```java
package valworkbench.measures.internal.ConsensusC;

import java.io.BufferedReader;

/**
 * The ConsensusClustering class provides an object that encapsulates methods and
 * states information for computing Consensus Cluster measure. This class takes
 * in input a DataMatrix and an InputMeasure object. Values of
 * mesaure are returned in a MeasureVector object. The effect of
 * computing such measure is basically to find a correct number of clusters with
 * the given algorithm A and benchmark dataset D.
 *
 *
 *
 * @author Raffaele Giancarlo
 * @author Filippo Utro
 * @author Davide Scaturro
 * @version 1.0
 * @see Measure
 *
 */
public class ConsensusClustering extends Measure {
    private ConsensusMatrix consensusMatrix;
    private final String MEASURE_NAME = "Consensus Clustering";
    private InputConsensus parameters;
    private DataMatrix dataMatrix;
    private MeasureVector measureValue, AkVector, minRelative;
    private HeaderData headerData;
    private String pathOutputCDF, pathOutputElements, pathOutputConsMatrix,
    pathAk;

    /**
     * Default class Constructor
     *
     */
    public ConsensusClustering() {
        // Empty constructor method
    }

    /**
     * Class constructor specifying input data Matrix and input
```

```
   parameters
62      *
63      * @param dataMatrix input data Matrix
64      * @see DataMatrix
65      * @param mParameters measure parameters
66      * @see InputMeasure
67      *
68      */
69     public ConsensusClustering(DataMatrix dataMatrix,
   InputConsensus mParameters)throws DirNotFoundException
70     {
71         this.parameters = mParameters;
72
73         this.dataMatrix = dataMatrix;
74
75         this.pathOutputCDF = this.parameters.getCurrentAuxDir()
76         + File.separator + "CDF.txt";
77
78         this.pathOutputElements =
   this.parameters.getCurrentAuxDir()
79         + File.separator + "Elements.txt";
80
81         this.pathOutputConsMatrix =
   this.parameters.getCurrentAuxDir()
82         + File.separator + "Consensus_Matrix.txt";
83
84         this.pathAk = this.parameters.getCurrentAuxDir() +
   File.separator
85         + "AK.txt";
86     }
87
88     /**
89      * Computes measure values
90      *
91      * @see InputMeasure
92      * @return a Measure_Vector object containing measure values
93      * @see MeasureVector
94      */
95     public MeasureVector computeMeasure() throws
   FileNotFoundException, IOException, NumberFormatException,
   Exception
96     {
97
98         // reads number of items
99         int N = this.dataMatrix.getNOfItem();
```

```
100
101         int B = this.parameters.getNumberOfIteration();
102         int nGeneselect = (int) ((N *
    this.parameters.getPercentage()) / 100);
103         double ak[] = new double[this.parameters.getKMax()
104                                  - this.parameters.getKMin() + 1];
105         double dK[] = new double[this.parameters.getKMax()
106                                  - this.parameters.getKMin() + 1];
107
108         boolean flagOut;
109         String commandLine = null;
110         int exitValue;
111         double max_ak=0;
112
113         this.measureValue = new
    MeasureVector(this.parameters.getKMax()
114                 - this.parameters.getKMin() + 1);
115
116         this.AkVector=new MeasureVector(this.parameters.getKMax()
117                 - this.parameters.getKMin() + 1);
118
119         this.minRelative = new MeasureVector(1);
120         // instance of the matrices
121         IndicatorMatrix indMatrix = new IndicatorMatrix(N);
122         ConnettivityMatrix connMatrix = new ConnettivityMatrix(N);
123         this.consensusMatrix = new ConsensusMatrix(N);
124
125         // instance temporary matrix
126         IndicatorMatrix tmpindMatrix = new IndicatorMatrix(N);
127         ConnettivityMatrix tmpconnMatrix = new
    ConnettivityMatrix(N);
128         int Nmissing = 0;
129
130         for (int k = 0; k < this.parameters.getKMax()-
    this.parameters.getKMin() + 1; k++) {
131
132             System.out.println("Execute clustering algorithm "
133                     + parameters.getAlgorithmName() + " for "
134                     + Integer.toString(k +
    this.parameters.getKMin())
135                     + " clusters");
136
137             int missingOut = 0;
138
139             //Inizializes Consensus Matrix et all
```

```
140             this.consensusMatrix.setConsensusMatrixtoZero();
141             indMatrix.setIMatrixtoZero();
142             connMatrix.setConnMatrixtoZero();
143 
144             //Iteration cicle for Consensus
145             for (int b = 0; b < B; b++) {
146 
147                 System.out.print("Compute on the resample data at
    the "
148                         + Math.floor((double) (b + 1) / B * 100) +
    "% \n");
149 
150                 tmpindMatrix.setIMatrixtoZero();
