The measures package

The external measure packages

The adjustedRand package

In this section, we provide a detailed description of the InputARand.java and AdjustedRand.java classes.

The InputARand.java class

It can be instantiated by one of the following constructors:

```java
public InputARand()

public InputARand(int kMin, int kMax, String algorithmPath, String algCommandLine, String outputPath, String algInputPath, String algOutputPath)
```  

```java
throws NotValidOptionException, FileAlgNotFoundException, InvalidCmndLineException, DirNotFoundException

public InputARand(int kMin, int kMax, String algorithmPath, String algCommandLine, String outputPath, String algInputPath, String algOutputPath, boolean initExtFlag, String initExtAlgPath, String initExtCommandLine, String initExtInpPath, String initExtOutPath)
```  

```java
throws NotValidOptionException, FileAlgNotFoundException, InvalidCmndLineException, DirNotFoundException
```

where the input parameters have been detailed in Additional File 1. The first is the empty constructor. In the last two, the user must provide all the input needed to compute the AdjustedRand.java class,
including the clustering algorithm used for the computation of the measure. In particular, the last constructor defines the input when the clustering algorithm is used with an external initialization.

**The AdjustedRand.java class**

It allows to compute the Adjusted Rand Index in two modes: single and iterative. In single mode, the agreement is computed between two given partitions of \( D \) stored both in the clusterMatrix data format. In the iterative mode, the agreement is computed between a given partition (e.g. the gold solution) and a set of partitions of \( D \) produced by a clustering algorithm, for all \( k \) in \( [k_{\text{min}}, k_{\text{max}}] \), i.e., \( C_{k_{\text{min}}}, \ldots, C_{k_{\text{max}}} \). In order to execute in single mode, the user must instantiate the class with the empty constructor:

```java
public AdjustedRand()
```

and compute the measure with the following method:

```java
public double computeAdjustedRand(ClusterMatrix trueSolution,
                                  ClusterMatrix clusterSolution,
                                  int nOfItem)
```

For the iterative mode execution, the user must define the input via the following constructor:

```java
public AdjustedRand(DataMatrix dataMatrix,
                     InputARand mParameters,
                     ClusterMatrix trueSolution)
```

Then, one can compute the measure with the following method:

```java
public MeasureVector computeMeasure()
throws FileNotFoundException,
         IOException, NumberFormatException, Exception
```

with the output of the method being a MeasureVector where all the \( k_{\text{max}} - k_{\text{min}} + 1 \) measure values are stored.

**The F-Index**

**The method**

The F-index [39] combines notions from information retrieval, such as precision and recall, in order to evaluate the agreement of a clustering solution \( C \) with respect to a reference partition \( \overline{C} \). Given \( \overline{c}_i \) and \( c_j \), their relative precision is defined as the ratio of the number of elements of class \( \overline{c}_i \) within cluster \( c_j \), divided by the size of cluster \( c_j \). That is:

\[
\text{Prec}(\overline{c}_i, c_j) = \frac{n_{i,j}}{n_j}
\]  

(1)
Moreover, their relative recall is defined as the ratio of the number of elements of class \( \tau_i \) within cluster \( c_j \), divided by the size of class \( \tau_i \). That is:

\[
Rec(\tau_i, c_j) = \frac{n_{i,j}}{n_i}
\]  

(2)

The F-index is then defined as an harmonic mean that uses the precision and recall values, with weight \( b \):

\[
F(\tau_i, c_j) = \frac{(b^2 + 1) \cdot Prec(\tau_i, c_j) \cdot Rec(\tau_i, c_j)}{b^2 \cdot Prec(\tau_i, c_j) + Rec(\tau_i, c_j)}
\]  

(3)

Equal weighting for precision and recall is obtained by setting \( b = 1 \). Finally, the overall F-index is:

\[
F = \sum_{\tau_i \in C} \frac{n_i}{n} \cdot \max_{c_k \in C} F(\tau_i, c_k)
\]  

(4)

\( F \) is a measure with value in the range \([0, 1]\), and the closer the measure is to one, the better the agreement between the two partitions is.

The findex package

In analogy with the adjustedRand package, it contains two main classes: the InputFIndex.java and the FIndex.java classes.

