Heuristic analysis for detecting interesting code segments through classification

Jonathan Simmons, Jan Svacina

Problem statement

Problem: Code clone tools are

1. depended on a parser that parses the language
2. depended on a language specification that evolves

Proposed solution:

- Use classification based on deep learning
  - Label each line of code as variable assignment, method call, etc
- Group code segments through deep learning
  - Maximize interesting code snippets by maximizing entropy

Benefits:

Building and running a model is faster than writing or running a parser

Problem statement cont.

Application:
Search for some patterns in classified code

Benefits:

- Increase semantic code clone detection efficiency by reducing the codebase to be analyzed
  - We pose that while running this learning model will take time t_1 on a system S_1 and running a semantic code clone detection algorithm on system S_1 will take time t_2 and that though the overall composite time cost is $t_1 + t_2$, eventually, as the model decreases the size of S_1 and in turn $t_2 > t_3$, the new composite speed will be quicker: $t_1 + t_2 > t_1 + t_3$.
  - or that there will exist a point in which running both this model and a given semantic code clone detection algorithm in tandem will be quicker than running just a semantic code clone detection algorithm.

Proposed approach overview

1. Classify code
2. Find interesting parts of code
3. Compare interesting parts of code by standard techniques

Theoretical Classifications for Lines of Code

1. Conditional
    a. Ternary
    b. Else
    c. If
    d. Else if
2. Block end
3. Loop
    a. Do
    b. While
    c. For
    d. foreach
4. Primitive assignment
5. Object creation
    a. Via new
    b. Via method call
6. Method call
    a. Static method
    b. External static method
    c. Member function
    d. External member function
7. Lambda / other

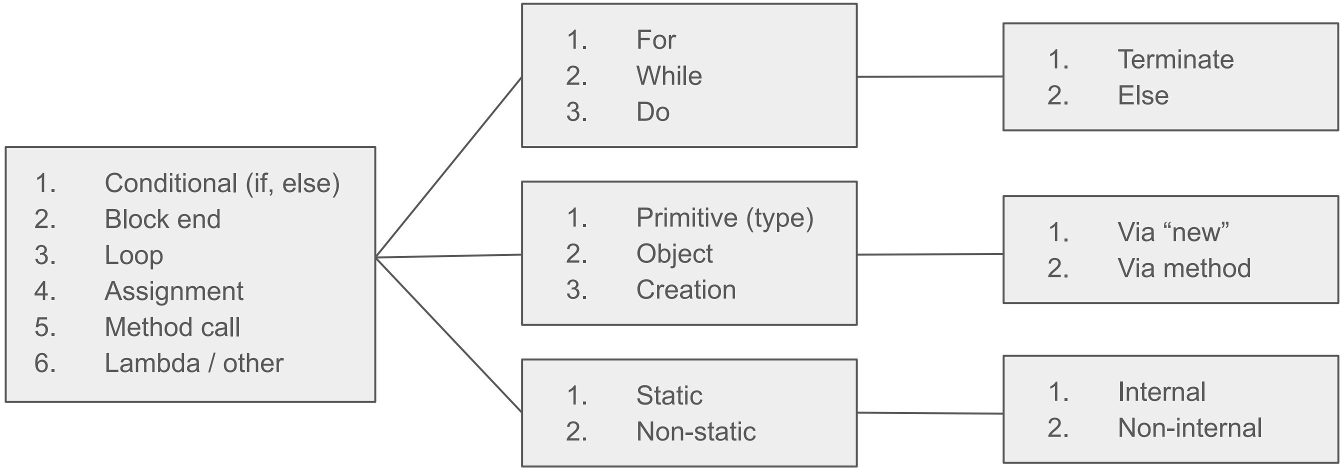
- Stereotypes per line of code
- Variable and method associations

Weigh stereotypes by how much they could impact entropy (*e.g. method calls have more behavior and information than assigning to an integer*)

Uninteresting Stereotypes
- Import statements
- Printing
- Logging

Classification of lines of code

Basis for deep learning: varying granularity



1. Label lines of code

```
double exampleMethod() {
     if (predicate) {
          int x = 2;
          internalMethodCall();
          Object o = new Object();
          Object o2 = factoryMethod(x);
          while (predicate) {
               x++;
               o.setVal(o2.getVal() / x);
               print("value: " + o.getVal());
               if (predicate) break;
          }
          print("something");
          return (double) o.getVal() + x;
     }
}
```

package::class::exampleMethod::1::declaration
package....::2::conditional::if
package....::3::assignment::primitive
package....::4::method_call::internal
package....::5::assignment::object::new
package....::6::assignment::object::method_call::internal
package....::7::loop::while
package....::8::assignment::primitive
package....::9::method_call::internal::object_call
package....::10::method_call::print
package....::11::condition::if::single_line
package....::12::end_block
package....::13::method_call::print
package....::14::return::double
package....::15::end_block
package....::16::end_block

2. Find interesting segments

Deriving interesting part out of code snippets, sanitize from the pool those segments which contain solely uninteresting segments (from the labeling).

