IMA Project 1

Tommaso Verzegnassi

April 23, 2023

This is my project report for the god-classes refractoring project. If you have any trouble running or understanding some parts of the code let me know. Information about the directory structure can be found in the readme.md file.

0 Code Repository

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

Repo: https://github.com/infoMA2023/project-01-god-classes-Tommaso9999
Commit ID: 098378248015f579ad9aa07bbd445e2534e0ac1b

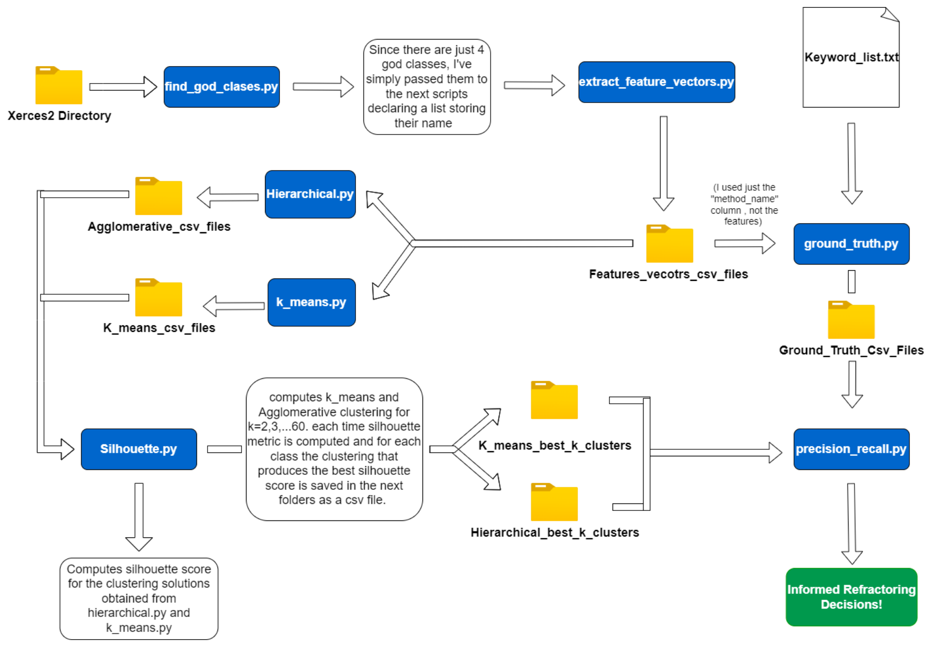
Xerces2 Directory
find_god_clases.py
Since there are just 4 god classes, I've simply passed them to the next scripts declaring a list storing their name
extract_feature_vectors.py
Keyword_list.txt
Hierarchical.py
Agglomerative_csv_files
(I used just the "method_name" column , not the features)
Features_vecotrs_csv_files
ground_truth.py
k_means.py
K_means_csv_files
Ground_Truth_Csv_Files
computes k_means and Agglomerative clustering for k=2,3,...60. each time silhouette metric is computed and for each class the clustering that produces the best silhouette score is saved in the next folders as a csv file.
Silhouette.py
K_means_best_k_clusters
Hierarchical_best_k_clusters
precision_recall.py
Computes silhouette score for the clustering solutions obtained from hierarchical.py and k_means.py
Informed Refractoring Decisions!

Figure 1: Repository structure map

1 Data Pre-Processing

1.1 God Classes

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

Class Name	**# Methods**
CoreDocumentImpl	125
DTDGrammar	101
XSDHandler	118
XIncludeHandler	116

Table 1: Identified God Classes

1.2 Feature Vectors

Table 2 shows aggregate numbers regarding the extracted feature vectors for the god classes. Every column is a field or method that belongs to the class that has been accessed at least once by one of the methods of the class that define the rows.

