Clustering

Marco Nobile

May 2019

1 Introduction

In this report I will present the results obtained using clustering to support the automated refactoring of God classes. In particular I'm going to try to find the so called 'God classes' in the Java library Xerces2, used for parsing, validating and manipulating XML documents.

2 Data pre-processing:

To begin with, we need to define how we are going to detect a God class, and the condition that we are going to apply is the following: C is a God class iff $|M(C) > \mu(M) + 6\sigma(M)|$, where $M(C)$ is the number of methods in class C and M is the set of all methods across all classes. In our case, for the Xerces2 library we have $\mu(M) = 10.407$ and $\sigma(M) = 14.636$
To obtain this information we need first to parse the files in the Xerces2 library, considering all the *ClassDeclaration* nodes contained in each all AST tree for all the classes (skipping the inner classes).
After doing so, and after applying a filter to detect the God classes as described above, we obtain that the classes: XSDHandler, DTDGrammar, XIncludeHandler, CoreDocumentImpl with respectively 118, 101, 116 and 125 methods, are God Classes.
After this initial step we can proceed with the *feature extraction* process. Since our aim is to cluster the elements of the God classes to obtain a valid (and possible) refactoring of the class, we follow the idea that cohesive groups of methods operate on the same class fields and invoke the same class methods. For this reason we try to collect our data such that we can understand for each class, which method and fields it contains (considering also all the global fields accessed by the methods and all the methods accessed by other methods).
Following this procedure we create 4 .csv file, one for each God class identified. The rows of this .csv are all the methods contained in the class considered, while the columns are all the fields and all the methods accessed by the methods in the column 'method name'. The entries of the .csv are boolean values, that identify the absence/presence of a method/field in the given method.
Gathering this information yields to 4 .csv of shape:

class name	n. rows	n. cols
CoreDocumentImpl	117	67
DTDGrammar	91	103
XIncludeHandler	108	179
XSDHandler	106	194

Table 1: Shapes of the 4 .csv for each class

3 Clustering

In this section I will report the results obtained applying 2 different techniques for clustering: k-Means and Hierarchical clustering.

3.1 k-Means

Using the function **KMeans** of the sklearn.cluster package we can obtain a clusterization of the data obtained by this point-assignment algorithm. The parameters that I choose to use are:

- n_clusters, discussed later
- init = 'k-means++', to speed up convergence
- algorithm = 'full', since our data is sparse for the boolean nature

As we can see, we need to pick a value for the number of clusters that we want to obtain. To find the *optimal* value of k there are different criteria that could be used, and in this project we are going to use the silhouette_score contained in sklearn.metrics. The Silhouette score is calculated using the mean intra-cluster distance and the mean nearest-cluster distance for each sample.

3.2 Hierarchical Clustering

Using the function **AgglomerativeClustering** contained in the sklearn.cluster package we can obtain a clusterization combining point (in the beginning) and cluster based on their "closeness". The parameter that we have to choose are:

- n_clusters, discussed later
- affinity, closeness measure: 'euclidean' or 'l1' or 'l2' or 'manhattan'
- linkage, which linkage criterion to use: 'complete', 'average' or 'ward'

Once again we have to pick the desired number of cluster, which states at which highness we are going to cut the dendrogram obtained with the algorithm. We have also to care about the fact that if we pick the linkage "ward" we can only use the "euclidean" distance. To find the *optimal* value of k we use Silhouette score as described above.

Now I'm going to present the results of this algorithms for each God class identified in section 1.

3.3 CoreDocumentImpl: k-Mean

Using a for loop I have evaluated all the possible k from k=2 up to k=81, and I have obtained that the best value of k is 45 for this class, with a silhouette value of 0.735. Furthermore we can also notice that the value of the silhouette score does not change for $k > 45$. More in particular we can inspect the number of elements contained in each cluster:

ClusterID	n. of elements	ClusterID	n. of elements
0	46	22	1
1	7	23	2
2	1	24	2
3	1	25	1
4	1	26	2
5	1	27	1
6	2	28	1
7	6	29	1
8	1	30	1
9	1	31	1
10	1	32	1
11	3	33	1
12	3	34	1
13	1	35	1
14	4	36	1
15	2	37	1
16	1	38	1
17	1	39	1
18	2	40	1
19	2	41	1
20	3	42	1
21	1	43	1
-	-	44	1

Table 2: Size of clusters for CoreDocumentImpl: k-Mean

We can also comment that that the larger is the number of cluster that we require, the lower is the average number of elements contained in each clusters. For this reason in the table above we can see that we have many cluster with a small number of elements.

