Project: God classes

Information Modeling & Analysis

Paolo Tonella

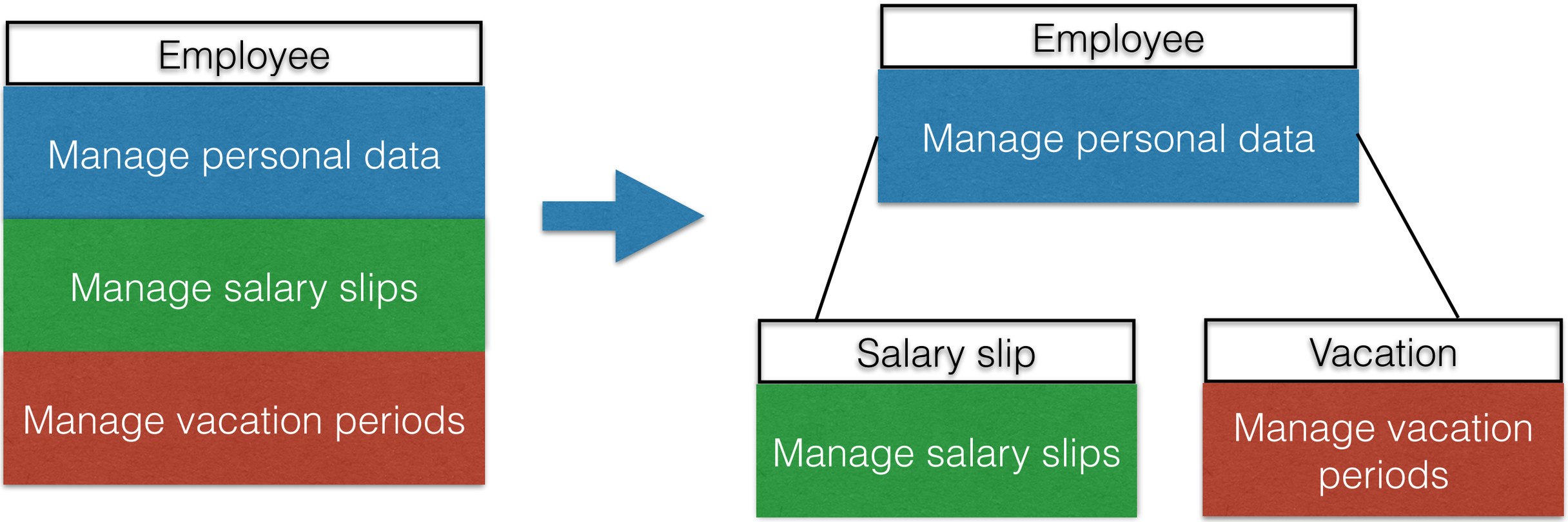
paolo.tonella@usi.ch

Goal of the project

Use clustering to support automated refactoring of God classes

A ***God class*** is a class that does too much:

- violates the "single responsibility" and "abstraction/encapsulation" OO design principles
- does not support reuse, because it does not implement a single, well defined functionality
- does not abstract from the domain, since it is usually entangled with multiple implementation details (hence it is also difficult to test)



Steps of the project

1. [data pre-processing] Identify God classes
2. [data pre-processing] Extract feature vectors
3. [clustering] Apply clustering algorithms to partition the God classes
4. [evaluation] Measure the quality of the God class partitions

Subject

Xerces2: Java library for parsing, validating and manipulating XML documents

Create your github classroom repository at:

- https://classroom.github.com/a/gqGgxsF3

You'll find the Xerces2 source code in the resources folder.

Note: You must create your repository through the link provided above. Please do not create any custom repositories yourself!

Step 1: Identify God classes

1. Parse the source code using the Python library `javalang`
   - https://github.com/c2nes/javalang
2. Identify God classes by the number of methods they contain, applying the following condition:

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

where M(C) is the set of methods in class C; M is the set of all methods across all classes

Step 1: Identify God classes

Hints:

- get all files in a directory by Python's walk:
  - `path, dirs, files in os.walk(inputPath)`
- populate a data frame with columns `'class_name', 'method_num'`
- visit the AST of each class by Python's
  - `for path, node in tree`
- recognize AST node type (e.g., class declaration) by `type(node)` (e.g., `is ClassDeclaration`)
- consult Javalang's file `tree.py` to know the AST node types and/or print available attributes/values at each AST node
  - parsing a small code snippet: `tree = javalang.parse.parse("class A { int f() {return 0;} }")`

https://github.com/c2nes/javalang

Project report

Section 1: Data pre-processing

- describe the code written to identify God classes and to extract feature vectors
- report/comment data on identified God classes (e.g., #classes, #methods)
- report/comment data on feature vectors extracted for God classes (e.g., #feature vectors, #attributes)

