# Project: God Classes

**Iva Ilcheva**

## Contents

Part I

# Data pre-processing

## 1 God classes and feature vectors

### 1.1 Code

#### 1.1.1 Indentify god classes

1. **get_java_files()** $\longrightarrow$ Retrieves all *.java files.
2. **parse_files()** $\longrightarrow$ Opens each file, filters the tree for ClassDeclaration, records the class names and the number of methods in them in the same order.
3. **compute_mean_and_std()** $\longrightarrow$ Computes mean and std over the number of methods.
4. A dictionary of the class names and number of methods corresponding by index are both added to a Dataframe.
5. Said Dataframe is traversed with the God class condition and the god classes are extracted in a separate new Dataframe.
6. **get_god_classes_paths()** $\longrightarrow$ The whole list of files is traversed again and the classes' paths that match the found god classes are appended to an array to be used in the **extract-feature-vectors.py** script.

#### 1.1.2 Extract feature vectors

1. For each god class path from the array produced in the previous script:
2. **get_fields()** $\longrightarrow$ Retrieves class' attributes names.
3. **get_methods()** $\longrightarrow$ Retrieves class' methods names.
4. **get_fields_accessed_by_method()** $\longrightarrow$ Retrieves accessed class' attributes names in class' methods.
5. **get_methods_accessed_by_method()** $\longrightarrow$ Retrieves invoked class' methods names by class' methods.
6. A Dataframe for the feature vectors, consisting of **columns**: *'method_name'+fields+methods* and **rows:** *methods* is created empty.
7. **get_methods()** $\longrightarrow$ Returns also a dictionary of the method name, an array of its invoked methods and its accessed fields.

8. All the method_infos are traversed and for each field in the whole list of columns, if that same field is in the found_columns (meaning the accessed attributes and invoked methods in that particular method_info) 1 is appended to the feature vector for that row-column, otherwise - 0.

9. A Dataframe for that vector row is created and appended to the general feature vector Dataframe.

10. In the end, before moving to the next path a feature vector csv file is generated for the god class.

## 1.2 Data

### 1.2.1 Indentified god classes

Table 1: Indentified god classes

| **Class name** | **Number of methods** |
|---|---|
| XSDHandler | 118 |
| DTDGrammar | 101 |
| XIncludeHandler | 116 |
| CoreDocumentImpl | 125 |

After identifying the god classes based on the provided condition:

$$\text{God}(C) \Longleftrightarrow |M(C)| > \mu(M) + 6\sigma(M)$$

I filtered a total of **4** of them and the corresponding number of methods for each one can be found in the table above. Each of them contains more than 100 methods respectively, making them stand out from the others.

### 1.2.2 Extracted feature vectors

Table 2: Extracted feature vectors

| **Class name** | **Feature vectors** | **Fields** |
|---|---|---|
| XSDHandler | 106 | 120 |
| DTDGrammar | 91 | 75 |
| XIncludeHandler | 92 | 108 |
| CoreDocumentImpl | 23 | 117 |

The number of feature vectors and attributes of each god class can be seen in the table above, where the number of feature vectors also corresponds to the number of methods in the class not in the whole ***.java** file. Except for **DTDGrammar** all other ones have more feature vectors than actual number of fields. It could be a class that is designated for performing actions upon data, rather than predominantly storing it. The difference between its two columns, however, is too negligible to make assumptions. **CoreDocumentImpl**, on the other hand, seems to serve exactly as variable storer with its **117** fields and only **23** methods. In order to be completely certain if this is the case, we would have to examine carefully each feature vectors' file and count the number of 1 occurences for the accessed fields and invoked methods separately.

# Part II
# Clustering

## 2 Algorithm configurations

The unmentioned parameters are left as default.

### 2.1 KMeans

The algorithm is set to **auto**, depending on the density of the data it chooses either **Elkan - Triangle inequality** for dense or the classical **EM-style** for sparse data. The number of **n_clusters** is provided manually and is between 2 and 80.

### 2.2 Hierarchical

The metric used to compute the linkage is **Euclidean**. The linkage rule is **ward** which minimizes the variance of the clusters being merged. The number of **n_clusters** is provided manually and is between 2 and 80.

