MACHINE LEARNING FOR SOFTWARE MAINTAINABILITY

Anna Corazza, Sergio Di Martino, Valerio Maggio
Alessandro Moschitti, Andrea Passerini, Giuseppe Scanniello, Fabrizio Silverstri

JIMSE 2012
August 28, 2012 Montpellier, France
"A software system must be continuously adapted during its overall life cycle or it progressively becomes less satisfactory" (cit. Lehman's Law of Software Evolution)

- Software Maintenance is one of the most expensive and time consuming phase of the whole life cycle
  - Anticipating the Maintenance operations reduces the cost
  - 85%-90% of the total cost are related to the effort necessary to comprehend the system and its source code [Erlikh, 2000]
• Provide *models* and *views* representing the relationships among different *software artifacts*

• *Clustering of Software Artifacts*

• **Advantages:**
  
  • To aid the comprehension
  
  • To reduce maintenance effort
• Provide models and views representing the relationships among different software artifacts

• Clustering of Software Artifacts

• Advantages:
  • To aid the comprehension
  • To reduce maintenance effort
• Provide *models and views* representing the relationships among different *software artifacts*

• **Clustering of Software Artifacts**

• **Advantages:**
  • To aid the comprehension
  • To reduce maintenance effort
Software Artifacts may be analyzed at **different** levels of abstractions.
Software Artifacts may be analyzed at different levels of abstractions.
Software Artifacts may be analyzed at **different levels of abstractions**.

The different levels of abstractions lead to different **analysis tasks**:

- **Identification of functional modules and their hierarchical arrangement**
  - i.e., Clustering of Software classes

- **Identification of Code Clones**
  - i.e., Clustering of Duplicated code fragments (blocks,
SOFTWARE ARTIFACTS CLUSTERING

**Problem:** Definition of a proper similarity measure to apply in the clustering analysis, which is able to exploit the considered representation of software artifacts

- Mine information directly from the source code:
- Exploit the **syntactic/lexical information** provided in the source code text
- Exploit the **relational information** between artifacts
- e.g., Program Dependencies

```java
class WrappedClassLoader extends ClassLoader {
    private Bundle bundle;
    public WrappedClassLoader(Bundle bundle) {
        super();
        this.bundle = bundle;
    }
    /* (non-Javadoc)
     * @see java.lang.ClassLoader#findClass(java.lang.String)
     */
    public Class findClass(String name) throws ClassNotFoundException {
        return bundle.loadClass(name);
    }
    /* (non-Javadoc)
     * @see java.lang.ClassLoader#findResource(java.lang.String)
     */
    public URL findResource(String name) {
        return bundle.getResource(name);
    }
    /* (non-Javadoc)
     * @see java.lang.ClassLoader#findResources(java.lang.String)
     */
    protected Enumeration findResources(String name) throws IOException {
        return bundle.getResources(name);
    }
```
MINING LARGE REPOSITORIES

- Analysis of **large** and **complex systems**
- Solutions and algorithms must be able to **scale** efficiently
  
  (*in the large and in the many*)
ADVANCED MACHINE LEARNING FOR SOFTWARE MAINTENANCE

**Idea:** *Definition of Machine Learning techniques to mine information from the source code*

- Combine different kind of information (lexical and structural)
  - Application of **Kernel Methods** to software artifacts
- Provide flexible and computational effective solutions to analyze large data sets
**ADVANCED MACHINE LEARNING FOR SOFTWARE MAINTENANCE**

**Idea:** *Definition of Machine Learning techniques to mine information from the source code*

- Combine different kind of information (lexical and structural)
  - Application of **Kernel Methods** to software artifacts
- Provide flexible and computational effective solutions to analyze large data sets

**Advanced Machine Learning**

- Learning with syntactic/semantic information (Natural Language Processing)
- Learning in relational domains (Structured-output learning, Logic Learning, Statistical Relational Learning)
A **Kernel** is a function between (arbitrary) pairs of entities.

It can be seen as a kind of **similarity measure**.

Based on the idea that structured objects can be described in terms of their constituent parts.

Generalize the computation of the dot product to arbitrary domains.

Can be easily tailored to specific domains.