Segments that contain highly repetitive patterns that provide no meaningful semantics (low entropy):

1. assignment::primitive
2. assignment::primitive
3. assignment::primitive
4. assignment::primitive

2. Find interesting segments

```
double exampleMethod() {
    if (predicate) {
        int x = 2;
        internalMethodCall();
        Object o = new Object();
        Object o2 = factoryMethod(x);
    }
    print("other part");
    int y = 12;
    int z = 14;
    double q = 4.0;
    String t = "" + y + " " + z + ":" + q;
    print("End method");
    return q;
}
```

Interesting

Uninteresting

```
path..::method::1::declaration
path..::method::2::conditional::if
path..::method::3::assignment::primitive
path..::method::4::method_call::internal
path..::method::5::assignment::object::new
path..::method::6::assignment::object::method::internal
path..::method::7::end_block
```

High Entropy

```
path..::method::8::method_call::print
path..::method::9::assignment::primitive
path..::method::10::assignment::primitive
path..::method::11::assignment::primitive
path..::method::12::assignment::string
path..::method::13::method_call::print
path..::method::14::return::double
path..::method::15::end_block
```

Low Entropy
-Mostly assignment
-Uninteresting stereotypes

2. Find interesting segments

How do we single out segments to determine their entropy and in turn interest value?

Greedy approach of moving the selection window up and down increasing height to get maximum entropy. If the maximum entropy was found previously at a smaller height, decrease.

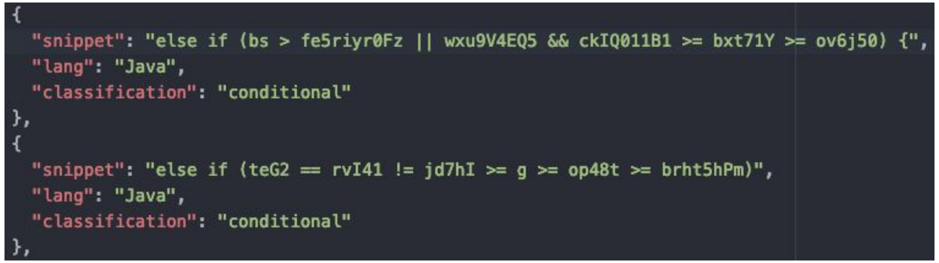
We let the algorithm find the boundary of the window by learning what is considered interesting with respect to our classifications and example code.

3. Apply standard techniques

We will then supply a standard semantic code clone detection software with this subset of code segments gathered from our deep learning model.

Current Status

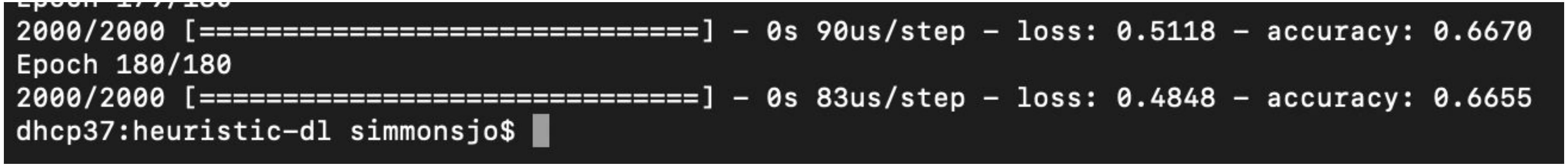
- A basic set of classifications for prototyping (single layer)
- A 4 layer sequential neural network (128 x 64 x 64 x 2)
  - The input fires up to 128 neurons for a line length maximum of 128
  - Based on token recognition
    - Benefit: Easy to produce network
    - Down-side: Token based, recognition is based solely on largeness of dataset size
  - 2 output neurons ((0.23, 0.77) -> %23 loop, %77 conditional)
- 4,000 auto generated loop / conditional statements for testing



```
{
  "snippet": "else if (bs > fe5riyr0Fz || wxu9V4EQ5 && ckIQ011B1 >= bxt71Y >= ov6j50) {",
  "lang": "Java",
  "classification": "conditional"
},
{
  "snippet": "else if (teG2 == rvI41 != jd7hI >= g >= op48t >= brht5hPm)",
  "lang": "Java",
  "classification": "conditional"
},
```

Current Status

- Our v1 prototype classifications
  - Loop, Conditional
- Once v1 is verified, expand to v2
  - Loop, Conditional, Method Call, Assignment, Creation of new object, Termination, Switch, Case, Lambda, Inline
- Once v2 is verified, expand to v3
  - Convolution / filtering for Assignment::primitive, Assignment::object, Creation::new, Creation::method etc.



```
2000/2000 [==============================] - 0s 90us/step - loss: 0.5118 - accuracy: 0.6670
Epoch 180/180
2000/2000 [==============================] - 0s 83us/step - loss: 0.4848 - accuracy: 0.6655
dhcp37:heuristic-dl simmonsjo$
```

Future Work

- The case of multiple, separate, interesting segments in a single method
    - Currently we search for the maximum possible entropy of one code segment per method. What if a method performs two distinct and low cohesion behaviors?
- Run the experiment and prove that there is some substantive speedup when running our model + traditional semantic code clone detection
- Tweak our classifications and weights for best results
- Train the model based on word associations rather than tokens
    - Experimentally we showed with predictable predicate names, accuracy dramatically increased.
        - Tokenize - lose language agnostic property
        - Word association - keep agnosticism, gain accuracy, reduce loss

Questions?