## 3 Clusters

I decided to document on **min** $\rightarrow$ k = 2, **mean** $\rightarrow$ k = 41 and **max** $\rightarrow$ k = 80 (or the value at which the number of distinct clusters is reached and a convergence warning is produced).

### 3.1 k = 2

In the tables will be shown for **Cluster 0** and **Cluster 1** the number of belonging methods. God class corresponds to the feature vector of the god class that the clustering was performed upon. The label of each table introduces which algorithm is performed.

Table 3: KMeans

| **God class** | **C0** | **C1** |
|---|---|---|
| XSDHandler | 3 | 103 |
| DTDGrammar | 62 | 29 |
| XIncludeHandler | 3 | 105 |
| CoreDocumentImpl | 99 | 18 |

Table 4: Hierarchical

| **God class** | **C0** | **C1** |
|---|---|---|
| XSDHandler | 3 | 103 |
| DTDGrammar | 35 | 56 |
| XIncludeHandler | 106 | 2 |
| CoreDocumentImpl | 114 | 3 |

Both algorithms perform poorly on clustering the god classes for this value of **k**. Only **DTDGrammar** is somewhat clustered meaningfully. All others' majorities are placed in one of the clusters.

## 3.2 k = 41

As the size of the tables would be too big for the observations I will comment on, let me redirect you to *./report−clusters/...−41.csv* files for all god classes and both algorithms (that contain them). #Clusters, god class and algorithm are stated in the file names.

For **KMeans** there are around 10-16 significant clusters with the number varying for different god classes, all others contain 1 element.

For **Hierarchical** there are around 15-22 significant clusters with the number varying for different god classes, all others contain 1 element.

Both of the algorithms perform slightly better with **Hierarchical** producing more heterogeneous clustering, but still at least half the elements belong to their own cluster. They could possibly be merged with a better picked **k**.

## 3.3 k = 80 or #distinct_clusters

As the size of the tables would be too big for the observations I will comment on, let me redirect you to *./report − clusters/... − #distinct_clusters.csv* files for all god classes and both algorithms (that contain them). #Clusters, god class and algorithm are stated in the file names.

For **KMeans** there are around 7-15 significant clusters with the number varying for different god classes, all others contain 1 element.

For **Hierarchical** there are around 1-15 significant clusters with the number varying for different god classes, all others contain 1 element.

Both of the algorithms perform worse than **k=41** and slightly better than **k=2** with **KMeans** producing more heterogeneous clustering, but still at least **1/6** of the elements belong to their own cluster. They definitely would be merged with a better picked **k**.

## 4 Silhouette

Table 5: #Clusters for max silhouette

| **God class** | **KMeans** | **Hierarchical** |
|---|---|---|
| XSDHandler | 42 | 41 |
| DTDGrammar | 60 | 58 |
| XIncludeHandler | 2 | 2 |
| CoreDocumentImpl | 2 | 3 |

For all clusterings for all god classes, I computed the silhouettes in order to find the **most optimal k** that maximizes the silhouette metric and produces the most heterogenious clustering. The numbers in the table correspond to the value of said **k** for the god class and algorithm on that row - column.

We can see that both **KMeans** and **Hierarchical** perform best in the same range of +/- 1-2 number of clusters. **XSDHandler** and **DTDGrammar** require a higher value of **k** to reach convergence whereas **XIncludeHandler** and **CoreDocumentImpl** require the bare minimun **k** to perform their best.

Part III

# Evaluation

## 5 Ground truth, precision and recall

### 5.1 Ground truth

Table 6: CoreDocumentImpl

| **Keyword_id** | **Size** |
|---|---|
| 14 | 75 |
| 3 | 14 |
| 4 | 6 |
| 7 | 5 |
| 11 | 4 |
| 5 | 4 |
| 13 | 3 |
| 9 | 2 |
| 2 | 2 |
| 0 | 2 |

DTDGrammar

| **Keyword_id** | **Size** |
|---|---|
| 14 | 64 |
| 2 | 18 |
| 3 | 5 |
| 12 | 2 |
| 10 | 2 |

XIncludeHandler

| **Keyword_id** | **Size** |
|---|---|
| 14 | 90 |
| 13 | 9 |
| 5 | 4 |
| 12 | 2 |
| 10 | 2 |
| 3 | 1 |

XSDHandler

| **Keyword_id** | **Size** |
|---|---|
| 14 | 57 |
| 6 | 22 |
| 1 | 20 |
| 5 | 3 |
| 3 | 3 |
| 2 | 1 |

The produced ground truth shows that with the picked **keywords** (a total of 15 of them) there is not at all heterogeneity in the clusterings as well. **14** actually corresponds to **none** keyword, meaning that as a whole the list doesn't really match well the naming of the methods. As observed with the algorithms as well we have a small amount of clusters that actually are worth taking into account and do not contain only 1 element, what is more the majority of elements belong to a particular one.