The InputFIndex.java class

In analogy with the InputARand.java class, it can be instantiated by one of the following two constructors:

public InputFIndex()  
public InputFIndex(int kMin, int kMax,  
String algorithmPath, String algCommandLine,  
String outputPath, String algInputPath, String algOutputPath)  
throws NotValidOptionException, FileAlgNotFoundException,  
InvalidCmdLineException, DirNotFoundException  
  
public InputFIndex(int kMin, int kMax,  
String algorithmPath, String algCommandLine,  
String outputPath, String algInputPath,  
String algOutputPath, boolean initExtFlag,  
String initExtAlgPath, String initExtCommandLine,  
String initExtInputPath, String initExtOutputPath)  
throws NotValidOptionException, FileAlgNotFoundException,  
InvalidCmdLineException, DirNotFoundException  

3
**The FIndex.java class**

In analogy with the AdjustedRand.java class, it can be instantiated by one of the following two constructors:

```java
public FIndex()
public FIndex(DataMatrix dataMatrix, InputMeasure mParameters, ClusterMatrix trueSolution)
```

it can be computed via either computeFMeasure or computeMeasure, depending on which constructor is used (i.e. either in single or iterative mode).

**The FM-Index**

*The method*

The FM-index [40] is defined as follows:

\[
FM_k = \frac{T_k}{\sqrt{U_k \cdot V_k}}
\]  

(5)

where:

\[
T_k = \sum_{i=1}^{k} \sum_{j=1}^{k} n_{ij}^2 - n
\]  

(6)

\[
U_k = \sum_{i=1}^{k} n_{i.}^2 - n
\]  

(7)

\[
V_k = \sum_{j=1}^{k} n_{.j}^2 - n
\]  

(8)

The measure has values in the range \([0, 1]\), and the closer the measure is to one, the better the agreement between the two partitions is.

**The fmindex package**

In analogy with the adjustedRand package, it contains two main classes: the InputFMIndex.java and the FMIndex.java classes.

**The InputFMIndex.java class**

In analogy with the InputARand.java class, it can be instantiated by one of the following two constructors:
public InputFMIndex()
public InputFMIndex(int kMin, int kMax, String algorithmPath, String algCommandLine, String outputPath, String algInputPath, String algOutputPath) throws NotValidOptionException, FileAlgNotFoundException, InvalidCmndLineException, DirNotFoundException

public InputFMIndex(int kMin, int kMax, String algorithmPath, String algCommandLine, String outputPath, String algInputPath, String algOutputPath, boolean initExtFlag, String initExtAlgPath, String initExtCommandLine, String initExtInpPath, String initExtOutPath) throws NotValidOptionException, FileAlgNotFoundException, InvalidCmndLineException, DirNotFoundException

The FMIndex.java class

In analogy with the AdjustedRand.java, it can be instantiated by one of the following two constructors:

public FMIndex()
public FMIndex(DataMatrix dataMatrix, InputMeasure mParameters, ClusterMatrix trueSolution)

and it can be computed via either computeFMIndex or computeMeasure, depending on which constructor is used (i.e. either in single or iterative mode).

The nullmeasures package

InputNullM.java class It can be instantiated by one of the following five constructors:

public InputNullM()

public InputNullM(int kMin, int kMax, String algorithmPath, String algCommandLine, String outputPath, String algInputPath, String algOutputPath) throws NotValidOptionException, FileAlgNotFoundException, InvalidCmndLineException, DirNotFoundException

public InputNullM(int kMin, int kMax, String algorithmPath, String algCommandLine, String outputPath, String algInputPath, String algOutputPath, boolean initExtFlag, String initExtAlgPath, String initExtCommandLine, String initExtInpPath, String initExtOutPath) throws NotValidOptionException, FileAlgNotFoundException,
InvalidCmndLineException, DirNotFoundException

The following two constructors provide the input for the HLink.java class:

```java
public InputNullM(int kMin, int kMax,
        String outputPath, String algInputPath,
        String algOutputPath, String hAlgPath,
        String type, String outputHierarchical)
throws NotValidOptionException, FileAlgNotFoundException,
        InvalidCmndLineException, DirNotFoundException

public InputNullM(int kMin, int kMax,
        String algorithmPath, String algCommandLine,
        String outputPath, String algInputPath,
        String algOutputPath, boolean predictNOfCluster,
        String hAlgPath, String type,
        String outputHierarchical, boolean adjustmentFactor)
throws NotValidOptionException, FileAlgNotFoundException,
        InvalidCmndLineException, DirNotFoundException
```

With reference to the parameters detailed in Additional File 1, the following two additional input parameters can be used, depending on the method:

- hAlgPath: the path of the binary executable of the hierarchical clustering algorithm;
- type: the type of hierarchical clustering algorithm: it must assume value “a”, “c” or “s”, indicating Average, Complete and Single, respectively.

