCond   ::=  ε | WriteOp TCond | MoveOp TCond |
                                    if TCond == const TCond else TCond

MoveOp  ::=  Up, Left, Right, DownFirst, DownLast,
             NextDFS, PrevDFS, NextLeaf,
             PrevLeaf,PrevNodeType, PrevNodeValue, PrevNodeContext

WriteOp ::=  WriteValue, WriteType, WritePos
```

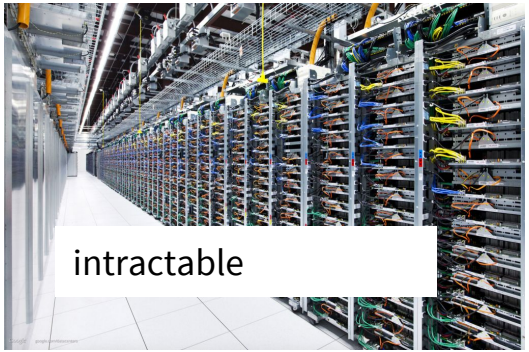
Semantics



Step 3: Synthesize f_{best}



Enumerative search



Enabled via advances in synthesis

Discussed later

[1] **Learning Programs from Noisy Data**
Veselin Raychev, Pavol Bielik, Martin Vechev, Andreas Krause
ACM POPL 2016

[2] **PHOG: Probabilistic Model for Code**
Pavol Bielik, Veselin Raychev, Martin Vechev
ICML 2016

[3] **Probabilistic Model for Code with Decision Trees**
Veselin Raychev, Pavol Bielik, Martin Vechev
ACM OOPSLA 2016

Step 4: Use f_{best} to train and predict

Query

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

f_{best}

```
Left
WriteValue
Up
WritePos
Up
DownFirst
DownLast
WriteValue
```

Context γ

```
{}
{hide}
{hide}
{hide, 3}
{hide, 3}
{hide, 3}
{hide, 3}
{hide, 3, notify}
```

{Previous Property, Parameter Position, API name}

Automatically learned program analysis

Evaluation of a machine learning system

100'000
JavaScript files
Training data

From different
projects

50'000
JavaScript files
Evaluation data

Synthesize f_{best}

Learn model

P(x | context)

Accuracy
evaluation

Code prediction
system

Deep3

Experiment: JavaScript API prediction

Synthesis time: ~**100** hours

f_{best} ~ **2000** instructions

There is only one context and even it is suboptimal

Probabilistic Model	Accuracy
Last two tokens, Hindle et. al. [ICSE'12]	22.2%
Last two APIs, Raychev et. al. [PLDI'14]	30.4%
Deep3 (Synthesized Model)	**66.6%**

Beyond code… to Natural Languages

Pavol Bielik, Veselin Raychev, Martin Vechev
ICLR 2017