Section 2: Clustering

- report/comment the algorithm configurations (distance function, linkage rule, etc.)
- report/comment data about the clusters produced by the two algorithms at various k (#clusters, size of clusters)
- report/comment the results of Silhouette

Section 3: Evaluation

- report/comment ground truth, precision and recall
- discuss the usefulness of automated clustering to support God class refactoring

Project: God classes

Information Modeling & Analysis

Paolo Tonella

paolo.tonella@usi.ch

Project: God classes

Information Modeling & Analysis

Paolo Tonella

paolo.tonella@usi.ch

Steps of the project

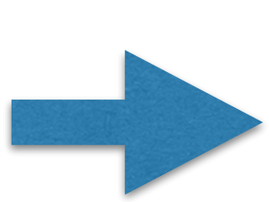
1. [data pre-processing] Identify God classes
2. [data pre-processing] Extract feature vectors
3. [clustering] Apply clustering algorithms to partition the God classes
4. [evaluation] Measure the quality of the God class partitions

Step 2: Extract feature vectors

Idea: cohesive groups of methods operate on the same class fields and invoke the same class methods.

1. Entities = methods (i.e., one feature vector per method)
2. Attributes = references to class fields, invocations of class methods

```
class A {
 int x, y;
 void f() {x = 0; y++;}
 void g() {y = 1; f();}
}
```



```
     x   y   f   g
f: <1,  1,  0,  0>
g: <0,  1,  1,  0>
```

column `g` can be omitted

Step 2: Extract feature vectors

Hints:

- write utility function to get all class fields/methods and all accessed fields/methods: `get_fields(java_class)`, `get_methods(java_class)`, `get_fields_accessed_by_method(method)` [look at `member` of `MemberReference` nodes], `get_methods_accessed_by_method(method)` [look at `MethodInvocation` nodes]
- if a `MemberReference` includes a non empty qualifier (e.g., `a.x`), consider the qualifier (`a`), not the member (`x`)
- populate a data frame with column 1 = `'method_name'` and following columns containing the feature vector attributes (accessed fields/methods)
- consider only the class matching the input file name, to skip inner classes
- in case of overloading, add methods only once to the data frame (e.g., `df['method_name'].isin([method.name]).any()`)
- if necessary, replace missing values with zeros (`df = df.fillna(0)`)
- if necessary, assign type `int` to attributes before generating CSV file (`df[attr] = df[attr].astype('int')`)
- generate CSV file by `df.to_csv(class_name + ".csv")`

Project report

Section 1: Data pre-processing

- describe the code written to identify God classes and to extract feature vectors
- report/comment data on identified God classes (e.g., #classes, #methods)
- report/comment data on feature vectors extracted for God classes (e.g., #feature vectors, #attributes)

Section 2: Clustering

- report/comment the algorithm configurations (distance function, linkage rule, etc.)
- report/comment data about the clusters produced by the two algorithms at various k (#clusters, size of clusters)
- report/comment the results of Silhouette

Section 3: Evaluation

- report/comment ground truth, precision and recall
- discuss the usefulness of automated clustering to support God class refactoring

Project: God classes

Information Modeling & Analysis

Paolo Tonella

paolo.tonella@usi.ch

Project: God classes

Information Modeling & Analysis

Paolo Tonella

paolo.tonella@usi.ch

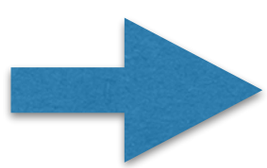
Steps of the project

1. [data pre-processing] Identify God classes
2. [data pre-processing] Extract feature vectors
3. [clustering] Apply clustering algorithms to partition the God classes
4. [evaluation] Measure the quality of the God class partitions