151                 tmpconnMatrix.setConnMatrixtoZero();
152                 //Resample data
153 
154                 //Creates item permutation and selects items for
    the computation
155                 int permutation[] =
    permutation(this.dataMatrix.getNOfItem());
156 
157                 int geneData[] = new int[nGeneselect];
158                 for (int s = 0; s < nGeneselect; s++) {
159                     geneData[s] = permutation[s];
160                 }
161 
162                 resample(nGeneselect, this.dataMatrix,
    permutation);
163 
164                 //Executes Algorithm for resample data
165 
166                 //Defines output path
167                 String tempPath = parameters.getAlgOutputPath()
168                 + Integer.toString(k + this.parameters.getKMin())
169                 + ".txt";
170 
171                 if (parameters.isInitExtFlag()) {
172 
173                     // Defines command Line for the external
    algorithm
174                     commandLine = composeCmndLine(k +
    parameters.getKMin(),
175                             "\""+parameters.getInitExtAlgPath()
    +"\"", "\""+parameters
176                             .getInitExtInpPath()+"\"",
```

```
    "\""+parameters
177                             .getInitExtOutPath()+"\"", null, 0,
    parameters
178                             .getInitExtCommandLine(), false);
179
180                     // Executes algorithm for external
    initialization
181                     exitValue = this.executeAlgorithm(commandLine,
    parameters
182                             .getInitExtOutPath());
183
184                     // checks correct execution
185                     if (exitValue == 1) {
186                         // if incorrect execution, sets random
    inizialization
187                         parameters.setInitExtFlag(false);
188                     }
189                 }
190
191                 // Composes command line for algorithm execution
192                 commandLine = composeCmndLine(k +
    this.parameters.getKMin(),
193                         "\""+parameters.getAlgorithmPath()+"\"",
    "\""+parameters
194                         .getAlgInputPath()+"\"",
    "\""+tempPath+"\"", "\""+parameters
195                         .getInitExtOutPath()+"\"", 0, parameters
196                         .getAlgCommandLine(), parameters
197                         .isInitExtFlag());
198
199                 // Executes the algorithm
200                 exitValue = this.executeAlgorithm(commandLine,
    tempPath);
201
202                 // Checks correct execution
203                 flagOut = false;
204                 if (exitValue == 1) {
205                     System.out.println("Missing execution in
    iteration "+b);
206                 } else if (exitValue == 0) {
207                     // Checks correct output (i.e. Missing out)
208                     flagOut = checkOut(tempPath);
209                 }
210
211                 if (flagOut) {
```

```
212                     // Loads cluster matrix
213                     ClusterMatrix clusterMatrix = new
    ClusterMatrix();
214                     clusterMatrix.loadFromFile(tempPath);
215
216                     ClusterMatrix clusterMatrixReal = unResample(
217                             clusterMatrix, permutation, k
218                             + this.parameters.getKMin());
219
220                     // Calculates Connettivity and indicator
    Matrix
221                     tmpindMatrix.setIMatrix(geneData);
222
    tmpconnMatrix.setConnMatrix(clusterMatrixReal);
223
224                     // Sums Inidicator and Connettivity Matrix
225                     indMatrix.addIMatrix(tmpindMatrix);
226                     connMatrix.addConnMatrix(tmpconnMatrix);
227
228                     missingOut++;
229                 }
230             }
231
232             System.out.println("");
233
234             double cdfVector[];
235
236             // Checks number of missing
237             if (((double) (missingOut / B)) > 0.1) {
238
239                 // Calculates Consensus Matrix
240
    this.consensusMatrix.setConsensusMatrix(connMatrix, indMatrix);
241
242                 // Calculates CDF values
243                 double sortVectorelement[] = this.consensusMatrix
244                 .vectorSortElement();
245                 cdfVector = new double[sortVectorelement.length];
246                 for (int s = 0; s < sortVectorelement.length; s++)
    {
247                     cdfVector[s] = this.consensusMatrix
248                     .calculateCDF(sortVectorelement[s]);
249                 }
250
251                 // Calculates Ak values
```

```
252                 ak[k] = calulateAk(sortVectorelement, cdfVector);
253                 if(k==0)
254                 {
255                   max_ak=ak[k];
256                 }
257                 if ((!