Class Name	**# Feature Vectors**	**# Attributes***
CoreDocumentImpl	117	67
DTDGrammar	99	121
XSDHandler	117	205
XIncludeHandler	110	186

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

2 Clustering

2.1 Algorithm Configurations

When applying k means and agglomerative clustering algorithms I have used the standard euclidean distance of Scikit-learn's library. For agglomerative clustering I have used the default 'ward' linkage rule as it minimizes the variance of the clusters merged.

2.2 Testing Various K & Silhouette Scores

(1) I applied both clustering algorithms 58 times, from k=2,3, ... 60. Each time I computed the silhouette score which is a metric that indicates the quality of the obtained clustering. More specifically this value measures simultaneously how high the inter distance between different clusters is and how low the average

intra distance within the same cluster is. A high silhouette score means that the distance of elements within the same cluster is low and the distance between different clusters is high meaning that the clustering is of high quality. Below, for both algorithms, you can find line charts displaying silhouette scores for different K values for each class.

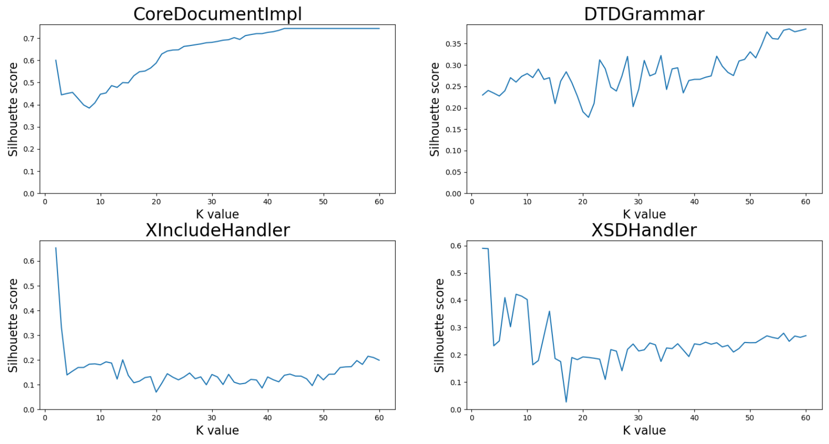

Figure 2: Silhouette scores of k means clustering for k=2,3, ..., 60

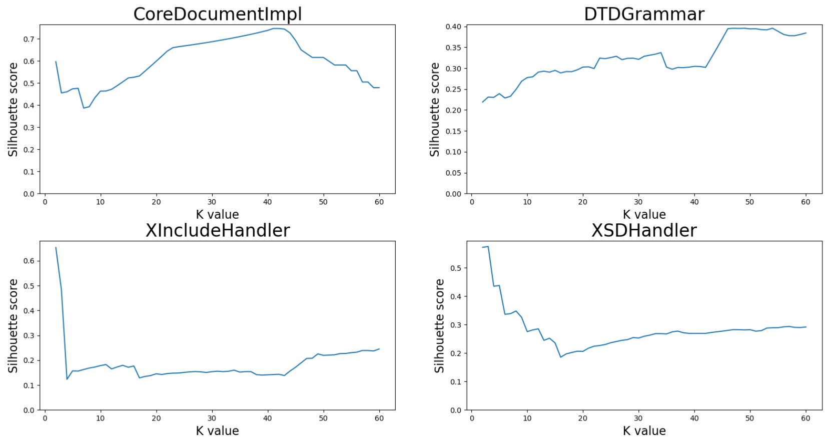

Figure 3: Silhouette scores of Agglomerative clustering for k=2,3, ..., 60

(2) As we can observe by the charts some classes like XIncludeHandler and XSDHandler have better silhouette scores for low k values, 2 or 3 depending on the algorithm. The other two classes have better scores at much higher k

values for both algorithms. We can also see that DTDGrammar could potentially reach a higher silhouette score for k values greater than 60 as the best k is at values close to 60, it would therefore be interesting to increase k values for that class and see what happens. Furthermore it seems that there is not a very significant difference between Agglomerative clustering and K means clustering as, for their best K, the silhouette scores are very close for every class. This can be easily observed in Table 3

*note: it should be noted that k-means algorithm has a randomic factor in its implementation as the initial choice of centroids is random. Nonetheless the differences that I got were very small, therefore the charts of silhouette score more or less always had the same shape and a similar best k.