Step 3: Clustering

1. **Cluster God class methods by k-means algorithm**
2. **Cluster God class methods by hierarchical, agglomerative algorithm**
3. Compute silhouette metrics to chose the best k

```
class A {
 int x, y;
void f() {x = 0;}
void g() {y = 1;}
void h() {y++;}
}
```



feature vector
file.csv

k = 5

k_means.py
hierarchical.py

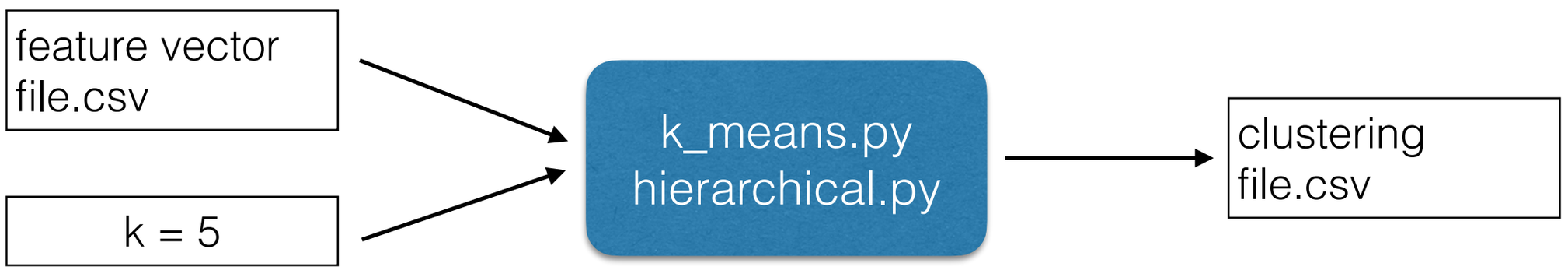
clustering
file.csv

```
cluster_id, method_name
0, f
1, g
1, h
```

Step 3: Clustering

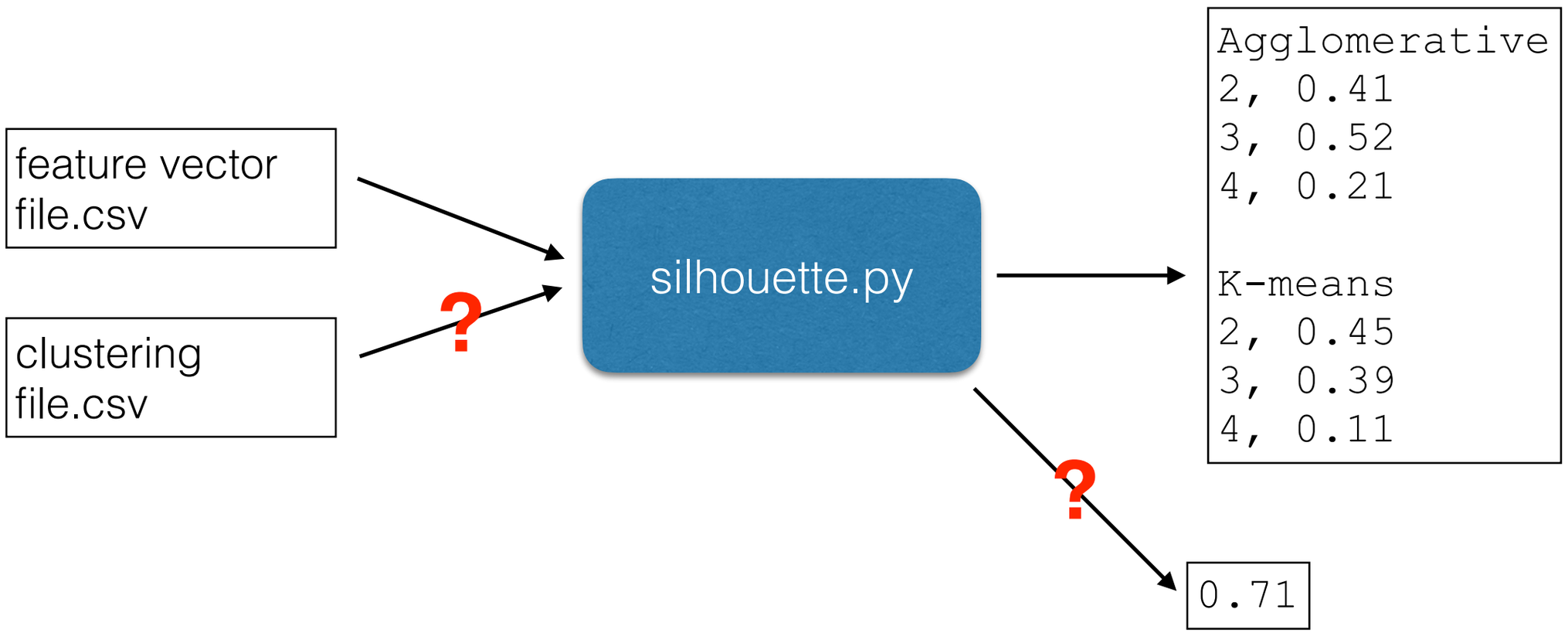
Hints:

- use `KMeans` and `AgglomerativeClustering` from module `sklearn.cluster` (with `n_clusters=k` as parameter)
- read from CSV file into a data frame
- convert data frame to array using `df.values`, after dropping column `method_name`
- for improved readability, when printing the result, group methods by cluster



Step 3: Silhouette

1. Cluster God class methods by k-means algorithm
2. Cluster God class methods by hierarchical, agglomerative algorithm
3. **Compute silhouette metrics to chose the best k**

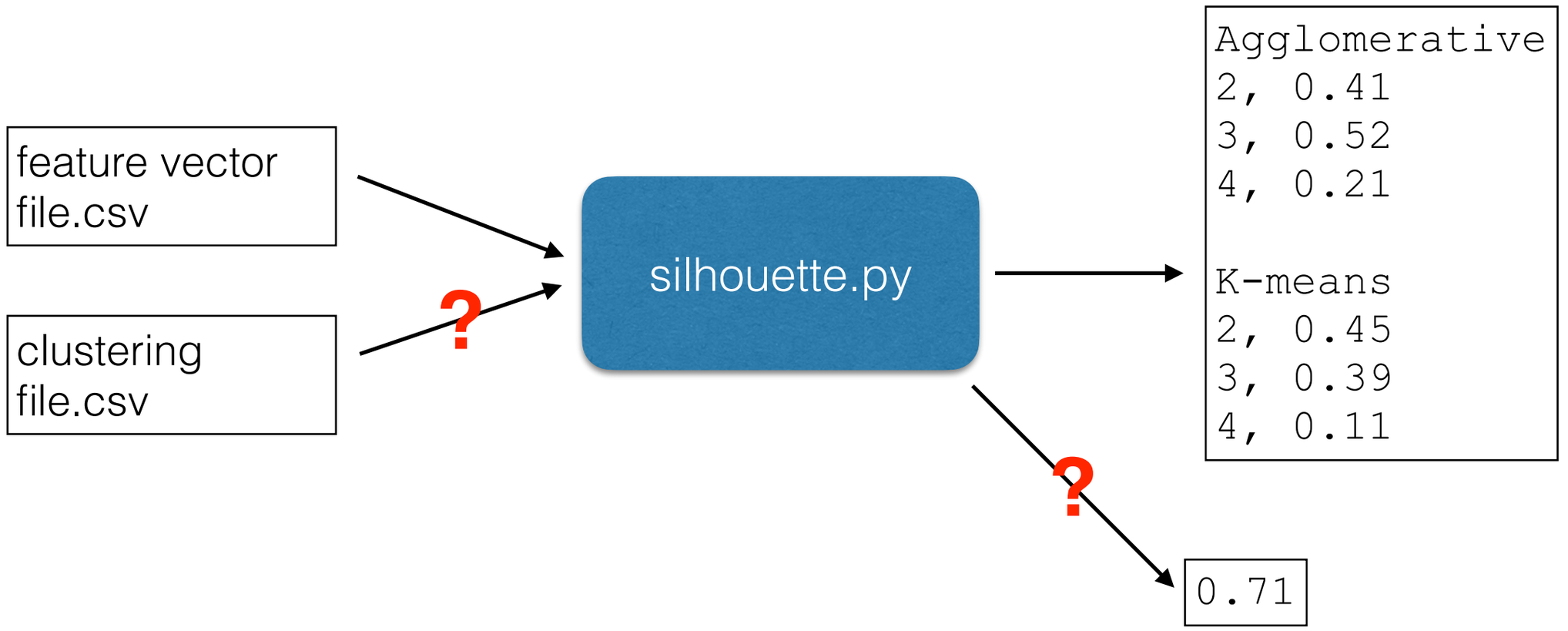


If no clustering file is provided, it computes the silhouette metrics for both algorithms with k ranging from 2 to MAX_k (e.g., 60); if a clustering file is provided, it computes the silhouette metrics for only such clustering

Step 3: Silhouette

Hints:

- use `silhouette_score` from module `sklearn.metrics`
- read from CSV files into data frames
- convert data frames to arrays using `df.values`, if necessary after dropping column `method_name`



Project report

Section 1: Data pre-processing

- describe the code written to identify God classes and to extract feature vectors
- report/comment data on identified God classes (e.g., #classes, #methods)
- report/comment data on feature vectors extracted for God classes (e.g., #feature vectors, #attributes)