    this.parameters.isHierarchical())&&(this.parameters.isReplaceAk())
    )
258                 {
259                   if (ak[k]>max_ak)
260                   {
261                       max_ak=ak[k];
262                   }
263                   else
264                   {
265                       ak[k]=max_ak;
266                   }
267                 }
268                 missingOut = 1;
269                 Nmissing++;
270             } else {
271                 ak[k] = Double.NaN;
272                 missingOut = 0;
273                 cdfVector = new double[1];
274 
275             }
276 
277             // Stores CDF, Elements and Consensus Matrix values in
    a file
278 
    this.consensusMatrix.storeToFileCDF(this.pathOutputCDF, k
279                     + this.parameters.getKMin(), cdfVector,
    missingOut);
280 
    this.consensusMatrix.storeToFileElement(this.pathOutputElements,
281                     missingOut);
282             this.consensusMatrix.storeToFileConsensusMatrix(
283                     this.pathOutputConsMatrix, k +
    this.parameters.getKMin(),
284                     missingOut);
285 
286             this.measureValue.setNOfCluster(k, k +
    this.parameters.getKMin());
287         }
288 
```

```
289         if(ak[this.parameters.getKMax()-
    this.parameters.getKMin()]<ak[this.parameters.getKMax()-
    this.parameters.getKMin()-1])
290           ak[this.parameters.getKMax()-
    this.parameters.getKMin()]=ak[this.parameters.getKMax()-
    this.parameters.getKMin()-1];
291         // Calculates dk values
292         dK = calculateDk(ak);
293 
294         // Writes Output and Ak values
295         this.measureValue.setMeasureValue(dK);
296         storeToFileAk(ak);
297 
298 
299         this.headerData = this.writeHeader();
300         this.headerData.storeToFile(parameters.getOutputPath()
301                 + File.separator + "Measure_Header.vhf");
302         this.measureValue.storeToFile(parameters.getOutputPath()
303                 + File.separator + "Measure_Values.txt");
304 
305         //Extract Suggested Number of Cluster
306         int nOfCluster=0;
307         while((nOfCluster<(this.parameters.getKMax()-
    this.parameters.getKMin()+1))&&(dK[nOfCluster]>dK[nOfCluster+1]))
308             nOfCluster++;
309 
310         this.minRelative.setNOfCluster(0,
    nOfCluster+this.parameters.getKMin());
311 
312         this.minRelative.setMeasureValue(0, dK[nOfCluster]);
313 
314         this.minRelative.storeToFile(parameters.getOutputPath()
315                 + File.separator + "nOfCluster_Value.txt");
316         //Ends to compute measure
317         return this.measureValue;
318     }
319 
320     /**
321      * Creates an array of a pseudorandom permutation of n numbers
322      *
323      * @param n
324      * @return an array with a pseudorandom permutation of n.
325      */
326     private int[] permutation(int n) {
327         int[] v = new int[n];
```

```
328         int i;
329         for (i = 0; i < n; i++) {
330             v[i] = i;
331         }
332         // Shuffles
333         for (i = 0; i < n; i++) {
334             int r = (int) (Math.random() * (i + 1)); // int
    between 0 and i
335             int swap = v[r];
336             v[r] = v[i];
337             v[i] = swap;
338         }
339         return v;
340     }
341 
342     /**
343      * Creates a file containing the resample data matrix
344      *
345      * @param nItemselect number of items for the new dataset
346      * @param dataMatrix a reference to a Data_Matrix object
347      * @see DataMatrix
348      * @param permutation a vector of integer containing
    permutation for the resample
349      */
350     public void resample(int nItemselect, DataMatrix dataMatrix,
    int  permutation[]) throws Exception
351     {
352         DataMatrix pMatrix = new DataMatrix(nItemselect,
    dataMatrix
353                 .getNOfFeature());
354 
355         DataMatrix pattMatrix = dataMatrix.copyDMatrix();
356 
357         // Swaps of rows
358         pMatrix.setFeatureName(pattMatrix.getFeatureName());
359         for (int i=0;i<nItemselect;i++)
360         {
361             pMatrix.setItemName(i,
    pattMatrix.getItemName(permutation[i]));
362             pMatrix.setItemDescription(i,
363 
    pattMatrix.getItemDescription(permutation[i]));
364 
365 
    pMatrix.setItemValueRow(i,pattMatrix.getItemValueRow(permutation[i
```

```
]));
366         }
367 
368         // Writes the resample dataset
369         pMatrix.storeToFile(this.parameters.getAlgInputPath());
370 
371     }
372 
373     /**
374      * Calculates the Ak values for the Consensus Matrix
375      *
376      * @param sortVectorelement a vector of double containing
sorted elements of Consensus Matrix
377      * @param cdfVector a vector of double for the CDF values
378      * @return the Ak values
379      */
380     public double calulateAk(double sortVectorelement[], double
cdfVector[]) {
381 
382         double ak = 0;
383         for (int i = 1; i < sortVectorelement.length; i++) {
384             ak += (sortVectorelement[i] - sortVectorelement[i -
1])
385             * cdfVector[i];
386         }
387 
388         return ak;
389     }
390 
391     /**
392      * Calculates the dk values.