## 5.2 Precision and recall

Table 7: CoreDocumentImpl

| **Algorithm** | **Precision** | **Recall** |
|---|---|---|
| **KMeans** | 0.6775658492279746 | 0.2564455139223101 |
| **Hierarchical** | 0.677858439201452 | 0.25678927466483326 |

Table 8: DTDGrammar

| **Algorithm** | **Precision** | **Recall** |
|---|---|---|
| **KMeans** | 0.881578947368421 | 0.0614397065566254 |
| **Hierarchical** | 0.8774193548387097 | 0.062356717102246675 |

Table 9: XIncludeHandler

| **Algorithm** | **Precision** | **Recall** |
|---|---|---|
| **KMeans** | 0.6958318361480417 | 0.956532477154853 |
| **Hierarchical** | 0.6958318361480417 | 0.956532477154853 |

Table 10: XSDHandler

| **Algorithm** | **Precision** | **Recall** |
|---|---|---|
| **KMeans** | 0.36025641025641025 | 0.972318339100346 |
| **Hierarchical** | 0.3570042685292976 | 0.9095402867029164 |

In the tables above **precision** stands for the fraction of common intra pairs among the retrieved intra pairs by the clustering algorithm and the ground truth over the ones only from the clustering algorithm. **Recall** is the fraction of common intra pairs among the retrieved intra pairs by the clustering algorithm and the ground truth over correctly labeled ones from the ground truth. The algorithms were run with the **most optimal k** values that can be found in the **Silhouette** section. For **CoreDocumentImpl** and **DTDGrammar** the precision is quite high but the recall is poor. **XIncludeHandler** shows an outstanding high precision and recall values for the clustering to ground truth comparison. For **XSDHandler** has a very poor precision but quite high recall.

# 6 Usefulness of automated clustering to support God class refactoring

Let's imagine the following setup of having a team of developers before and after introducing automated clustering into their daily work. Performing automated clustering regularly while working on a big project with a lot of people would be beneficial. It could serve a vital role similar to unit or integration testing during the development process. It would be ideal to run such automated clustering at the end of each work day to detect ill classes that need less refactoring at that given moment in time due to being caught in the early stages of becoming a god class. Developers will easily refactor a smaller class than one of a few hundred lines. It might take a bit more time to run it everyday but saves a lot more in the future in terms of lines of code that need to be refactored and also a strategy of how to perform such a huge refactoring. I am glad I learnt about this problem and this possible way of preventing it. I'd definitely make use of this in my future endeavors.

## Part IV
# Appendix

Executing **sh run.sh** in the command line in the same directory runs all scripts sequentially in the correct order to produce the desired results. Each script produces a folder with the same name and the corresponding ***.csv** files.

Listing 1: run.sh

```
#!/bin/bash
echo 'Extracting the feature vectors...'
sh ./extract-feature-vectors.sh
echo 'Finished!'
echo 'Clustering god classes with kmeans and hierarchical algorithms...'
sh ./clustering.sh
echo 'Finished!'
echo 'Computing silhouette metric for k = {2 .. 80} for both algorithms...'
sh ./silhouettes.sh
echo 'Finished!'
echo 'Computing the best k for both algorithms individually...'
sh ./most-optimal-k.sh
echo 'Finished!'
echo 'Defining the ground truth...'
sh ./ground-truth.sh
echo 'Finished!'
echo 'Computing the precision and recall...'
sh ./prec-recall.sh
echo 'Finished'
```

- Indentify god classes: **find_god_classes.py**
- Extract feature vectors: **extract-feature-vectors.py**
- Perform clustering: **k-means.py** and **hierarchical.py**
- Compute precision and recall: **prec-recall.py**

**NB!: *xerces2-j-trunk* folder has to be in the same directory.**