Information Modelling & Analysis – Project 1

Claudio Maggioni

Code Repository

The code and result files part of this submission can be found at:

Repository: https://github.com/infoMA2023/project-01-god-classes-maggicl

Commit ID: **TBD**

Data Pre-Processing

God Classes

The first part of the project requires to label some classes of the *Xerces* project as "God classes" based on the number of methods each class has. Specifically, I label "God classes" the classes that have a number of methods six times the standard deviation above the the mean number of methods, i.e. where the condition

$$|M(C)| > \mu(M) + 6\sigma(M)$$

holds.

To scan and compute the number of methods of each class I use the Python library `javalang`, which implements the Java AST and parser. The Python script `./find_god_classes.py` uses this library to parse each file in the project and compute the number of methods of each class. Note that only non-constructor methods are counted (specifically the code counts the number of `method` nodes in each `ClassDeclaration` node).

Then, the script computes mean and standard deviation of the number of methods and filters the list of classes according to the condition described above. The file `god_classes/god_classes.csv` then is outputted listing all the god classes found.

The god classes I identified, and their corresponding number of methods can be found in Table 1.

Table 1: Identified God Classes

Class Name	**# Methods**
org.apache.xerces.impl.xs.traversers.XSDHandler	118
org.apache.xerces.impl.dtd.DTDGrammar	101
org.apache.xerces.xinclude.XIncludeHandler	116
org.apache.xerces.dom.CoreDocumentImpl	125

Feature Vectors

In this part of the project we produce the feature vectors used to later cluster the methods of each God class into separate clusters. We produce one feature method per non-constructor Java method in each god class.

The columns of each vector represent fields and methods referenced by each method, i.e. fields and methods actively used by the method in their method's body.

When analyzing references to fields, additional constraints need to be specified to handle edge cases. Namely, a field's property may be referenced (e.g. an access to array `a` may fetch its `length` property, i.e. `a.length`). In this cases I consider the qualifier (i.e. the field itself, `a`) itself and not its property. When the qualifier is a class (i.e. the code references a property of another class, e.g. `Integer.MAX_VALUE`) we consider the class name itself (i.e. `Integer`) and not the name of the property. Should the qualifier be a subproperty itself (e.g. in `a.b.c`, where `a.b` would be the qualifier according to `javalang`)

For methods, I only consider calls to methods of the class itself where the qualifier is unspecified or `this`. Calls to parent methods (i.e. calls like `super.something()`) are not considered.

The feature vector extraction phase is performed by the Python script `extract_feature_vectors.py`. The script takes `god_classes/god_classes.csv` as input and loads the AST of each class listed in it. Then, a list of all the fields and methods in the class is built, and each method is scanned to see which fields and methods it references in its body according to the previously described rules. Then, a CSV per class is built storing all feature vectors. Each file has a name matching to the FQDN (Fully-qualified domain name) of the class. Each CSV row refers to a method in the class, and each CSV column refers to a field, method or referenced class. A cell has the value of 1 when the method of that row references the field, method or class marked by that column, and it has the value 0 otherwise. Columns with only zeros are omitted.

Table 2 shows aggregate numbers regarding the extracted feature vectors for the god classes. Note that the number of attributes refers to the number of fields, methods or classes actually references (i.e. the number of columns after omission of 0s).

Table 2: Feature vector summary (*= used at least once)

Class Name	**# Feature Vectors**	**# Attributes***
org.apache.xerces.impl.xs.traversers.XSDHandler	106	183
org.apache.xerces.impl.dtd.DTDGrammar	91	106
org.apache.xerces.xinclude.XIncludeHandler	108	143
org.apache.xerces.dom.CoreDocumentImpl	117	63

Clustering

Algorithm Configurations

Report/comment the algorithm configurations (distance function, linkage rule, etc.). You may do so in any form you feel suited, but a short paragraph of text is probably sufficient.

Testing Various K & Silhouette Scores

(1) Report data about the clusters produced by the two algorithms at various k (#clusters, size of clusters, silhouette scores). You may use any suitable format (table, graph, ...).

(2) Briefly comment your results. What is the best configuration, and why? Anything else you observed?

Evaluation

Ground Truth

I computed the ground truth using the command .... The generated files are checked into the repository with the names ....

Comment briefly on the strengths & weaknesses of our ground truth.

Precision and Recall

Table 3: Evaluation Summary

Class Name	**Agglomerative**		**K-Means**	
	Prec.	Recall	Prec.	Recall
...	...	...	...	...

Precision and Recall, for the optimal configurations found in Section 3, are reported in Table 3.

Practical Usefulness

Discuss the practical usefulness of the obtained code refactoring assistant in a realistic setting (1 paragraph).