```
Writing in natural lan g
```

Predict next character

Immediately useful for virtual keyboards (phones)

DSL for natural language: **see exercise**.

Beyond code… to Natural Languages

Dataset **D:**

Hutter Prize Wikipedia Dataset

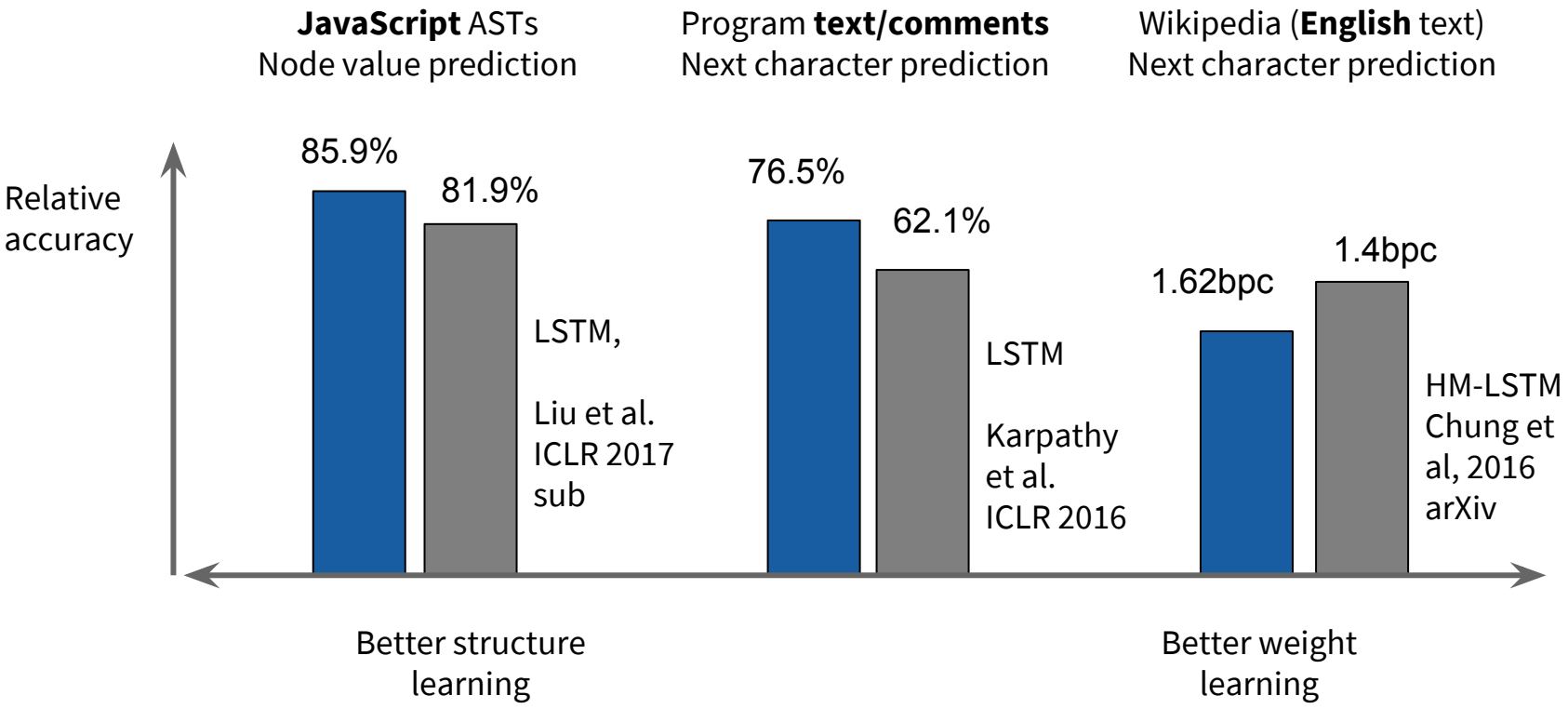
Synthesis time: ~**8** hours

f_{best} ~ **9000** instructions

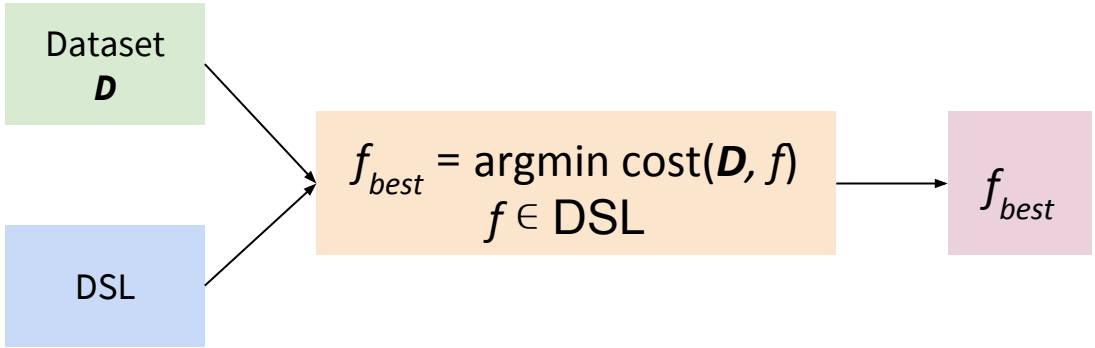
Probabilistic Model	Entropy (bits per char)
Stacked neural network LSTM (Graves 2013)	1.67
MI-LSTM, HM-LSTM	1.40-1.44

Synthesis Two orders of magnitude faster to query than LSTM **1.62**

Synthesis vs Neural networks



How does it work?



Cost: entropy

Entropy = Number of bits to encode the dataset

$$H(\boldsymbol{D}) = -\sum_{x \in \boldsymbol{D}} 1/n \log_2 Pr(x)$$

$$x = (\text{ prefix }, \text{ char })$$

$$Pr(x) = P(\text{ char } | f(\text{prefix}))$$

Example dataset:
abcbabc

f_1 = `Left WriteChar`

f_1(prefix)	char	
f_1(abcbab_) = b	c	P(c|b) = 2/3
f_1(abcba_) = a	b	P(a|b) = 1/3
f_1(abcb_) = b	a	
f_1(abc_) = c	b	P(b|a) = 1
f_1(ab_) = b	c	P(b|c) = 1
f_1(a_) = a	b	P(a|ε) = 1
f_1(_) = ε	a	

H(D) = -2/7 $\log_2$(2/3) - 1/7 $\log_2$(1/3)
0.39 bits per char

Cost: entropy

Entropy = Number of bits to encode the dataset

$$H(\boldsymbol{D}) = -\sum_{x \in \boldsymbol{D}} 1/n \log_2 Pr(x)$$

$$x = (\text{ prefix }, \text{ char })$$

$$Pr(x) = P(\text{ char } | f(\text{prefix}))$$

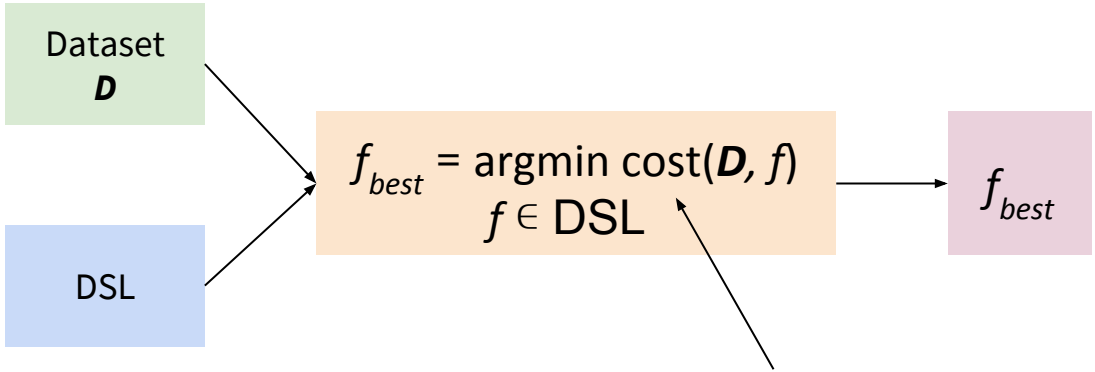
Example dataset:
abcbabc

f_2 = `Left Left WriteChar`

f_2(prefix)	char	
f_2(abcbab_) = a	c	P(c|a) = 1
f_2(abcba_) = b	b	P(b|b) = 1
f_2(abcb_) = c	a	P(a|c) = 1
f_2(abc_) = b	b	