The NullMeasure.java class It is an abstract class that defines all public methods common to all other null measure classes. They are:

- NullMeasureGeneric.java
- NullMeasureHierarchical.java
- NullMeasureHierarchicalInit.java

each of the previous classes provides two class constructors, i.e., the empty constructor and a constructor that takes as input a DataMatrix (see Additional File 1) and the InputMeasure (see Additional File 1). For brevity, we limit ourselves to show only the first class:

```java
public NullMeasureGeneric()
public NullMeasureGeneric(DataMatrix dataMatrix, InputMeasure mParameters)
```

The measure is computed via the following method:
public MeasureVector computeMeasure() throws FileNotFoundException, IOException, NumberFormatException, Exception

The internal measure package
The WCSS package
The InputWCSS.java class

In analogy with the InputNullM.java class, it can be instantiated by one of the following constructors:

public InputWCSS()

public InputWCSS(int kMin, int kMax, String algorithmPath, String algCommandLine, String outputPath, String algInputPath, String algOutputPath, boolean predictNOfCluster) throws NotValidOptionException, FileAlgNotFoundException, InvalidCmndLineException, DirNotFoundException

public InputWCSS(int kMin, int kMax, String algorithmPath, String algCommandLine, String outputPath, String algInputPath, String algOutputPath, boolean predictNOfCluster, boolean initExtFlag, String initExtAlgPath, String initExtCommandLine, String initExtInpPath, String initExtOutPath) throws NotValidOptionException, FileAlgNotFoundException, InvalidCmndLineException, DirNotFoundException

public InputWCSS(int kMin, int kMax, String algorithmPath, String algCommandLine, String outputPath, String algInputPath, String algOutputPath, boolean predictNOfCluster, int numberOfSteps, int numberOfIteration, int percentage) throws NotValidOptionException, FileAlgNotFoundException, InvalidCmndLineException, DirNotFoundException

public InputWCSS(int kMin, int kMax, String algorithmPath, String algCommandLine, String outputPath, String algInputPath, String algOutputPath, boolean predictNOfCluster, String hAlgPath, String type, String outputHierarchical) throws NotValidOptionException, FileAlgNotFoundException, InvalidCmndLineException, DirNotFoundException

public InputWCSS(int kMin, int kMax, String algorithmPath, String algCommandLine, String outputPath, String algInputPath, String algOutputPath, boolean predictNOfCluster, String hAlgPath, String type, String outputHierarchical)
throws NotValidOptionException, FileAlgNotFoundException, 
InvalidCmndLineException, DirNotFoundExcep

The first is the empty constructor (therefore the user must provide the relevant input via the appropriate 
setter\(^1\) methods). The second defines all the parameters of a generic clustering algorithm. The third allows 
to use an external initialization for the selected clustering algorithm. The fourth allows to set the input for 
the approximation of WCSS presented in [20]. The fifth defines all the inputs to compute WCSS in 
conjunction with the HLink.java class. The last constructor defines the input to compute WCSS when the 
HLink.java class is used as an external initialization algorithm.

With reference to the parameters detailed in Additional File 1, the following four additional input 
parameters can be used, depending on the method:

- predictNOfCluster: a boolean flag. It must be set to true if an automatic estimation of the number 
of clusters via a geometric interpretation of the WCSS curve is to be computed, false otherwise.

- numberOfSteps: an integer representing the refresh step \(R\). That is, the measure uses the clustering 
algorithm to partition the dataset into \(k\) cluster only, for values of \(k\) multiples of \(R\).

- numberOfIteration: an integer indicating the maximum number of iterations to be executed by the 
partitional clustering algorithm.

- percentage: an integer in the range \([0,100]\) that represents the percentage of the numberOfIteration 
for the successive execution of the clustering algorithm.

- outputHierarchical: the file path name specifying where the hierarchical clustering algorithm stores 
its output files.

The interested reader is referred to [3] for an in depth treatment of the previous parameters.

The WithinClustersSumSquares.java class

This class is fully analogous to the NullMeasure.java class. It is an abstract class for:

- WCSSGeneric.java

- WCSSFast.java

\(^1\) It is a public member function which takes the desired new value as a parameter, validates it, and modifies the corresponding private variable.
In analogy with the NullMeasure.java class, each of the previous classes can be computed via the computeMeasure methods.