Section 2: Clustering

- report/comment the algorithm configurations (distance function, linkage rule, etc.)
- report/comment data about the clusters produced by the two algorithms at various k (min/mean/max size of clusters)
- report/comment the results of Silhouette

Section 3: Evaluation

- report/comment ground truth, precision and recall
- discuss the usefulness of automated clustering to support God class refactoring

Project: God classes

Information Modeling & Analysis

Paolo Tonella

paolo.tonella@usi.ch

Project: God classes

Information Modeling & Analysis

Paolo Tonella

paolo.tonella@usi.ch

Steps of the project

1. [data pre-processing] Identify God classes
2. [data pre-processing] Extract feature vectors
3. [clustering] Apply clustering algorithms to partition the God classes
4. [evaluation] Measure the quality of the God class partitions

Step 4: Evaluation

1. Define the ground truth
2. Compute precision and recall

precision: $p = \frac{|\,\text{intrapairs}(D[k]) \cap \text{intrapairs}(G)\,|}{|\,\text{intrapairs}(D[k])\,|}$

recall: $r = \frac{|\,\text{intrapairs}(D[k]) \cap \text{intrapairs}(G)\,|}{|\,\text{intrapairs}(G)\,|}$

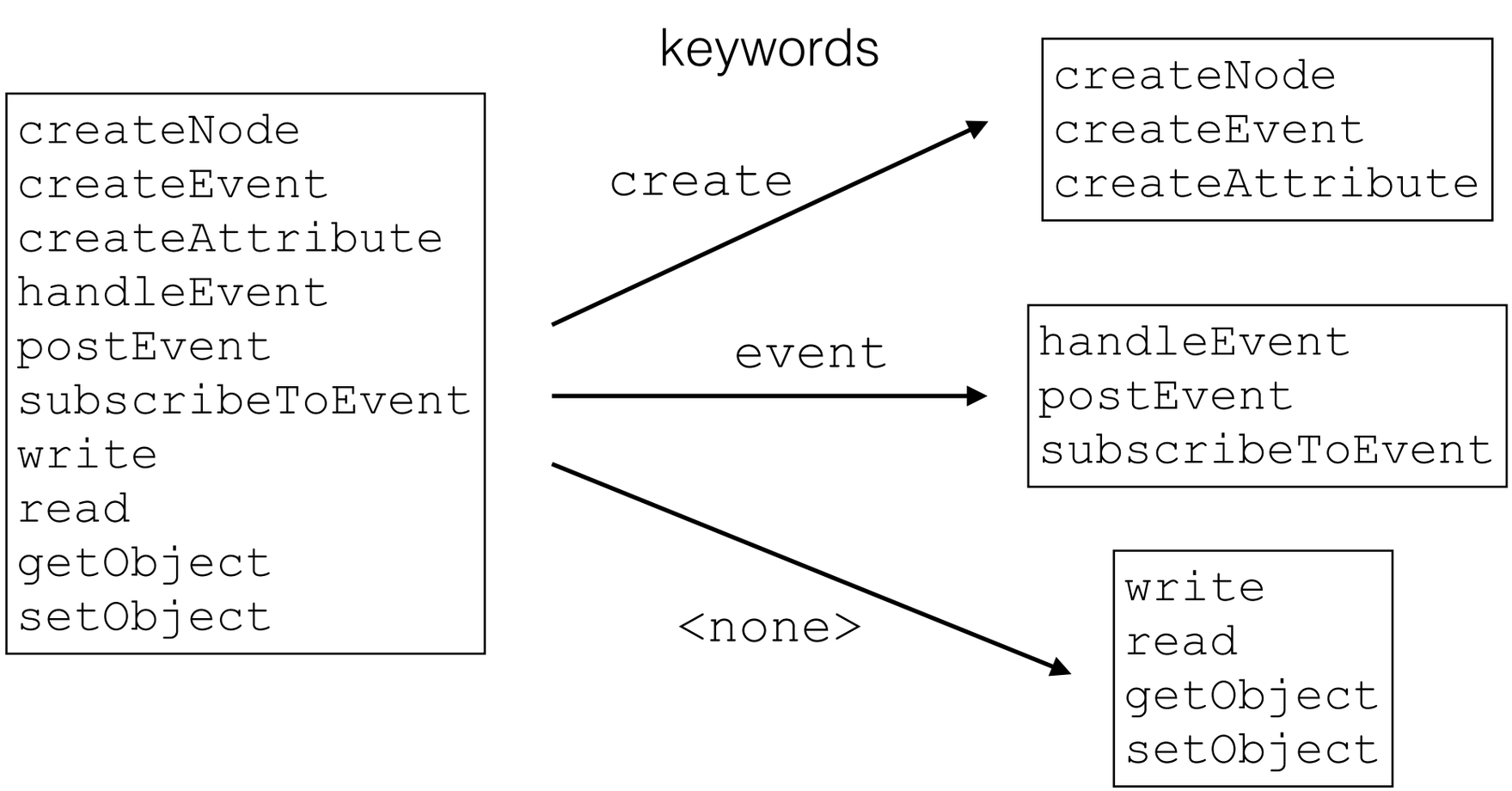
$$F_1 = \frac{2pr}{p+r}$$

$$\text{intrapairs}(\{C_1, \ldots, C_k\}) = \{\langle x, y\rangle \,|\, \exists i : x \in C_i, y \in C_i, x \neq y\}$$

Step 4: Evaluation

1. **Define the ground truth**
2. Compute precision and recall

Since we are not developers of Xerces, it might be difficult for us the define the ground truth manually. So, we approximate it by checking the presence of keywords in method names (as substrings).



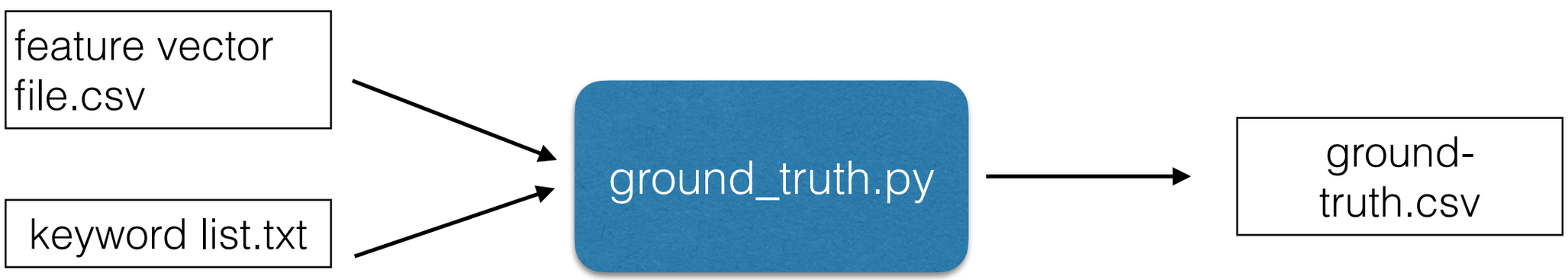
Step 4: Evaluation

1. **Define the ground truth**
2. Compute precision and recall