- **Tree Kernels**
- **Graph Kernels**
KERNELS FOR STRUCTURES

Computation of the **dot product** between (Graph) Structures

\[
K \left( \begin{array}{c}
\text{structure 1} \\
\text{structure 2} \\
\text{structure 3}
\end{array} \right) = 26
\]
Kernels for Languages

- Parse Trees represent the syntactic structure of a sentence.
- Tree Kernels can be used to measure the similarity between parse trees.
KERNALS FOR LANGUAGES

- Parse Trees represent the syntactic structure of a sentence
- Tree Kernels can be used to measure the similarity between parse trees

KERNALS FOR SOURCE CODE

- Abstract Syntax Trees (AST) represent the syntactic structure of a piece of code
- Research on Tree Kernels for NLP carries over to AST (with adjustments)
KERNELS FOR PARSE TREE

Sentence

John hit the ball

I like you

Parse Tree

V

hit

the

ball

S

NP

John

VP

NP

Det

N

S

NP

VP

Parse Tree Kernel

VP

NP

NP

Det

N

S

NP

VP

VP

NP

V

NP
KERNELS FOR AST

CODE

while (x < y) {
  x = x + 1
  y = y - 1
}

while (a < b) {
  a = a * 2
  b = b / 2
}

AST

while
  <
  x
  y
  =
  x +
  y -
  x 1
  y 1

block

while
  >
  a
  b
  =
  a *
  b /
  a 2
  b 2

block

AST KERNEL

while
  <
  block

while
  >
  block

<

=}

a

=}

b

<

=}

a *

=}

b /

<

=}

a 2

=}

b 2

=}

a

=}

b

=}

a

=}

b
Idea: Any learning algorithm relying on similarity measure can be used.

- **Supervised Learning**
  - Binary Classification
  - Multi-class Classification
  - Ranking

- **Unsupervised Learning**
  - Clustering
  - Anomaly Detection

KERNEL MACHINES
KERNEL MACHINES FOR CONE DETECTION

- **Supervised Learning**
  - Pairwise classifier: predict if a pair of fragments is clone

- **Unsupervised Learning**
  - Clustering: cluster together all candidate clones
KERNEL FOR CLONES

CODE

while (x < y) {
    x = x + 1
    y = y - 1
}

while (b > a) {
    a = a + 1
    b = b - 1
}

AST

while
    <
    x
    y
    block
        =
        x
        +
        y
        -
            x 1
            y 1

block

AST KERNEL

while
    <
    x
    y
    block
        =
        x
        +
        y
        -
            y 1
            x 1

block

while
    >
    b
    a
    block
        =
        a
        +
        b
        -
            a
            1
            b
            1
KERNEL LEARNING

- Construct a number of candidate kernels with different characteristics
  - *e.g.*, *Ignore variables names or not*

- Employ kernel learning approaches which learn a weighted combination of candidate kernels

- Useless/harmful kernels will get zero weight and will be discarded in the final model
Supervised Clustering

- Exploit information on already annotated pieces of software
- Training examples are software projects/portions with annotation on existing clones (clustering)
- A learning model uses training examples to refine the similarity measure for correctly clustering novel examples
SUMMARY

- Software has a rich structure and heterogeneous information
- Advanced Machine learning approaches are promising for exploiting such information
- **Kernel Methods** are natural candidate
  - e.g., see the analogy between NLP parse trees and AST
- Many applications:
  - architecture recovery, code clone detection, vulnerability detection ....
CASE STUDY: KERNELS FOR CLONES
Goal: “Identify and group all duplicated code fragments/functions”

Copy&Paste programming

Taxonomy of 4 different types of clones

Program Text similarities and Functional similarities

Clones affect the reliability and the maintainability of a software system
Kernels for Structured Data:

- The source code could be represented by many different data structures

- **Abstract Syntax Tree (AST)**
  - Tree structure representing the syntactic structure of the different instructions of a program (function)

- **Program Dependencies Graph (PDG)**
  - (Directed) Graph structure representing the relationship among the different statement of a program
int function (int parameter) {
    int k = 10;

    printf("Hello, this is the function");

    int i = 0;
    while (i < 7) {
        i++;
        // do something cool
    }
}

AST embeds both **Syntactic** and **Lexical** Information

- Program Instructions
- Name of Variables, Literals…
int function (int parameter) {
    int k = 10;
    printf("Hello, this is the function");
    int i = 0;
    while (i < 7) {
        i++;
        // do something cool
    }
}

AST embeds both **Syntactic** and **Lexical** Information

- Program Instructions
- Name of Variables, Literals...
int function (int parameter) {
    int k = 10;
    printf("Hello, this is the function");
    int i = 0;
    while (i < 7) {
        i++;
        // do something cool
    }
}