The Krzanowski and Lai measure

The method

Krzanowski and Lai [6] proposed an internal measure based on WCSS, but it is automatic, i.e., a numeric value for \( k^* \) is returned. Let

\[
DIFF(k) = (k - 1) \frac{m}{\text{WCSS}(k - 1)} - k \frac{m}{\text{WCSS}(k)}
\]  

(9)

Recall from the main manuscript the behavior of WCSS, as a function of \( k \) and with respect to \( k^* \). Based of those considerations, one expects the behavior of \( DIFF(k) \) to be:

(i) for \( k < k^* \), both \( DIFF(k) \) and \( DIFF(k + 1) \) should be large positive values.

(ii) for \( k > k^* \), both \( DIFF(k) \) and \( DIFF(k + 1) \) should be small values, and one or both might be negative.

(iii) for \( k = k^* \), \( DIFF(k) \) should be large positive, but \( DIFF(k + 1) \) should be relatively small (might be negative).

Based on these considerations, Krzanowski and Lai proposed to choose, as prediction of \( k^* \), the \( k \) maximizing:

\[
KL(k) = \left| \frac{DIFF(k)}{DIFF(k + 1)} \right|
\]  

(10)

That is,

\[
k^* = \arg \max_{2 \leq k \leq k_{max}} KL(k)
\]  

(11)

The KL package

In analogy with the nullmeasures package, it contains two main classes useful to compute the Krzanowski and Lai measure: the InputKL.java and the KrzanowskiLai.java classes.
The InputKL.java class

In analogy with the InputNullM.java class, it can be instantiated by one of the following constructors:

```java
public InputKL()

public InputKL(int kMin, int kMax,
String algorithmPath, String algCommandLine,
String outputPath, String algInputPath,
String algOutputPath)
throws NotValidOptionException, FileAlgNotFoundException,
InvalidCmndLineException, DirNotFoundException

public InputKL(int kMin, int kMax,
String algorithmPath, String algCommandLine,
String outputPath, String algInputPath,
String algOutputPath, boolean initExtFlag,
String initExtAlgPath, String initExtCommandLine,
String initExtInpPath, String initExtOutPath)
throws NotValidOptionException, FileAlgNotFoundException,
InvalidCmndLineException, DirNotFoundException

public InputKL(int kMin, int kMax,
String outputPath, String algInputPath,
String algOutputPath, String hAlgPath,
String type, String outputHierarchical)
throws NotValidOptionException, FileAlgNotFoundException,
InvalidCmndLineException, DirNotFoundException

public InputKL(int kMin, int kMax,
String algorithmPath, String algCommandLine,
String outputPath, String algInputPath,
String algOutputPath, String hAlgPath,
String type, String outputHierarchical)
throws NotValidOptionException, FileAlgNotFoundException,
InvalidCmndLineException, DirNotFoundException
```

The KrzanowskiLai.java class

This class is fully analogous to the NullMeasure.java class. It is an abstract class for:

- KLGeneric.java
- KLHierarchical.java
- KLHierarchicalInit.java

In analogy with the NullMeasure.java classes, each of the previous classes can be computed via the computeMeasure methods.
The Gap Statistics

The method

Recall that the “knee” in the WCSS curve can be used to predict $k^*$. Unfortunately, the localization of such a point may be subjective. Gap is an automatic method to estimate $k^*$, which is based on the following intuition about WCSS. The method computes the gap between the WCCS curve computed on a null model and the one computed on a real dataset. Since WCSS is expected to decrease sharply up to $k^*$, on the real dataset, while it is expected to have a nearly constant slope on the null model datasets, the size of the gap is expected to increase up to $k^*$ and then to decrease. Moreover, the WCSS curves are normalized via logs and a simulation error is also considered. Finally, it is worth pointing out that a more accurate prediction of $k^*$ is based on a Monte Carlo simulation, i.e., the method is executed several times and the most frequent outcome is taken as the prediction. The interested reader is referred to [8] for an in depth treatment of this measure.

The Gap package

In analogy with the nullmeasures package, it contains two main classes useful to compute Gap: the InputGap.java and the GapStatistics.java classes.

