Automatic Programming Error Class Identification with Code Plagiarism-Based Clustering

Dr **Sébastien Combéfis**[1] Arnaud Schils[2]

[1]École Centrale des Arts et Métiers (ECAM)

[2]Université catholique de Louvain (UCL)

November 14, 2016

ECAM
Brussels Engineering School

[CHESE 2016, Seattle, WA, USA]

Context

- Automatic assessment of codes

  Programming learning platforms, MOOCs,
  higher education courses, competitions...

- Important part of the assessment is the feedback
  - Positive for success, to summarize what has been learned
  - Constructive for failures, to explain what is wrong

- Impossible to foresee all the possible answers from learners

  Trying to maximise the number of covered cases

Motivation

- Provide teachers with information about learners
  - Understanding learners' difficulties
  - Getting a global overview of submitted codes
- Different aspects of a program can be assesses

  From functional testing to style checks
- Not possible to anticipate all the possible submissions

  Often the same mistakes, in particular for introductory courses

Goals

- One goal for each actor of the learning
  - Identify the **main error classes** produced by the learners

    Given a set of submitted codes
  - Generate a **good feedback** to understand the failure

    Given one submitted code that fails some tests
- Offline analysis of codes for on-the-fly feedback generation

  Find the best suitable feedback given a submitted code

Similar codes

- Two similar codes exhibit some common properties

  In particular, they can contain the same error

- Several ways to measure code similarity
  - Simple string comparisons (language-agnostic)
  - Comparing the ASTs (language-dependent)

- Code plagiarism detection tools measure code similarity

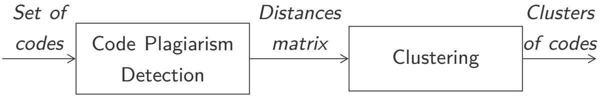
  Percentage of similar code, similar chunks identification...

Error classes identification

- Offline analysis of a set of submitted codes

  Identification of the main error classes produced by learners

- Two-step analysis from a set of code to a set of clusters
  - **Distance matrix** between codes via plagiarism detection
  - **Cluster of codes** via clustering



Clusters

- Each obtained cluster represents an error class

  Contains a "central" member which is the representative

- Several possible clusters given different configurations

  Automatically adjusted or manually by the teacher

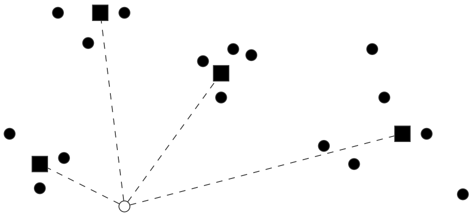
Feedback generation

- Association of one feedback for each representative (■)

  Characterizing the error class represented by the cluster (•)

- Distances between new code (∘) and representatives

  Feedback of the nearest is chosen, if proximity close enough



Experiments

- Prototype of the analysis framework to perform experiments

  Developed with the R programming language

- Codes extracted from the Code Hunt dataset

  Used 53 C# submissions from `Sector4_Level6`

- Tools used for the analysis framework
  - Code plagiarism detection tool: *JPlag*
  - Clustering: *k-medoids*, *Agglomerative Hierarchical Clustering*

JPlag

- Two configuration items to setup
  - **Programming language**: C#
  - **Sensitivity**: greatest sensitivity since codes are short
- Distance matrix $dist(i,j) = max\_possible\_similarity - similarity$

```
Language accepted: C# 1.2 Parser
Command line: -l c#-1.2 -t 1 *path to code files*
initialize ok
*n* submissions

Parsing Error in *file_1*:
  *file_1*: *error_name*
  ...

*m* submissions parsed successfully!
*n-m* parser errors!

Comparing *file_1*-*file_2*: *similarity*
...
Comparing *file_n-1*-*file_m*: *similarity*
```

k-medoids

- The medoid of each cluster is chosen as its centre

  That is the member closest to all the other ones

- Requires the number of clusters to be chosen *a priori*
  - Chosen by the teachers before launching the analysis
  - Automatically chosen to optimise a function of interest

- Increasing k until convergence of reconstruction error

  Sum of distances between elements and their medoid

Hierarchical Agglomerative Clustering

- Incrementally build clusters from bottom to top

  Starts with one cluster for each element

- At each step, merge the two closest clusters

  Ward's min. variance favour compact and spherical clusters

- Several advantages compared to k-medoids approach
  - Number of clusters should not be selected *a priori*
  - Generation of a dendrogram to select the desired clusters