3.4 CoreDocumentImpl: Hierarchical Clustering

Using the for loop as in the k-Mean clusterization, I have obtained that the parameters that yields to the maximum Silhouette score is the combination of euclidean distance, with the ward linakge, with 43 clusters, which allows us to obtain a Silhouette value of 0.737. We can also notice that the maximum number is cluster is 47. Once again I'm going to present the number of elements contained in each cluster:

ClusterID	n. of elements	ClusterID	n. of elements
0	2	22	2
1	2	23	1
2	2	24	2
3	3	25	1
4	6	26	1
5	2	27	1
6	3	28	1
7	1	29	1
8	46	30	1
9	2	31	1
10	2	32	1
11	7	33	1
12	1	34	1
13	1	35	1
14	4	36	1
15	1	37	1
16	1	38	1
17	1	39	1
18	1	40	1
19	3	41	1
20	2	42	1
21	1	-	-

Table 3: Size of clusters for CoreDocumentImpl: Hierarchical Clustering

We can notice that the clustarization for this class, both with Hierarchical Clustering and KMeans, yields to 45 or more clusters with few elements. For both the methods we have a good silhouette value (since it ranges from -1 to 1).

3.5 DTDGrammar: k-Mean

For this class the best value of k for the k-Mean algorithm is 59 with a silhouette value of 0.43. We can also notice that the silhouette value for $k > 67$ does not change. As before I will show the number of elements contained in each cluster:

ClusterID	n. of elements	ClusterID	n. of elements
0	2	29	1
1	16	30	1
2	2	31	2
3	1	32	1
4	1	33	1
5	2	34	1
6	1	35	1
7	1	36	1
8	2	37	2
9	1	38	1
10	2	39	1
11	3	40	1
12	1	41	2
13	1	42	1
14	1	43	1
15	2	44	1
16	2	45	1
17	1	46	1
18	1	47	2
19	1	48	1
20	2	49	1
21	1	50	1
22	1	51	2
23	1	52	1
24	2	53	1
25	2	54	1
26	1	55	1
27	1	56	1
28	1	57	1
-	-	58	1

Table 4: Size of clusters for DTDGrammar: KMeans

Once again, given the high number of clusters required, we have many cluster with 1 or 2 elements.

3.6 DTDGrammar: Hierarchical clustering

For this class the best value of k for the Hierarchical clustering is 57, euclidean distance and ward linkage. The maximum number of cluster is 67. The silhouette value obtained is 0.454. As before I will show the number of elements contained in each cluster:

ClusterID	n. of elements	ClusterID	n. of elements
0	3	29	1
1	2	30	1
2	2	31	1
3	2	32	1
4	2	33	1
5	2	34	2
6	2	35	1
7	2	36	1
8	3	37	1
9	2	38	2
10	1	39	1
11	2	40	1
12	1	41	1
13	1	42	1
14	1	43	1
15	16	44	1
16	2	45	1
17	2	46	1
18	2	47	1
19	2	48	1
20	1	49	1
21	1	50	1
22	1	51	1
23	1	52	1
24	1	53	1
25	1	54	1
26	1	55	1
27	1	56	1
28	1	-	-

Table 5: Size of clusters for DTDGrammar: Hierarchical clustering

3.7 XIncludeHandler: k-Mean

For this class I have obtained that the best value of k is 2 , with a silhouette value of 0.671. As before I will show the number of elements contained in each cluster:

ClusterID	n. of elements
0	106
1	2

Table 6: Size of clusters for XIncludeHandler: KMeans

3.8 XIncludeHandler: Hierarchical clustering

For this class I have obtained that the best k is 2, using the euclidean distance and the average linkage, the relative silhouette value is 0.684. As before I will show the number of elements contained in each cluster:

ClusterID	n. of elements
0	107
1	1

Table 7: Size of clusters for XIncludeHandler: Hierarchical clustering

3.9 XSDHandler: k-Mean

For this class I have obtained that the best value of k is 2 , with a silhouette value of 0.507. As before I will show the number of elements contained in each cluster:

ClusterID	n. of elements
0	5
1	101

Table 8: Size of clusters for XSDHandler: KMeans

3.10 XSDHandler: Hierarchical clustering

For this class I have obtained that the best k is 2, using the euclidean distance and the average linkage, the relative silhouette value is 0.571. We can also notice that we cannot have 80 clusters. As before I will show the number of elements contained in each cluster:

ClusterID	n. of elements
0	105
1	1

Table 9: Size of clusters for XSDHandler: Hierarchical clustering

As we can see for the last 2 classes analyzed we have a bad clusterization because data is split such that we have a big cluster on one side, and on the other side a single cluster with only one element (not useful result in our case because it would mean to refactor a God Class in 2 classes, the first with many methods, and the other with only a single method).

4 Summary for silhouette score:

Let's recap what we have discovered up till now:

4.1 For k-Means:

Class name	n. clusters	silhouette
CoreDocumentImpl	45	0.735
DTDGrammar	59	0.430
XIncludeHandler	2	0.671
XSDHandler	2	0.465

Table 10: Silhouette value for KMeans across all classes

As we can see in the table above, the highest silhouette is achieved by the clusterization of CoreDocumentImpl. Given that the value of the silhouette ranges in between +1 and -1 we have obtained quite positive results . for CoreDocumentImpl.

4.2 For Hierarchical clustering:

Class name	n. clusters	silhouette	distance	linkage
CoreDocumentImpl	43	0.737	euclidean	ward
DTDGrammar	57	0.454	euclidean	ward
XIncludeHandler	2	0.684	euclidean	average
XSDHandler	2	0.571	euclidean	average

Table 11: Silhouette value and parameters for Hierarchical clustering across all classes

As we can see in the table above, the highest silhouette is achieved by the culusterization of CoreDocumentImpl. Given that the value of the silhouette ranges in between +1 and -1 we have obtained quite positive results only in this case. Furthermore we can also notice that the results of the hierarchical clustering are overall better if compared to the k-Mean results.

5 Evaluation

Let's now try a final evaluation of the clustering using precision, recall and the f1 measure, obtained comparing the results of the clusterization with the ground truth:

Class name	method	precision	recall	f1
CoreDocumentImpl	k-Means (k=45)	0.675	0.296	0.411
CoreDocumentImpl	HieClu (k=43)	0.676	0.297	0.412
DTDGrammar	k-Means (k=57)	0.888	0.077	0.142
DTDGrammar	HieClu (k=57)	0.858	0.056	0.105
XIncludeHandler	k-Means (k=2)	0.696	0.956	0.806
XIncludeHandler	HieClu (k=2)	0.698	0.978	0.815
XSDHandler	k-Means (k=2)	0.362	0.972	0.527
XSDHandler	HieClu (k=2)	0.362	0.972	0.527

Table 12: Precision, recall and f1 measure across all classes

As we can see the results for the hierarchical clustering are slightly better across all the classes considered, this is also confirmed by the evaluation of the silhuette values in the previous section.
Given that the precision is calculated as the fraction of pairs correctly placed in the same cluster, we can see that we have a good precision for the class CoreDocumentImpl, DTDGrammar and XIncludeHandler, while for XSDHandler our result is not really good from the precision perspective.
The recall is the fraction of actual pairs that were identified, and we can see that the best results are obtained on the classes XIncludeHandler and XSDHandler, while we have a low recall for CoreDocumentImpl and DTDGrammar.
Finally the F-measure is just the harmonic mean of precision and recall, it ranges between 0 and 1, thus we can say that the most balanced result according to this measure is obtained for XIncludeHandler.