The InputGap.java class

In analogy with the InputNullM.java class, it can be instantiated by one of the following constructors:

public InputGap()

public InputGap(int kMin, int kMax, String algorithmPath, String algCommandLine, String outputPath, String algInputPath, String algOutputPath, int numberOfIteration, String resample, int numberOfRun)

throws NotValidOptionException, FileAlgNotFoundException, InvalidCmndLineException, DirNotFoundException

public InputGap(int kMin, int kMax, String algorithmPath, String algCommandLine, String outputPath, String algInputPath, String algOutputPath, boolean initExtFlag, String initExtAlgPath, String initExtCommandLine, String initExtInpPath, String initExtOutPath, int numberOfIteration, String resample, int numberOfRun)

throws NotValidOptionException, FileAlgNotFoundException, InvalidCmndLineException, DirNotFoundException
public InputGap(int kMin, int kMax, 
String algorithmPath, String algCommandLine, 
String outputPath, String algInputPath, 
String algOutputPath, int numberOfSteps, 
int numberOfIteration, int percentage, 
int numberOfResample, 
String resample, int numberOfRun) 
throws NotValidOptionException, FileAlgNotFoundException, 
InvalidCmdLineException, DirNotFoundException

public InputGap(int kMin, int kMax, 
String outputPath, String algInputPath, 
String algOutputPath, String hAlgPath, 
String type, String outputHierarchical, 
int numberOfIteration, String resample, 
int numberOfRun) 
throws NotValidOptionException, FileAlgNotFoundException, 
InvalidCmdLineException, DirNotFoundException

public InputGap(int kMin, int kMax, 
String algorithmPath, String algCommandLine, 
String outputPath, String algInputPath, 
String algOutputPath, String hAlgPath, 
String type, String outputHierarchical, 
int numberOfIteration, String resample, 
int numberOfRun) 
throws NotValidOptionException, FileAlgNotFoundException, 
InvalidCmdLineException, DirNotFoundException

It is worth pointing out that the fourth constructor defines the input parameters for the approximations of 
Gap proposed in [3].

With reference to the parameters detailed in Addition File 1 and for the InputWCSS.java class, the 
following three additional input parameters can be used, depending on the method:

- numberOfIteration: it is the number of iterations of Gap.
- resample: it is a string that represents the type of null model data generation (see Additional File 5).
- numberOfRun: it is the number of iterations of the Monte Carlo simulation.

The interested reader is referred to [20] for an in depth treatment of the previous parameters.

The GapStatistics.java class

This class is fully analogous to the NullMeasure.java class. It is an abstract class for:
In analogy with the NullMeasure.java classes, each of the previous classes can be computed via the computeMeasure methods.

FOM

The method

FOM is a family of internal validation measures specifically designed for microarray data [9]. It is based on the jackknife approach and has been designed for use as a relative measure assessing the predictive power of a clustering algorithm, i.e., its ability to predict the correct number of clusters in a dataset. We use the adjusted aggregate FOM for our experiments and, for brevity, we refer to it simply as FOM. The use of FOM in order to establish how many clusters are present in the data follows the same heuristic methodology outlined for WCSS, i.e., one tries to identify the “knee” in the FOM plot as a function of the number of clusters. The interested reader is referred to [9] for an in depth treatment of the measure.

The FOM package

In analogy with the nullmeasures package, it contains two main classes useful to compute FOM: the InputFOM.java and the FigureOfMerit.java class.

The InputFOM.java class

In analogy with the InputNullM.java class, it can be instantiated by one of the following constructors:

```java
public InputFOM()

public InputFOM(int kMin, int kMax,
                 String algorithmPath, String algCommandLine,
                 String outputPath, String algInputPath,
                 String algOutputPath, boolean adjustmentFactor,
                 boolean predictNOfCluster)
```

```java
throws NotValidOptionException, FileAlgNotFoundException,
InvalidCmndLineException, DirNotFoundExcepti
```
public InputFOM(int kMin, int kMax, 
String algorithmPath, String algCommandLine, 
String outputPath, String algInputPath, 
String algOutputPath, boolean initExtFlag, 
String initExtAlgPath, String initExtCommandLine, 
String initExtInpPath, String initExtOutPath, 
boolean adjustmentFactor, boolean predictNOfCluster) 
throws NotValidOptionException, FileAlgNotFoundException, 
InvalidCmndLineException, DirNotFoundException

public InputFOM(int kMin, int kMax, 
String outputPath, String algInputPath, 
String algOutputPath, boolean predictNOfCluster, 
int numberOfSteps, int numberOfIteration, 
int percentage, boolean adjustmentFactor) 
throws NotValidOptionException, FileAlgNotFoundException, 
InvalidCmndLineException, DirNotFoundException

public InputFOM(int kMin, int kMax, 
String algorithmPath, String algCommandLine, 
String outputHierarchical, String type, 
String hAlgPath, String type, 
boolean adjustmentFactor) 
throws NotValidOptionException, FileAlgNotFoundException, 
InvalidCmndLineException, DirNotFoundException

It is worth pointing out that the fourth constructor allows to define the input for the approximation of 
FOM proposed in [3].

