The following is intended to provide some insight into a line of research in Oracle Labs. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. Oracle reserves the right to alter its development plans and practices at any time, and the development, release, and timing of any features or functionality described in connection with any Oracle product or service remains at the sole discretion of Oracle. Any views expressed in this presentation are my own and do not necessarily reflect the views of Oracle.
Biscotti and Cannoli
An Initial Exploration into Machine Learning for the Purposes of Finding Bugs in Source Code

Tim Chappell*, Cristina Cifuentes, Paddy Krishnan, Shlomo Geva*
Queensland University of Technology*, Oracle Labs
November 15, 2016
Project Overview

• Imagine if machine learning could detect bugs for us in software
  – With good precision
  – With good recall
  – With good performance
  – And beat Parfait and other static code analysis tools at finding bugs in software

• This Friday Project is an investigation into what is feasible in this space
  – Project started in February 2016
Machine Learning is the subfield of computer science that “gives computers the ability to learn without being explicitly programmed” (Arthur Samuel, 1959)

— Wikipedia
Machine Learning Approaches

Supervised Learning

• The learning algorithm is given example inputs and their desired outputs, with the goal to learn a general rule that maps inputs to outputs

Unsupervised Learning

• The learning algorithm infers structure in its inputs to produce the outputs of interest
Machine Learning Approaches

Supervised Learning

• The learning algorithm is given example inputs and their desired outputs, with the goal to learn a general rule that maps inputs to outputs

• Two tools
  – Biscotti
  – Cannoli

Unsupervised Learning

• The learning algorithm infers structure in its inputs to produce the outputs of interest
Supervised Learning – Classifiers and Decision Trees

Diagram from: http://sebastianraschka.com/images/blog/2014/intro_supervised_learning/decision_tree_1.png
2D Decision Boundary
Iris Dataset Example

• Made use of two petal features (length and width)

• Classified into three classes of Irises (setosa, versicolor, virginica)
Abstracting The Iris Dataset Example

- **Features** are inputs
- **Classes** are outputs
- Dataset needs to contain features and classes
Abstracting The Iris Dataset Example

- **Features** are inputs
- **Classes** are outputs
- Dataset needs to contain features and classes
- For bugs in source code
  - Features == ?
  - Classes == bug type
Biscotti

Features → Biscotti (training) → Biscotti model → results.
### Biscotti’s Feature Selection

**Complexity of the code**
- Cyclomatic complexity
- Def-use chains
- # edges
- # knots
- Length of code
- Line count
- Nesting level
- Vocabulary
- Function start line
- Function end line
- …

**Text features**
- !
- ( 
- )
- ,
- 00
- 1
- …
- FILE
- …
- Input
- Logged
- …

**Intermediate Code instruction frequency**
- add
- alloca
- and
- ashr
- bitcast
- br
- call
- extractvalue
- fadd
- …
Biscotti’s Feature Selection

- Intermediate Code 2-grams
  - alloca-alloca
  - store-store
  - store-br
  - br-load
  - load-icmp
  - icomp-br
  - br-br
  - ...

- Clang –analyze output
  - Array-subscript-is-undefined
  - Bad-free
  - Dead-assignment
  - Dead-increment
  - Dereference-of-null-pointer
  - Double-free
  - Function-call-argument-is-an-uninitialized-value
  - Memory-leak
  - Out-of-bound-array-access
  - ...

- Output from other Static Code Analysis tools
  - Parfait
  - Splint
  - UNO
Feature Selection – Dimensionality Reduction

8,190 features reduced to 500
Feature Selection – Dimensionality Reduction

• LOONNE: leave one out nearest neighbour error
  – Removes the least distinguishing feature at each step by minimising the global error

Given a feature set FS,

\[
\text{GlobalError}(FS) = \text{Sum of all misclassifications for FS}
\]

LOONNE removes feature f if

for all other features \( f' \),

\[
\text{GlobalError}(FS-\{f\}) > \text{GlobalError}(FS-\{f'\})
\]
Biscotti’s Classification Algorithm

• Random Forests
  – Forest of 100 randomly-seeded decision trees using random subsets of the feature set
  – The outcomes of the decision trees are combined to produce a single outcome for each result
  – Useful when no natural probabilistic distribution amongst features

• Granularity of analysis: function level
  – Line number level too fine for initial experimentation
# Training and Test Datasets: BegBunch’s Accuracy Suites