f_2(ab_) = a	c	
f_2(a_) = ε	b	P(b|ε) = 1/2
f_2(_) = ε	a	P(a|ε) = 1/2

H(D) = -2/7 $\log_2$(1/2)
0.29 bit per character

How does it work?



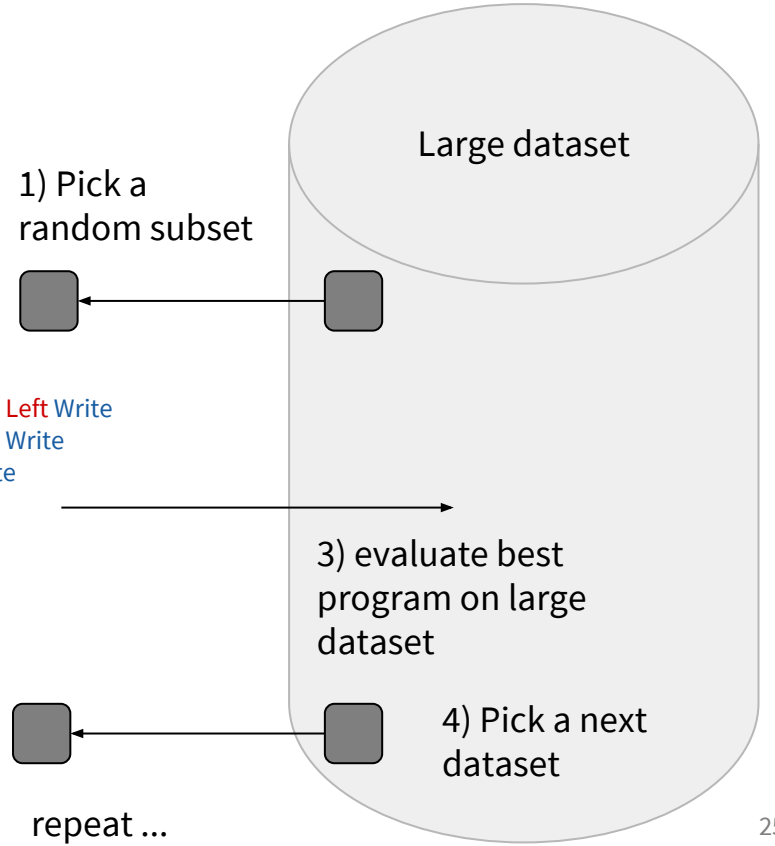
Evaluating cost is slow for large dataset

Idea: use counter-example-based synthesis

Synthesis with large datasets

Model described by a short program

Left Left Write



More expressive DSL with branches

Model described by a short program

if Left Write **== space**

- Left PrevChar Right Write

else

- Left Write

But different program for different kinds of predictions - first character of a word is different than other characters

Synthesized programs are not short anymore

Search space is intractable

Synthesis: DSL and decision trees

Interesting DSL program:

ε

Describes unconditional model

P(prediction | empty context)

Can use it to synthesize in pieces

Synthesis: DSL and decision trees

Interesting DSL program:

ε

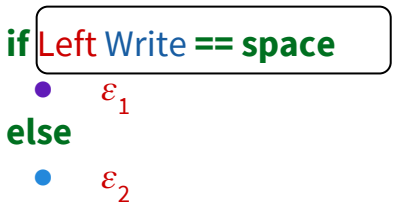
Describes unconditional model

P(prediction | empty context)

Can use it to synthesize in pieces

Synthesize condition expression that splits the dataset and gives it to empty programs in branches

1.



Synthesis: DSL and decision trees

Interesting DSL program:

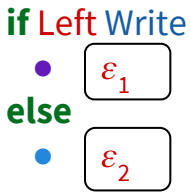
ε

Describes unconditional model

P(prediction | empty context)

Can use it to synthesize in pieces

2.



Recursively repeat for both branches

Synthesis: DSL and decision trees

Interesting DSL program:

ε

Describes unconditional model

P(prediction | empty context)

Can use it to synthesize in pieces

3.

if Left Write **== space**