Experiment #1

- *k*-medoids with $k = 4$ provides a good classification
  1. Body with one or two instructions
  2. Codes using the `switch` statement
  3. Fibonacci, `char` procesing, using `if` and `for` statements
  4. Seven different trends
- Increasing k correctly splits the clusters further

  Observed trends in the codes are correctly separated

```
using System;

public class Program {
  public static string Puzzle(string s) {
    char[] x = s.ToCharArray();
    int f1 = 1, f2 = 1, t;
    for (int i = 0; i < s.Length; i++) {
      x[i] = (x[i] - 'a' + f2) % 26 + 'a';
      t = f1;
      f1 += f2; f1 %= 26;
      f2 = t;
    }
    return new string(x);
  }
}
```

```
using System;

public class Program {
  public static string Puzzle(string s) {
    char[] x = s.ToCharArray();
    int f1 = 1, f2 = 1, t;
    for (int i = 0; i < s.Length; i++) {
      x[i] = (char)((x[i] - 'a' + f2) % 26 + 'a');
      t = f1;
      f1 += f2; f1 %= 26;
      f2 = t;
    }
    return new string(x);
  }
}
```

```
using System;

public class Program {
  public static string Puzzle(string s) {
    char[] x = s.ToCharArray();
    int f1 = 1, f2 = 1, t;
    for (int i = 0; i < s.Length; i++) {
      x[i] = (x[i] - 'a' + f2) % 26 + 'a';
      t = f1;
      f1 += f2; f1 %= 26;
      f2 = t;
    }
    return new string(x);
  }
}
```

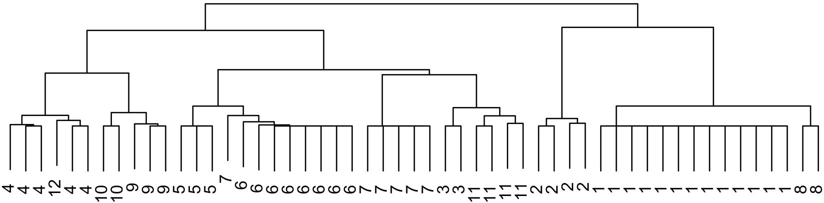
```
using System;

public class Program {
  public static string Puzzle(string s) {
    char[] arr = s.ToCharArray();
    uint fibim2 = 0, fibim1 = 0, fibi = 1;
    for(int i=0;i<arr.Length;++i){
      uint newchar = fibi % 26;
      if(arr[i] + newchar > 'z')
        arr[i] = arr[i] + newchar - 'z' + 'a' - 1;
      else
        arr[i] = (char)(arr[i] + newchar);
      fibim2 = fibim1;
      fibim1 = fibi;
      fibi = fibim1 + fibim2;
    }
    return new string(arr);
  }
}
```

Experiment #2

- Hierarchical Agglomerative provides a good classification

  Code from the same ideal cluster are together



Evaluation (1)

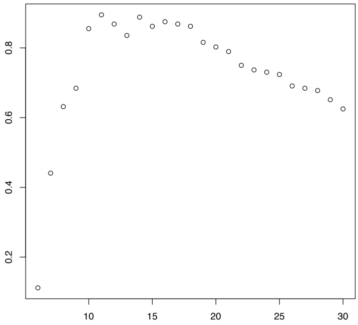
- Quality evaluation with a manual reference clustering

  Measuring the distance from the ideal clustering

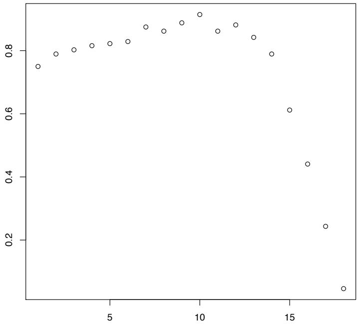
- Score of a clustering between 0 and 1

  A score of 1 *means that the clusters are exactly the same*

Evaluation (2)



k-medoids
with $k = 11$
$\rightarrow$ score $= 0.9$



Hierarchical Agglomerative
with $h = 10$
$\rightarrow$ score $= 0.91$

Conclusion (1)

- Analysis framework to generate feedback for learning
  - Offline analysis of error classes for teachers
  - On-the-fly analysis to generate feedback for learners
- Measure of code similarity with code plagiarism detection
- Error classes identification with clustering
- Preliminary experiments are promising

Conclusion (2)

- More experiments have to be performed

  With Code Hunt datasets and others

- Automatic selection of the number of clusters

  Finding criterion function to evaluate a set of clusters

- Using the framework with codes that do not compile

  Replace code plagiarism detection tools

- Evaluation of false positive and wrong feedbacks

  Could the learner be surprised with a non relevant feedback