Bugs are marked up in the suites

<table>
<thead>
<tr>
<th>BegBunch Suite</th>
<th>Type of Benchmark</th>
<th>Average Non-Commented Lines of Code</th>
<th># Functions</th>
<th># and Types of Bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cigital</td>
<td>Synthetic</td>
<td>15</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>Samate</td>
<td>Synthetic</td>
<td>20</td>
<td>2,366</td>
<td>Buffer overruns: 1709</td>
</tr>
<tr>
<td>Iowa</td>
<td>Synthetic</td>
<td>31</td>
<td>1,686</td>
<td>Memory leaks: 196</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Uninitialised vars: 131</td>
</tr>
<tr>
<td>OracleLabs-Accuracy*</td>
<td>Real</td>
<td>917</td>
<td>547</td>
<td></td>
</tr>
</tbody>
</table>

Trained with 4-fold cross-validation over test datasets

* These bug kernels were extracted from open source code, including relevant flow of control.
## Results ML (Biscotti) vs Static Code Analysis Tools

<table>
<thead>
<tr>
<th>Type of Bug</th>
<th>Splint</th>
<th>Parfait</th>
<th>Biscotti</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>500 features</td>
</tr>
<tr>
<td>Buffer overrun</td>
<td>581/999 TP</td>
<td>885/999</td>
<td>910/999</td>
</tr>
<tr>
<td></td>
<td>(58%)</td>
<td>(89%)</td>
<td>(91%)</td>
</tr>
<tr>
<td></td>
<td>343 FP</td>
<td>14 FP</td>
<td>262 FP</td>
</tr>
<tr>
<td>Memory leak</td>
<td>-</td>
<td>9/42</td>
<td>17/42</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(21%)</td>
<td>(40%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 FP</td>
<td>3 FP</td>
</tr>
<tr>
<td>Uninitialised variable</td>
<td>12/15 TP</td>
<td>13/15</td>
<td>8/15</td>
</tr>
<tr>
<td></td>
<td>(80%)</td>
<td>(87%)</td>
<td>(53%)</td>
</tr>
<tr>
<td></td>
<td>54 FP</td>
<td>11 FP</td>
<td>0 FP</td>
</tr>
</tbody>
</table>

Evaluated using 4-fold cross-validation over BegBunch dataset
What Did Biscotti Learn?

• Top 10 features
  – [Parfait] buffer overflow
  – [Parfait] read outside array bounds
  – [Splint] fresh storage not released before return
  – [Text]
  – [Complexity] function end line
  – [Parfait] uninitialised variable
  – [Splint] function exported but not used outside
  – [Splint] for body not block
  – [Text] contents

• Training datasets have high number of synthetic benchmarks
  – Biscotti learnt to rely on features that don’t make sense (e.g., end of line)

• None of the features are representative of a bug
# Results ML (Biscotti) vs Static Code Analysis Tools

<table>
<thead>
<tr>
<th>Type of Bug</th>
<th>Splint</th>
<th>Parfait</th>
<th>Biscotti</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>500 features</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1-2-grams + complexity features (553 features)</td>
</tr>
<tr>
<td>Buffer overrun</td>
<td>581/999 TP (58%)</td>
<td>885/999 (89%)</td>
<td>910/999 (91%)</td>
</tr>
<tr>
<td></td>
<td>343 FP</td>
<td>14 FP</td>
<td>262 FP</td>
</tr>
<tr>
<td>Memory leak</td>
<td>-</td>
<td>9/42 (21%)</td>
<td>17/42 (40%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 FP</td>
<td>3 FP</td>
</tr>
<tr>
<td>Uninitialised variable</td>
<td>12/15 TP (80%)</td>
<td>13/15 (87%)</td>
<td>8/15 (53%)</td>
</tr>
</tbody>
</table>

Evaluated using 4-fold cross-validation over BegBunch dataset
Biscotti Conclusions

• Need more datasets of representative bugs; marked up
  – I.e., not synthetic benchmarks

• The crux of supervised learning is determining the right set of features
  – What features make a bug a bug?
“Deep Learning succeeds when it’s difficult to figure out what features you want to use in your classifier”
Machine Learning Approaches

**Supervised Learning**

- The learning algorithm is given example inputs and their desired outputs, with the goal to learn a general rule that maps inputs to outputs

- Two tools
  - Biscotti
  - Cannoli

**Unsupervised Learning**

- The learning algorithm infers structure in its inputs to produce the outputs of interest
Supervised Learning – Convolutional Neural Networks

3-layer neural network

http://cs231n.github.io/assets/nn1/neural_net2.jpeg
Supervised Learning – Convolutional Neural Networks