AST embeds both **Syntactic** and **Lexical** Information

- Program Instructions
- Name of Variables, Literals...
int function (int parameter) {
    int k = 10;
    printf("Hello, this is the function");
    int i = 0;
    while (i < 7) {
        i++;
        // do something cool
    }
}
int function (int parameter) {
  int k = 10;

  printf("Hello, this is the function");

  int i = 0;
  while (i < 7) {
    i++;
    // do something cool
  }
}

• **Nodes** correspond to instructions

• **Edges** represent relationships between couple of nodes
int function (int parameter) {
    int k = 10;
    printf("Hello, this is the function");
    int i = 0;
    while (i < 7) {
        i++;
        // do something cool
    }
}

- **Nodes** correspond to instructions
- **Edges** represent relationships between couple of nodes
int function (int parameter) {
    int k = 10;
    printf("Hello, this is the function");
    int i = 0;
    while (i < 7) {
        i++;
        // do something cool
    }
}

- **Nodes** correspond to instructions

- **Edges** represent relationships between couple of nodes
```c
int function (int parameter) {
    int k = 10;
    printf("Hello, this is the function");
    int i = 0;
    while (i < 7) {
        i++;
        // do something cool
    }
}
```

- **Nodes** correspond to instructions
- **Edges** represent relationships between couple of nodes
int function (int parameter) {
  int k = 10;
  printf("Hello, this is the function");
  int i = 0;
  while (i < 7) {
    i++;
    // do something cool
  }
}

- **Nodes** correspond to instructions
- **Edges** represent relationships between couple of nodes
• **Two Types** of Nodes

  • *Control Nodes* (Dashed ones)
    - e.g., if - for - while - function calls...
  
  • *Data Nodes*
    - e.g., expressions - parameters...
Two Types of Nodes

Control Nodes (Dashed ones)
- e.g., if - for - while - function calls...

Data Nodes
- e.g., expressions - parameters...

Two Types of Edges (i.e., dependencies)

Control edges (Dashed ones)
- Data edges
The definition of a new Kernel for a Structured Object requires the definition of:

- Set of features to annotate each part of the object
- A Kernel function to measure the similarity on the smallest part of the object
  - *e.g.*, *Nodes of AST and Graphs*
- A Kernel function to apply the computation on the different (sub)parts of the structured object
Features: each node is characterized by a set of 4 features

Instruction Class
- i.e., LOOP, CONDITIONAL_STATEMENT, CALL

Instruction
- i.e., FOR, IF, WHILE, RETURN

Context
- i.e., Instruction Class of the closer statement node

Lexemes
- Lexical information gathered (recursively) from leaves
Tree Kernels for AST

- **Features**: each node is characterized by a set of 4 features

- **Instruction Class**
  - i.e., LOOP, CONDITIONAL_STATEMENT, CALL

- **Instruction**
  - i.e., FOR, IF, WHILE, RETURN

- **Context**
  - i.e., Instruction Class of the closer statement node

- **Lexemes**
  - Lexical information gathered (recursively) from leaves
**Goal:** Identify the maximum isomorphic Tree/Subtree

**Comparison of blocks to each other**

**Blocks:** Atomic unit for (sub) tree considered
• **Goal:** Identify the maximum isomorphic Tree/Subtree

• **Comparison of blocks to each other**

• **Blocks:** Atomic unit for (sub) tree considered

**KERNELS FOR CODE STRUCTURES:** AST

---

**TREE KERNELS FOR AST**
Features of nodes:

- Node Label
  - i.e., WHILE, CALL-SITE, EXPR, ...
- Node Type
  - i.e., Data Node or Control Node

Features of edges:

- Edge Type
  - i.e., Data Edge or Control Edge
Features of nodes:

- **Node Label**
  - i.e., WHILE, CALL-SITE, EXPR, ...

- **Node Type**
  - i.e., Data Node or Control Node

Features of edges:

- **Edge Type**
  - i.e., Data Edge or Control Edge
GRAPH KERNELS
FOR PDG

• **Goal:** Identify common subgraphs

• **Selectors:** Compare nodes to each others and explore the subgraphs of only “compatible” nodes (i.e., Nodes of the same type)

• **Context:** The subgraph of a node (with paths whose lengths are at most L to avoid loops)
**Graph Kernels for PDG**

- **Goal:** Identify common subgraphs
- **Selectors:** Compare nodes to each other and explore the subgraphs of only “compatible” nodes (i.e., Nodes of the same type)
- **Context:** The subgraph of a node (with paths whose lengths are at most $L$ to avoid loops)
**Graph Kernels for PDG**