With reference to the parameters detailed in Additional File 1 and for the InputWCSS.java class, the 
following additional input parameter can be used, depending on the method:

• adjustmentFactor: a boolean flag. It must be set to true if the adjustment factor is to be computed. 
  It is false otherwise.

The interested reader is referred to [3] for an in depth treatment of this parameter.
The FigureOfMerit.java class

This class is fully analogous to the NullMeasure.java class. It is an abstract class for:

- FOMGeneric.java
- FOMFast.java
- FOMHierarchical.java
- FOMHierarchicalInit.java

In analogy with the NullMeasure.java classes, each of the previous classes can be computed via the computeMeasure methods.

Diff-FOM

The method

Diff-FOM is an extension of KL to FOM. It is based on the computation of the following formula:

\[ D\text{DFOM}(k) = (k - 1)^{\frac{2}{m}} FOM(k - 1) - k^{\frac{2}{m}} FOM(k) \]

The “rule of thumb” that one uses to predict \( k^* \), via Diff-FOM, is the same as for KL, being based on the same intuition. The interested reader is referred to [3] for an in depth treatment of this measure.

The diffFOM package

In analogy with the nullmeasures package, it contains two main classes useful to compute Diff-FOM: the InputDiffFOM.java and the diffFOM.java classes.

The InputDiffFOM.java class

In analogy with the InputNullM.java class, it can be instantiated by one of the following constructors:

```java
public InputDiffFOM()

public InputDiffFOM(int kMin, int kMax,
                     String algorithmPath, String algCommandLine,
                     String outputPath, String algInputPath, String algOutputPath,
                     boolean adjustmentFactor, boolean predictNOfCluster)
```

Throws: NotValidOptionException, FileAlgNotFoundException, InvalidCmndLineException, DirNotFoundException
public InputDiffFOM(int kMin, int kMax,
String algorithmPath, String algCommandLine,
String outputPath, String algInputPath,
String algOutputPath, boolean initExtFlag,
String initExtAlgPath, String initExtCommandLine,
String initExtInpPath, String initExtOutPath,
boolean adjustmentFactor, boolean predictNOfCluster)
throws NotValidOptionException, FileAlgNotFoundException,
InvalidCmndLineException, DirNotFoundException

public InputDiffFOM(int kMin, int kMax, String outputPath,
String algInputPath, String algOutputPath,
boolean predictNOfCluster, String hAlgPath,
String type, String outputHierarchical,
boolean adjustmentFactor)
throws NotValidOptionException, FileAlgNotFoundException,
InvalidCmndLineException, DirNotFoundException

public InputDiffFOM(int kMin, int kMax,
String algorithmPath, String algCommandLine,
String outputPath, String algInputPath,
String algOutputPath, boolean predictNOfCluster,
String hAlgPath, String type,
String outputHierarchical, boolean adjustmentFactor)
throws NotValidOptionException, FileAlgNotFoundException,
InvalidCmndLineException, DirNotFoundException

It is worth pointing out that the fourth constructor allows to define the input for the approximation of

Diff-FOM proposed in [3].

The diffFOM.java class

This class is fully analogous to the NullMeasure.java class. It is an abstract class for:

- DiffFOMGeneric.java
- DiffFOMHierarchical.java
- DiffFOMHierarchicalInit.java

In analogy with the NullMeasure.java classes, each of the previous classes can be computed via the
computeMeasure methods.
CLEST

*The method*

CLEST generalizes in many aspects an approach proposed by Breckenridge for cluster validity and it can be regarded as a clever combination of hypothesis testing and stability-based techniques. It estimates the number of clusters in a dataset by iterating the following: randomly partition the original dataset in a learning and a training set. The first one is used to build a classifier for the data, then the classifier is used to derive “gold solution” partitions of the training set. It is then used to assess the quality of the partitions of the training set obtained by a given clustering algorithm. The interested reader is referred to [2] for an in depth treatment of this measure.

*The CLEST package*

In analogy with the `nullmeasures` package, it contains two main classes useful to compute the CLEST measure: the `InputCLEST.java` and the `CLEST.java` classes.