6 PCA and Clustering visualization

To obtain a visualization of the cluster I've decided to reduce the dimensionality of the data using PCA, and using only the first two dimensions obtained I have been able to visualize the different clusterization. I have decided to present the results only for low values of k s.t. we can have a cleaner view:

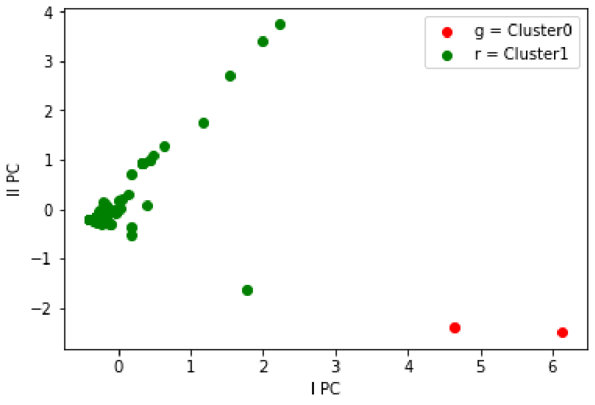

Figure 1: XIncludeHandler k-Means for k = 2

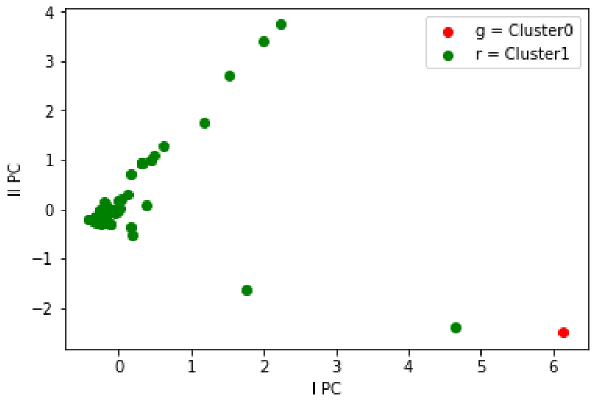

Figure 2: XIncludeHandler HieClu for k = 2

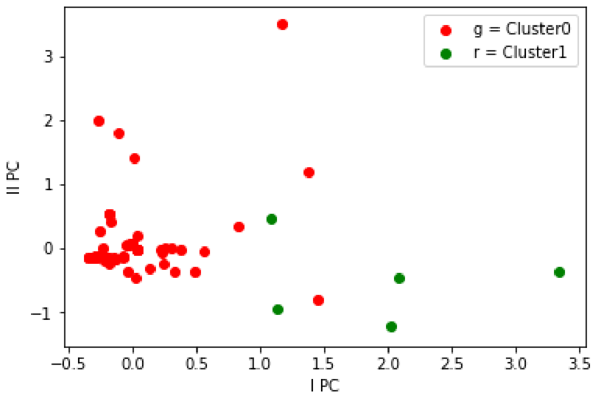

Figure 3: XSDHandler k-Means for k = 2

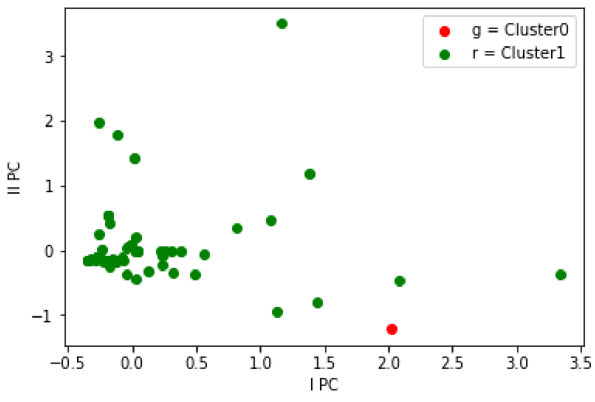

Figure 4: XSDHandler HieClu for k = 2

I would like to underline that in Figure 3, we are viewing the clusterization of the data obtained with the KMeans algorithm in the high dimensional space of the data, and therefore it makes sense that in the space defined by the first two pca components, the points in the same cluster are not close to each other.

7 Conclusions

The aim of the project was to analyze the options of automated clustering to support God class refactoring. As shown in the results, this process of clustering can yield to good performances (as for the class CoreDocumentImpl), but generally speaking the clusterization obtained on this dataset (in particular for low values of k) seem no useful to be a solid base for a refactorization process of the classes. What we could use from these result is the fact that for CoreDocumentImpl and for DTDGrammar we can see that we have respectively a cluster with 46 elements for the first one and 16 elements for the second one, this could be actually useful to proceed with a first split of the classes, which would slim down the their size. For further improvements of the clustering we could try to cluster the data in a lower dimensional space (e.g. applying PCA pre-clusterization) or we could try to find new measurement/features to describe the methods.