393      *
394      * @param ak a vector of double for ak values.
395      * @return a vector of double containing the dk values.
396      */
397     public double[] calculateDk(double ak[]) {
398 
399         double dk[] = new double[this.parameters.getKMax()
400                                  - this.parameters.getKMin() + 1];
401 
402         int missing = 0;
403         for (int s = 0; s < ak.length; s++)
404             if (ak[s] != Double.NaN) {
405                 missing++;
406             }
```

```
407
408         // Missing management
409
410         double aknoMissing[] = new double[missing];
411         int j = 0;
412
413         for (int s = 0; s < ak.length; s++)
414             if (ak[s] != Double.NaN) {
415                 aknoMissing[j] = ak[s];
416                 j++;
417             }
418
419         j = 1;
420
421         for (int i = 0; i < (this.parameters.getKMax() -
    this.parameters.getKMin() + 1); i++) {
422             if ((ak[i] != Double.NaN) & (j < missing )) {
423                 if (j > 1) {
424                     dk[i] = (aknoMissing[j] - aknoMissing[j - 1])
425                     / aknoMissing[j - 1];
426                 } else {
427                     dk[i] = aknoMissing[j-1];
428                 }
429                 j++;
430             } else
431                 dk[i] = Double.NaN;
432         }
433
434         return dk;
435     }
436
437     /**
438      * Checks if the outuput file produced by the external
    algorithm is a Missing
439      *
440      * @param path the output path
441      * @return a boolean value equal to true if is not missing
    output, false otherwise.
442      */
443     private boolean checkOut(String path) throws
    FileNotFoundException, IOException, NumberFormatException,
    Exception {
444
445         File fac = new File(path);
446         FileInputStream fis = new FileInputStream(fac);
```

```
447         InputStreamReader isr = new InputStreamReader(fis);
448         BufferedReader br = new BufferedReader(isr);
449         String linea = br.readLine();
450 
451         if (linea != null) {
452             return true;
453         }
454 
455 
456         return false;
457 
458     }
459 
460     /**
461      * Collects information about the experiment.
462      *
463      */
464     protected HeaderData writeHeader() {
465         HeaderData header = new HeaderData();
466         String stringParam = null;
467 
468         if (this.parameters.isInitExtFlag()) {
469 
    header.setAlgorithmName(this.parameters.getAlgorithmName() + "
    init. "
470                     + this.parameters.getInitAlgName());
471         } else {
472 
    header.setAlgorithmName(this.parameters.getAlgorithmName());
473         }
474         stringParam = Integer.toString(this.parameters.getKMin())
    + "_"
475         + Integer.toString(this.parameters.getKMax());
476         String algParam = "";
477         StringTokenizer st = new
    StringTokenizer(this.parameters.getAlgCommandLine());
478         for (int i = 0; i <= st.countTokens(); i++) {
479             String temp = st.nextToken();
480             if (temp.equals("<inputfile>"))
481                 ;
482             else if (temp.equals("<outputfile>"))
483                 ;
484             else if (temp.equals("<nofcluster>"))
485                 ;
486             else if (temp.equals("<extinit>"))
```

```
487             {
488                 StringTokenizer  stInit = new
    StringTokenizer(this.parameters.getInitExtCommandLine());
489                 for (int j = 0; j <= stInit.countTokens(); j++)
490                 {
491                     String temp2 = stInit.nextToken();
492                     if (temp2.equals("<inputfile>"));
493                     else if (temp2.equals("<outputfile>"));
494                     else if (temp2.equals("<nofcluster>"));
495                     else
496                     {
497                         algParam = algParam.concat(" "+temp2);
498                     }
499                 }
500             }
501             else
502             {
503                 algParam = algParam.concat(temp);
504             }
505         }
506         header.setAlgParameters(stringParam + " " + algParam);
507         header.setDatasetName(this.parameters.getDatasetName());