*The InputCLEST.java class*

In analogy with `InputNullM.java`, it can be instantiated by one of the following constructors:

```java
public InputCLEST()

public InputCLEST(int kMin, int kMax,
String algorithmPath, String algCommandLine,
String outputPath, String algInputPath,
String algOutputPath, int percentage,
int numberOfIteration, String resample,
int numberOfResample, String externalName,
double pMax, double dMin)
throws NotValidOptionException, FileAlgNotFoundException,
InvalidCmndLineException, DirNotFoundException

public InputCLEST(int kMin, int kMax,
String algorithmPath, String algCommandLine,
String outputPath, String algInputPath,
String algOutputPath, int percentage,
int numberOfIteration, String resample,
int numberOfResample, String externalName,
double pMax, double dMin,
boolean initExtFlag, String initExtAlgPath,
String initExtCommandLine, String initExtInpPath,
String initExtOutPath)
throws NotValidOptionException, FileAlgNotFoundException,
InvalidCmndLineException, DirNotFoundException
```

17
With reference to the parameters detailed in Additional File 1, the following five additional input parameters can be used, depending on the method:

- **percentage**: it is an integer in \([0, 100]\) representing the percentage of items extracted from the input dataset.
- **numberOfResample**: it is the number of resampling step.
- **externalName**: it is the name of an external measure.
- **pMax**: it is a real number in \([0, 100]\) representing the “significance level” threshold.
- **dMin**: it is a real number in \([0, 100]\) representing the minimum allowed difference between “computed and expected” values.

The interested reader is referred to [2] for an in depth treatment of the previous parameters.

**CLEST.java class**

It can be instantiated with one of the following two class constructors:

```java
public CLEST()
public CLEST(DataMatrix dataMatrix, InputCLEST mParameters)
```

In analogy with the `NullMeasure.java` class, it can be computed via the `computeMeasure` methods.

**Consensus**

**The method**

Consensus is a stability-based technique [5, 7]. Therefore, a large number of clustering solutions, each obtained via a sample of the original dataset, are used in order to identify the correct number of clusters based on the entries of a Consensus Matrix. Intuitively, the consensus matrix indicates the level of agreement of clustering solutions that have been obtained via independent sampling of the dataset. Based on experimental observations and sound arguments, Monti et al. [7] derive a “rule of thumb” in order to estimate the real number \(k^*\) of clusters present in \(D\). For brevity, in what follows, only the key points are presented. The interested reader can find a full discussion in [3–5, 7]. The empirical cumulative distribution of the entries of the consensus matrix is computed. In an ideal situation in which there are \(k\) clusters and the clustering algorithm is so good to provide a perfect classification, such a curve is bimodal, with peaks at zero and one. Monti et al. observe and validate experimentally that the area under the CDF curve is an
increasing function of \( k \). That result has also been confirmed by the experiments in Giancarlo et al. [3]. In particular, for values of \( k = k^* \), that area has a significant increase, while its growth flattens out for \( k > k^* \).

However, Monti et al. propose a closely associated method, described next. For a given \( k \), the area of the corresponding CDF curve is estimated and an increasing function \( \Delta \) is computed for successive values of \( k \). Again, Monti et al. observe experimentally that: (i) For each \( k \leq k^* \), there is a pronounced decrease of the \( \Delta \) curve. That is, the incremental growth of the area under the CDF decreases sharply. (ii) For \( k > k^* \), there is a stable plot of the \( \Delta \) curve. That is, for \( k > k^* \), the growth of the area flattens out. From this behavior, the “rule of thumb” to identify \( k^* \) with the use of the \( \Delta \) curve is: take as \( k^* \) the abscissa corresponding to the smallest non-negative value where the curve starts to stabilize; that is, no big variation in the curve takes place from that point on. Moreover, it is worth pointing out that FC, a speed up of Consensus, has been proposed in [4]. The main idea is that costly computational duplications are avoided when the clustering algorithm is hierarchical. Indeed, it becomes possible to interleave the computation of the measure with the level bottom-up construction of the hierarchical tree underlying the clustering algorithms. The interested reader is referred to [7] and [4,5] for an in depth treatment of Consensus and FC, respectively.

**The ConsensusC package**

In analogy with the `nullmeasures` package, it contains two main classes useful to compute Consensus: the `InputConsensus.java` and the `Consensus.java` class.