- **Goal:** Identify common subgraphs
- **Selectors:** Compare nodes to each other and explore the subgraphs of only “compatible” nodes (i.e., Nodes of the same type)
- **Context:** The subgraph of a node (with paths whose lengths are at most L to avoid loops)
**Goal:** Identify common subgraphs

**Selectors:** Compare nodes to each others and explore the subgraphs of only “compatible” nodes (i.e., Nodes of the same type)

**Context:** The subgraph of a node (with paths whose lengths are at most L to avoid loops)
• Comparison of results with other two clone detector tools:
  • AST-based Clone detector
  • PDG-based Clone Detector
• Comparison of results with other two clone detector tools:
  • AST-based Clone detector
  • PDG-based Clone Detector
• No publicly available clone detection dataset
• Comparison of results with other two clone detector tools:
  • AST-based Clone detector
  • PDG-based Clone Detector

• No publicly available clone detection dataset
  • No unique set of analyzed open source systems
Comparison of results with other two clone detector tools:

- AST-based Clone detector
- PDG-based Clone Detector

No publicly available clone detection dataset

- No unique set of analyzed open source systems
- Usually clone results are not available
Comparison of results with other two clone detector tools:

- AST-based Clone detector
- PDG-based Clone Detector

No publicly available clone detection dataset

- No unique set of analyzed open source systems
- Usually clone results are not available

Two possible strategies:
• Comparison of results with other two clone detector tools:
  • AST-based Clone detector
  • PDG-based Clone Detector
• No publicly available clone detection dataset
  • No unique set of analyzed open source systems
  • Usually clone results are not available
• Two possible strategies:
  • To automatically modify an existing system with randomly generated clones
• Comparison of results with other two clone detector tools:
  • AST-based Clone detector
  • PDG-based Clone Detector

• No publicly available clone detection dataset
  • No unique set of analyzed open source systems
  • Usually clone results are not available

• Two possible strategies:
  • To automatically modify an existing system with randomly generated clones
  • Manual classification of candidate results
EMPIRICAL EVALUATION

BENCHMARKS AND DATASET

<table>
<thead>
<tr>
<th>Project</th>
<th>Size (KLOC)</th>
<th># PDGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache-2.2.14</td>
<td>343</td>
<td>3017</td>
</tr>
<tr>
<td>Python-2.5.1</td>
<td>435</td>
<td>5091</td>
</tr>
</tbody>
</table>

- Comparison with another Graph-based clone detector
- MeCC (ICSE2011)
- **Baseline** Dataset
  - Results provided by MeCC
- **Extended** Dataset
  - Extension of Clones results by manual evaluation of candidate clones
  - Agreement rate calculation between the evaluators
EMPIRICAL EVALUATION OF TREE KERNEL FOR AST

- Comparison with another (pure) AST-based clone detector
- Comparison on a system with randomly seeded clones

Results refer to clones where code fragments have been modified by adding/removing or changing code statements.
Clone results with different similarity thresholds

- Precision
- Recall
- F1
RESULTS WITH APACHE 2.2.14

<table>
<thead>
<tr>
<th>Threshold</th>
<th>#Clones in the Baseline</th>
<th>#Clones in the Extended Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>874</td>
<td>1089</td>
</tr>
<tr>
<td>0.99</td>
<td>874</td>
<td>1514</td>
</tr>
</tbody>
</table>
# RESULTS WITH PYTHON 2.5.2

<table>
<thead>
<tr>
<th>Threshold</th>
<th>#Clones in the Baseline</th>
<th>#Clones in the Extended Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>858</td>
<td>1066</td>
</tr>
<tr>
<td>0.99</td>
<td>858</td>
<td>2119</td>
</tr>
</tbody>
</table>

---

**Evaluation Graph Kernels for PDG**

- **Precision**: Baseline vs. Extended
  - Threshold 0.99: Baseline 0.99, Extended 1.00
  - Threshold 1.00: Baseline 0.99, Extended 1.00

- **Recall**: Baseline vs. Extended
  - Threshold 0.99: Baseline 0.99, Extended 1.00
  - Threshold 1.00: Baseline 0.99, Extended 1.00
CHALLENGES AND OPPORTUNITIES

- Learning Kernel Functions from Data Set

- Kernel Methods **advantages:**
  - flexible solution to be tailored to specific domain
  - efficient solution easy to parallelize
  - combinations of multiple kernels

- Provide a publicly available data set
THANK YOU FOR YOUR KIND ATTENTION