Convolutional neural network

http://cs231n.github.io/assets/cnn/cnn.jpeg
Cannoli
Cannoli’s Architecture

The quick brown fox jumped over the lazy dogs
Training Dataset: BegBunch’s Scalability Suites

Bugs are not marked up in these suites

<table>
<thead>
<tr>
<th>BegBunch Suite</th>
<th>Average Non-Commented Lines of Code</th>
<th># Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calysto</td>
<td>87,636</td>
<td>11,214</td>
</tr>
<tr>
<td>OracleLabs-Scalability</td>
<td>394,739</td>
<td>53,448</td>
</tr>
</tbody>
</table>
# Results ML (Cannoli) vs Static Code Analysis Tools

## Training on Scalability Suite (50/50 split), testing on OpenSolaris ONNV b93* (no split)

<table>
<thead>
<tr>
<th>Type of Bug</th>
<th>Parfait v0.4.1</th>
<th>Cannoli</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer overrun</td>
<td>221 TP, 81 FP</td>
<td>213/221 TP, 56095 FP</td>
</tr>
<tr>
<td>Memory leak</td>
<td>506 TP, 94 FP</td>
<td>497/506 TP, 47414 FP</td>
</tr>
</tbody>
</table>

---

Training on Scalability Suites using Parfait v1.7.1.3 results as ground truth

* 168,666 functions
Results ML (Cannoli) vs Static Code Analysis Tools
Training on BegBunch’s Accuracy Suites (no split), testing on OpenSolaris ONNV b93*

<table>
<thead>
<tr>
<th>Type of Bug</th>
<th>Parfait v0.4.1</th>
<th>Cannoli</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer overrun</td>
<td>221 TP, 81 FP</td>
<td>23/221 TP, 9146 FP</td>
</tr>
<tr>
<td>Memory leak</td>
<td>506 TP, 94 FP</td>
<td>0/506 TP, 174 FP</td>
</tr>
<tr>
<td>Uninitialised variable</td>
<td>30 TP, 16 FP</td>
<td>0/30 TP, 153 FP</td>
</tr>
</tbody>
</table>

Training on Scalability Suites using Parfait v1.7.1.3 results as ground truth

* 168,666 functions
What Did Cannoli Learn?
Cannoli Conclusions

• Image recognition techniques not ideal for source code analysis

• Results from black-box techniques are not very useful for bug detection
  – No bug traces can be derived for developers to understand the results of the tool
## Summary Of The State Of The Art

<table>
<thead>
<tr>
<th>Paper</th>
<th>Venue-Year</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brun, Ernst</td>
<td>ICSE-04</td>
<td>Properties inferred using both buggy and fixed code</td>
</tr>
<tr>
<td>Yamaguchi et al.</td>
<td>ACSAC-12</td>
<td>Extrapolate vulnerabilities from known vulnerabilities using AST representations</td>
</tr>
<tr>
<td>ALETHEIA</td>
<td>CCS-14</td>
<td>Statistical analyses to predict “rare” vulnerabilities; tunable to focus on FP elimination/TP detection. Basic features (per Biscotti)</td>
</tr>
<tr>
<td>JSNice</td>
<td>POPL-15</td>
<td>Use program dependence graphs and statistical prediction to deobfuscate JavaScript code</td>
</tr>
<tr>
<td>Mou et al.</td>
<td>AAAI-16</td>
<td>Convolutional Neural Networks using AST representation to identify code similarities</td>
</tr>
<tr>
<td>Wang et al.</td>
<td>ICSE-16</td>
<td>Use Deep Belief Networks and AST representation to detect within project and cross project defects</td>
</tr>
<tr>
<td>Greico et al.</td>
<td>CODASPY-16</td>
<td>Use static and dynamic features (state of memory) to detect vulnerabilities</td>
</tr>
</tbody>
</table>
Summary

• Two ML approaches were implemented to find bugs in C code
  – Bisco@: supervised learning using a random forest of decision trees and LOONNE
  – Cannoli: supervised learning using a convolutional neural network

• Both learned “something”
  – But results are tied to the datasets used; i.e., doesn’t learn to find bugs in unseen code

• Biscotti captures syntactic features of the program
  – *Need to capture semantic features*

• Need a lot more representative data
Future Plans

1. Create enough data for datasets
   – Representative proportion of buggy vs non-buggy code
   – Representative number of bugs for each bug type of interest
   – Fixed version of each buggy example

2. Explore different approaches to encode semantics
   – Use of buggy vs fixed code to determine features of interest [Ernst’04]
   – Use of recurrent neural network with long short-term memory (LSTM)
Q&A
Integrated Cloud
Applications & Platform Services