**The InputConsensus.java class**

In analogy with `InputNullM.java`, it can be instantiated by one of the following constructors:

```java
public InputConsensus()

public InputConsensus(int kMin, int kMax,
                      String algorithmPath, String algCommandLine,
                      String outputPath, String algInputPath,
                      String algOutputPath, int percentage,
                      int numberOfIteration, boolean hierarchical,
                      boolean replaceAk)
```

```java
throws NotValidOptionException, FileAlgNotFoundException,
         InvalidCmdLineException, DirNotFoundException
```

```java
public InputConsensus(int kMin, int kMax,
                      String algorithmPath, String algCommandLine,
                      String outputPath, String algInputPath,
                      String algOutputPath, int percentage,
```

19
public InputConsensus(int kMin, int kMax, String outputPath, String algInputPath, String hAlgPath, String type, int percentage, int numberOfIteration, boolean replaceAk)
throws NotValidOptionException, FileAlgNotFoundException, InvalidCmndLineException, DirNotFoundException

public InputConsensus(int kMin, int kMax, String algorithmPath, String algCommandLine, String outputPath, String algInputPath, String algOutputPath, String hAlgPath, String type, String outputHierarchical, int percentage, int numberOfIteration, boolean replaceAk)
throws NotValidOptionException, FileAlgNotFoundException, InvalidCmndLineException, DirNotFoundException

The first three constructors define the input for the implementation of Consensus proposed by Monti et al. [7], while all constructors can be used for the Fast Consensus algorithms proposed in [4].

With reference to the parameters detailed in Additional File 1, the following five additional input parameters can be used, depending on the method:

- percentage: it is an integer in [0, 100] representing the percentage of items extracted from the input dataset.
- numberOfIteration: it is the number of iteration steps.
- hierarchical: it is a boolean value indicating if the clustering algorithm is hierarchical
- replaceAk: it is a boolean value. When it is true, if for some $k$ the function $A(k)$ is less than $A(k - 1)$, then $A(k) = A(k - 1)$.

The Consensus.java class

It is an abstract class extending the Measure class and encapsulating methods and state information needed for the computation of both Consensus and Fast Consensus. The following classes are extension of Consensus.java:
In analogy with the `NullMeasure.java` classes, each of the previous classes can be computed via the `computeMeasure` methods.

**Model Explorer**

*The method*

The idea supporting Model Explorer is that $k^*$ has been identified once one finds a $k$ such that the partitions into $k$ clusters produced by a clustering algorithm are similar, when obtained by repeatedly subsampling of the dataset. The output of the measure is an array of dimension $(k_{max} - 1) \times \text{numberOfResample}$, analogous to the one computed by CLEST. Each row of the array is histogrammed separately, and the optimal number of clusters is predicted to be the lowest value of $k$ such that there is a transition of the value distribution from being close to one to a wider range of values. The interested reader is referred to [1] for an in depth treatment of this measure.

*The modelExplorer package*

In analogy with the `nullmeasures` package, it contains two main classes useful to compute the Model Explorer measure: the `InputModelExplorer.java` and the `ModelExplorer.java` classes.

*The InputModelExplorer.java class*

In analogy with `InputNullM.java`, it can be instantiated by one of the following constructors:

```java
public InputModelExplorer()

public InputModelExplorer(int kMin, int kMax,
    String algorithmPath, String algCommandLine,
    String outputPath, String algInputPath,
    String algOutputPath, int percentage,
    int numberOfResample, String externalName)

throws NotValidOptionException, FileAlgNotFoundException,
        InvalidCmndLineException, DirNotFoundException

public InputModelExplorer(int kMin, int kMax,
    String algorithmPath, String algCommandLine,
    String outputPath, String algInputPath,
    String algOutputPath, int percentage,
    int numberOfResample, String externalName)

throws NotValidOptionException, FileAlgNotFoundException,
        InvalidCmndLineException, DirNotFoundException
```

21
String algorithmPath, String algCommandLine, 
String outputPath, String algInputPath, 
String algOutputPath, int percentage, 
int numberOfResample, String externalName, 
boolean initExtFlag, String initExtAlgPath, 
String initExtCommandLine, String initExtInpPath, 
String initExtOutPath)
throws NotValidOptionException, FileAlgNotFoundException, 
InvalidCmndLineException, DirNotFoundException

The ModelExplorer.java class

It can be instantiated with one of the two following class constructors:

public ModelExplorer()
public ModelExplorer(DataMatrix dataMatrix, 
InputModelExplorer mParameters)

In analogy with the NullMeasure.java class, it can be computed via the computeMeasure methods. We remark that the output of this measure is provided as a multidimensional array stored into a text file due to the “rule of thumb” of this measure to identify $k^*$ (see [1,3]). Each line of the text file corresponds to the data vector obtained for each $k$.

References


