Precise Piecewise Affine Models from Input-Output Data

Rajeev Alur  Nimit Singhania
University of Pennsylvania

EXCAPE PI Meeting, June 2015
Motivation

Real World Systems → Models

Complex → Easy to Analyze

Factory Automation

Robotic Surgery
Motivation

Real World Systems

Models

Data Driven

Input-Output Data
Motivation

Real World Systems

Data Driven

Input-Output Data

Piecewise Affine Models
1D Piecewise Affine Model

\[
\begin{align*}
\text{if } (x < 2.5) & \quad 5 - x \\
\text{else if } (x < 7.5) & \quad x \\
\text{else} & \quad 15 - x
\end{align*}
\]
2D Piecewise Affine Model

\[ l_1: 0 \]
\[ l_2: y - x \]
\[ l_3: (y - x - 16)/5 \]

Linear Functions
2D Piecewise Affine Model

$l_1: 0$

$l_2: y - x$

$l_3: (y - x - 16)/5$

Linear Functions

(x < -1 and y < -1) or (x > 1 and y > 1)

y < x - 4

Guard Predicates
Problem

Input-Output Data $D$  

Piecewise Affine Model $f$

$f$ closely approximates $D$  
$f$ is simple
Problem

• Given input-output data $D: \mathbb{R}^d \times \mathbb{R}$ and error bound $\delta$, learn a piecewise affine model $f: \mathbb{R}^d \to \mathbb{R}$

• $|f(a) - b| \leq \delta$ for all points $(a, b) \in D$

• Minimize size of model
  • Minimize number of linear functions and size of predicates
Problem

• Given input-output data $\mathbf{D}: \mathbb{R}^d \times \mathbb{R}$ and error bound $\delta$, learn a piecewise affine model $f: \mathbb{R}^d \rightarrow \mathbb{R}$

• $|f(a) - b| \leq \delta$ for all points $(a, b) \in \mathbf{D}$

• Minimize size of model
  • Minimize number of linear functions and size of predicates

• Two problems
  • Learn linear functions
  • Learn guard predicates for associated regions
Problem

• Given input-output data \( \mathbf{D}: \mathbb{R}^d \times \mathbb{R} \) and error bound \( \delta \), learn a piecewise affine model \( \mathbf{f}: \mathbb{R}^d \rightarrow \mathbb{R} \)

• \[ | \mathbf{f}(a) - b | \leq \delta \text{ for all points } (a, b) \in \mathbf{D} \]

• Minimize size of model
  • Minimize number of linear functions and size of predicates

• Two problems
  • Learn linear functions
  • Learn guard predicates for associated regions

• Minimizing the model size - NP-Hard
  • Best effort solution
Learning Linear Functions

- Select random point \( p \)
- Fit a linear function that covers points in neighborhood of \( p \)
- Remove the covered points and repeat
Learning Linear Functions

\[ l_1: 0 \]

\[ l_2: y - x \]

\[ l_3: \frac{(y - x + 16)}{5} \]
Learning Linear Functions

l1: 0
l2: y - x
l3: (y - x + 16)/5
Learning Linear Functions

\[ l_1: 0 \]

\[ l_2: y - x \]

\[ l_3: (y - x + 16)/5 \]
Learning Linear Functions

\[ l_1: 0 \]
\[ l_2: y - x \]
\[ l_3: \frac{y - x + 16}{5} \]
Learning Guard Predicates

l1: 0
l2: y - x
l3: (y - x + 16)/5
Learning Guard Predicates

l1: 0
l2: y - x
l3: (y - x + 16)/5
Learning Guard Predicates

l1: 0
l2: y - x
l3: (y - x + 16)/5

Positive
Negative
Learning Guard Predicates

- Problem of classification
Learning Guard Predicates

- Problem of classification
- SVM, Logistic Regression
  - Small model
  - Imprecise
Learning Guard Predicates

- Problem of classification
- SVM, Logistic Regression
  - Small model
  - Imprecise
- Decision Trees, Nearest Neighbor
  - Precise
  - Huge model
Learning Guard Predicates

- Propose a counterexample guided approach
- Learns **precise** classifier
- Learns **small** predicate
- Inspired by “Beautiful Interpolants” by Albarghouthi and McMillan
Learning Guard Predicates
Learning Guard Predicates

2y + 1 < 0
Learning Guard Predicates

\[ 2y + 1 < 0 \]
Learning Guard Predicates

\[ 2y + 1 < 0 \]
Learning Guard Predicates

\[ 3y - 4x - 10 < 0 \]
Learning Guard Predicates

\[ 3y - 4x - 10 < 0 \]
Learning Guard Predicates

\[3y - 4x - 10 < 0\]
Learning Guard Predicates

\[ 4y - 3x - 16 < 0 \]
Learning Guard Predicates

\[ 4y - 3x - 16 < 0 \]
Learning Guard Predicates

\[ 4y - 3x - 16 < 0 \]
Learning Guard Predicates

\[4y - 3x - 16 < 0 \text{ and } 2y - 3x + 6 > 0\]
Learning Guard Predicates

\[ x < -1.5 \text{ and } y < -1.5 \]

or

\[ x > 1.5 \text{ and } y > 1.5 \]
## Evaluation

<table>
<thead>
<tr>
<th>Model</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mosaic</td>
<td>0.516</td>
<td>0.61</td>
<td>0.969</td>
<td>2.248</td>
</tr>
<tr>
<td>LinReg + DTree</td>
<td>0.551</td>
<td>0.642</td>
<td>1.142</td>
<td>3.114</td>
</tr>
<tr>
<td>LinReg + SVM</td>
<td>0.571</td>
<td>0.949</td>
<td>1.159</td>
<td>3.585</td>
</tr>
<tr>
<td>Ferrari-Trecate</td>
<td>0.731</td>
<td>1.122</td>
<td>1.534</td>
<td>3.058</td>
</tr>
</tbody>
</table>

Data from Pick and Place Machines (Averaged over 15 runs)
Conclusion

• Novel application of Verification to Machine Learning
  • Scalable techniques for precise reasoning
    • Counter-example guided strategies
    • Abstraction-refinement
    • Compositional reasoning
  • Useful when mistakes in data are costly
Mosaic

• Project Webpage: 
  http://www.seas.upenn.edu/~nimits/mosaic/

• Source Code and Benchmarks: 
  https://github.com/nimit-singhania/mosaic