- Left PrevChar Right Write

else

- Left Write

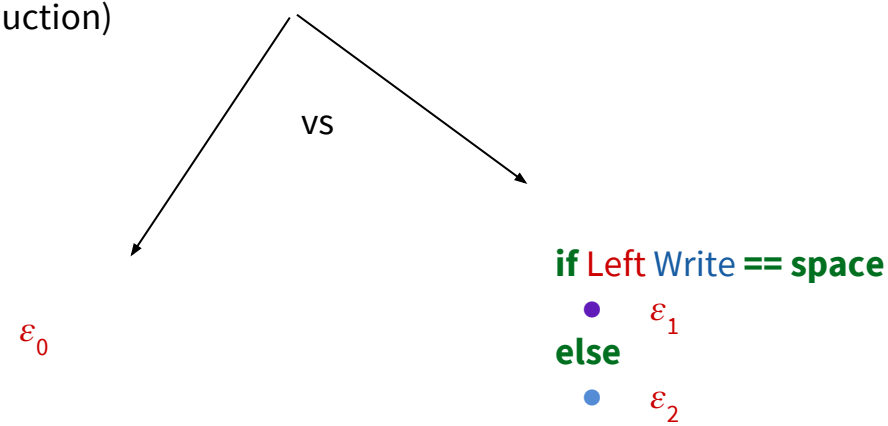
Synthesize branch-free programs if recursive branches do not improve score

Synthesis: DSL and decision trees

Main result 1: When optimization goal is entropy, then this is a formulation of the **ID3** decision tree learning algorithm.

ID3 is the most commonly used decision tree algorithm, but it is not optimal

ID3 maximizes information gain (i.e. entropy reduction)



Synthesis: DSL and decision trees

Main result 2: DSL formulation allows us to instantiate other previously unexplored decision tree algorithms

New decision tree algorithm: E13

1. `Left Write`

   Synthesize one program for the entire dataset

2. **if** `Left Write` **== space**
   - `Left Write`

   else
   - `Left Write`

   Synthesize condition expression for 1st program in both branches

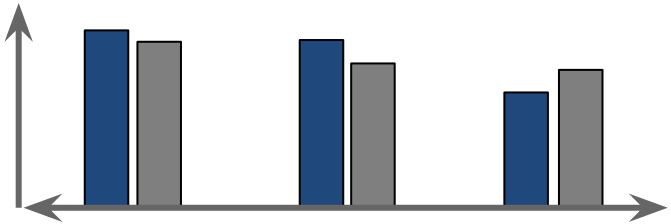
3. **if** `Up WriteType` **== MethodCallExpression**
   - `Left PrevChar Right Write`

   else
   - `Left Write`

   Synthesize recursively both both branches

Probabilistic model with synthesis

Accuracy **competitive** with neural networks



Explainable predictions: Synthesized DSL programs are interpretable by humans

```
if Left Write == space
    • Left Write
else
    • Left Write
```

Application in programming **tools** and **beyond**

```
f.open("file" , "r");

f. ?
```

Natural language processing

DeepCode.AI

ETH spin-off

MSc theses offered

If you are interested to work on interpretable machine learning, write me veselin@deepcode.ai and martin.vechev@inf.ethz.ch

Demo