Class Name	**Agglomerative**		**K-Means**	
	best k	Silhouette	best k	Silhouette
CoreDocumentImpl	41	0.75	43	0.74
DTDGrammar	54	0.4	56	0.39
XIncludeHandler	2	0.65	2	0.65
XSDHandler	3	0.58	2	0.56

Table 3: Best k Summary

3 Evaluation

3.1 Ground Truth

I computed the ground truth using the script ground-truth.py that takes a txt file storing the keywords and feature-vectors.csv file that is used just too loop through all the methods of our analysis. Therefore It does not operate on the features themselves. The output of the script are 4 csv files, one for each class, that contain the clustering solution that will be considered as ground truth. The generated files are then placed into the repository named ground-truth-csv-files.

I would say that this ground truth creation method is very intuitive and easy to use as it essentially groups methods with similar recurrent substrings in their names together. Therefore the main strength of it is that it is easy to implement and most likely leads to a good ground truth as methods with similar names most likely access similar fields and methods. The main weakness is that we have to take for granted this assumption, not always methods with similar names operate with the same fields/methods. Therefore a more precise approach would need a deeper analysis of the source code, but this has the con of being much harder and longer to do. Therefore this technique we've used for ground truth identification is easy to use and has good chance of being close to reality, the main cost is that, if you need to do a more precise analysis, you may need a different procedure. With that being said I think that, when it comes to refractoring, this kind of ground truth works well. In this domain doing very accurate splitting of the classes is probably not that important, if a method ends up in a class rather than another it probably won't create too many problems. The important thing is that, as a whole, the organization of the code makes sense and I think that this technique is valid to meet this objective. I'm saying this as Xerces2 is a library made by professionals that are very likely to give relevant names to the methods of a class, clearly this ground-truth approach would be very ineffective if developers gave random/non relevant names to their code. I would use a different approach if we needed to apply refractoring to code made by amateurs or students.

3.2 Precision and Recall

Class Name	**Agglomerative**		**K-Means**			
	Prec.	Recall	F score	Prec.	Recall	F score
CoreDocumentImpl	0.66	0.31	0.42	0.67	0.31	0.42
DTDGrammar	0.73	0.08	0.15	0.88	0.1	0.17
XIncludeHandler	0.69	0.93	0.8	0.7	0.95	0.8
XSDHandler	0.21	0.55	0.31	0.21	0.54	0.30

Table 4: Evaluation Summary

Precision and Recall, for the optimal configurations found in Section 2, are reported in Table 4.

3.3 Practical Usefulness

When developing large projects that consume a lot of resources like Xerces2 library optimizing the work of software engineers is an important task. Having organized code is crucial as it helps to save up time and reduce errors. Our analysis focused on god classes that, in simple terms, are classes that do too much. Generally those are not an ideal thing to have as it can lead to the inefficiencies above. The clustering algorithms implemented in this project and their subsequent evaluation shown that there are relevant refractoring opportunities in some of the God classes identified. Lowering the complexity of such classes can lead to a better management of resources as programmers can access and understand the contents of the now splitted god class much faster and quickly address bugs. The clustering algorithms used suggest that the first class that should be refractored is XIncludeHandler as our clustering solution, when compared against ground truth, produced a very good F score of 0.8. Other classes had lower F scores, we could try to apply different algorithms or change some configurations to see the F score gets higher. These results also depend on the ground truth itself, we could also test other methods for ground truth computation and see if the results change. Overally I think that this project is a good example of how unsupervised machine learning algorithms such as k means and agglomerative clustering can be used in a software engineering project.