508         header.setDatasetType(this.parameters.getDatasetType());
509         header.setDateTime();
510         header.setMeasureName(this.MEASURE_NAME);
511         String parametri = "";
512         parametri = parametri + " Number of Iterations:"
513         + this.parameters.getNumberOfIteration()
514         + " Percentage of Dataset:" +
    this.parameters.getPercentage()
515         + "%";
516         header.setMeasParameters(parametri);
517         return header;
518     }
519
520     /**
521      * Finds the relation between the original data matrix and the
    resample data
522      * matrix
523      *
524      * @param clusterMatrix a reference to a Cluster_Matrix object
    containing
525      * the resample data matrix
526      * @see ClusterMatrix
527      * @param permutation a vector of integer containing a
```

```
   permutation
528      * @param k the number of clusters
529      * @return the correct index of cluster matrix for the
   resample data matrix
530      * @see ClusterMatrix
531      */
532     public ClusterMatrix unResample(ClusterMatrix clusterMatrix,
533             int[] permutation, int k) {
534         ClusterMatrix realClusterMatrix = new ClusterMatrix(k);
535 
536         for (int i = 0; i < k; i++) {
537             int cl[] = clusterMatrix.getCMatrixRow(i);
538             int rl[] = new int[cl.length];
539 
540             realClusterMatrix.setClusterSize(i, cl.length);
541             realClusterMatrix.instanceCMatrixRow(i, cl.length);
542 
543             for (int j = 0; j < cl.length; j++) {
544                 rl[j] = permutation[cl[j]];
545             }
546 
547             realClusterMatrix.setCMatrixRow(i, rl);
548         }
549 
550         return realClusterMatrix;
551     }
552 
553     /**
554      * Stores the Ak value in a file.
555      *
556      * @param ak a vector of double containing the Ak values.
557      */
558     private void storeToFileAk(double ak[]) throws
   FileNotFoundException, IOException, NumberFormatException,
   Exception
559     {
560 
561         // Opens file in append
562         FileOutputStream file = new FileOutputStream(this.pathAk);
563         PrintStream Output = new PrintStream(file);
564 
565         // Writes the file
566         for (int i = 0; i < ak.length; i++) {
567             Output.println("--- Ak " + (i +
   this.parameters.getKMin())
```

```
568                     + " ---");
569             if (ak[i] != Double.NaN) {
570                 Output.println("| " + ak[i]);
571             } else {
572                 Output.println("* Missing");
573             }
574             Output.println("--- End Ak " + (i +
    this.parameters.getKMin())
575                     + " ---");
576         }
577 
578         // Closes the file
579         Output.close();
580     }
581 
582     /**
583      * Return  the k-th Consensus Cluster Matrix in
    Consensus_Matrix.txt in auxiliary directory
584      * @param k the number of clusters
585      * @return the Consensus Matrix
586      * @see ConsensusMatrix
587      */
588     public ConsensusMatrix getConsensusMatrix(int k) throws
    FileNotFoundException, IOException, NumberFormatException,
    Exception
589     {
590         ConsensusMatrix matrix=new
    ConsensusMatrix(this.dataMatrix.getNOfItem());
591 
    matrix.loadFromFileConsensusMatrix(this.pathOutputConsMatrix, k);
592 
593         return matrix;
594     }
595 
596 
597     /**
598      * Return  the elements of k-th Consensus Cluster Matrix in
    Elements.txt in auxiliary directory
599      * @param k number of clusters
600      * @return the consensus matrix
601      */
602     public List<Double> getElements(int k) throws
    FileNotFoundException, IOException, NumberFormatException,
    Exception
603     {
```

```
604         ConsensusMatrix matrix=new
    ConsensusMatrix(this.dataMatrix.getNOfItem());
605         return matrix.getElemnts(this.pathOutputElements, k,
    this.parameters.getKMin());
606 
607     }
608 
609     /**
610      * Gets a Measure_Vector containing the Ak values previously
    computed
611      * @see MeasureVector
612      * @return a reference to a Measure_Vector object
613      *
614      */
615     public MeasureVector getAkVector() {
616         return AkVector;
617     }
618 
619 
620 }
621 
```