keywords (ordered)

create	error
object	content
cache	parameter
uri	subset
standalone	global
encoding	component
identifier	
user	

When multiple keywords match a method name, the first match in this ordered keyword lists is taken



Step 4: Evaluation

1. Define the ground truth
2. **Compute precision and recall**

```
createNode
createEvent
createAttribute
```

```
handleEvent
postEvent
subscribeToEvent
```

Intra-pairs

```
<createNode, createEvent>
<createNode, createAttribute>
<createEvent, createAttribute>
<createEvent, createNode>
<createAttribute, createNode>
<createAttribute, createEvent>
<handleEvent, postEvent>
<handleEvent, subscribeToEvent>
<postEvent, subscribeToEvent>
<postEvent, handleEvent>
<subscribeToEvent, handleEvent>
<subscribeToEvent, postEvent>
```

Hints:

- use Python's `set` data type, which provides set operations (`union`, `intersection`), on intra-pairs to compute precision and recall

Project report

Section 1: Data pre-processing

- describe the code written to identify God classes and to extract feature vectors
- report/comment data on identified God classes (e.g., #classes, #methods)
- report/comment data on feature vectors extracted for God classes (e.g., #feature vectors, #attributes)

Section 2: Clustering

- report/comment the algorithm configurations (distance function, linkage rule, etc.)
- report/comment data about the clusters produced by the two algorithms at various k (#clusters, size of clusters)
- report/comment the results of Silhouette

Section 3: Evaluation

- report/comment ground truth, precision and recall
- discuss the usefulness of automated clustering to support God class refactoring

Project: God classes

Information Modeling & Analysis

Paolo Tonella

paolo.tonella@usi.ch