User Manual for KCC—a MATLAB package for K-means-based Consensus Clustering

Hao Lin[1], Hongfu Liu[2], Junjie Wu[3], Hong Li[4], Stephan Günnemann[5]

1. INTRODUCTION

This user manual systematically presents the usage of the MATLAB package on K-means-based Consensus Clustering (KCC) accompanying the following paper:

> Hao Lin, Hongfu Liu, Junjie Wu, Hong Li, and Stephan Günnemann. 2022. Algorithm xxxx: KCC: A MATLAB Package for K-means-based Consensus Clustering. *ACM Trans. Math. Softw.*

The package was developed and tested in MATLAB R2012a under Linux. To those without access to MATLAB and those who prefer to use free open source software, we also investigate the usage of KCC with OCTAVE, and find it is also compatible with OCTAVE without additional efforts. OCTAVE version?.

For package installation, you need to first unpack the compressed archive into your current directory. It consists of a *source code* folder Matlab, a folder userManual with this comprehensive *user manual*, and a *license* file LICENSE indicating that the package is distributed under GNU GENERAL PUBLIC LICENSE (Version 3). Then under the Matlab folder, you need to add one of its subfolder, i.e., the Src folder, to the MATLAB path. The directory structure of the Matlab folder is described as follows.

- Src (core functions for conducting KCC)
  - BasicCluster_RFS.m (function to generate BPs with RFS)
  - BasicCluster_RPS.m (function to generate BPs with RPS)
  - Preprocess.m (function to prepare for consensus clustering)
  - KCC.m (consensus function)
  - exMeasure.m (function to compute validity scores for clustering results)
  - load_sparse.m (auxiliary function to load input text data as a sparse matrix)
  - hungarian.m (auxiliary function for cluster label assignment)
  - BasicCluster_RPS_missing.m (auxiliary function to generate IBPs with strategy-I)
  - addmissing.m (auxiliary function to generate IBPs using strategy-II)
  - distance_* (distance functions)
  - gClusterDistribution.m (auxiliary function to calculate cluster distribution for BPs)
  - Ucompute.m, Ucompute_miss.m (auxiliary function for utility calculation)
  - gCentroid.m, gCentroid_miss.m (auxiliary function for centroid update)
  - sCentroid.m, sCentroid_miss.m (auxiliary function for centroid initialization)
- Drivers (illustrative examples)
  - data (input data for illustration)
  - demo.m (function for KCC with different utility functions)

[1]Department of Informatics, Technical University of Munich, Germany, linh@in.tum.de.
[2]Michtom School of Computer Science, Brandeis University, USA, hongfuliu@brandeis.edu.
[3]School of Economics and Management, Beihang University, China, wujj@buaa.edu.cn.
[4]School of Economics and Management, Beihang University, China, hong_lee@buaa.edu.cn.
[5]Department of Informatics, Technical University of Munich, Germany, guennemann@in.tum.de.

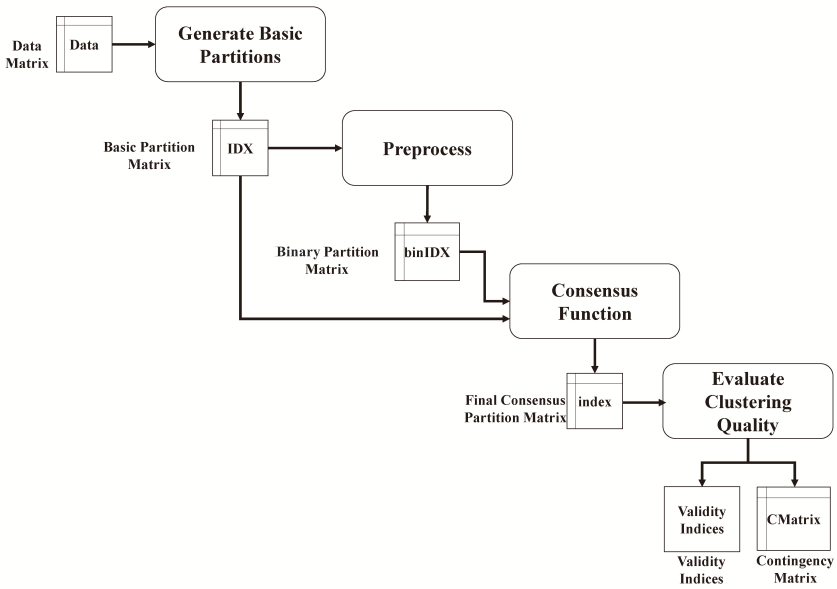

Fig. 1. Typical program flow of KCC package.

demoIBPI.m (function for KCC with IBPs generated by strategy-I)
demoIBPII.m (function for KCC with IBPs generated by strategy-II)
demoNumberBP.m (function for KCC with varying number of BPs)
demoStrategyBP.m (function for KCC with RFS strategy for BP generation)

Figure 1 illustrates a typical program flow of using the KCC package, which includes data preparation, basic partitions generation, consensus clustering preprocessing, consensus function, and clustering quality evaluation. We can see from Figure 1 that, in a typical flow for consensus clustering, a real-world matrix Data is first input to generate a basic partition matrix IDX. The basic partition matrix is then input to a Preprocess function to produce the sparse representation of $\mathcal{X}^{(b)}$, i.e., the binary matrix binIDX. The binary matrix binIDX, along with the basic partition matrix IDX is the input of the final consensus clustering via a K-means heuristic, also known as consensus function, which produces a consensus partition matrix index. Lastly, the clustering quality is evaluated with an exMeasure function, which outputs several external validity indices and a contingency matrix. We will take a detailed look at the important functions and fields in the Guides in Section 2.

2. GUIDES

2.1 BasicCluster_RFS

This function generates basic partition results using K-means as a basic clustering algorithm with Random Feature Selection (RFS) strategy.

Syntax:

```
IDX = BasicCluster_RFS(Data, r, K, dist, nFeature)
```

Input parameters:

`Data` : input data matrix
`r` : the predefined number of basic partitions in the cluster ensemble
`K` : the predefined number of clusters in the basic partitions
`dist` : the distance measure used in K-means clustering
`nFeature` : the number of randomly selected partial features

Output parameters:

`IDX` : a matrix indicating the cluster labels for data points in basic partitions

Discussion:

The parameter `Data` is an `n` × `p` matrix of data, whose rows correspond to `n` observations, and columns correspond to `p` features. The available options of the parameter `dist` can be found from the official documentation of the Matlab `kmeans` function, and the most widely used distance metric is `'sqEuclidean'`, which denotes the squared Euclidean distance. For `'sqEuclidean'`, the centroid for each cluster is calculated as the mean of the data points in that cluster. For selecting features, we first sample `nFeature` values uniformly at random without replacement from the integers `1` to `p`, where `p` is the dimension of all features in the input data. Then the sampled values are used as the indices of the selected features. The output `IDX` is a `n` × `r` cluster labels matrix for `n` data points in `r` basic partitions.

2.2 BasicCluster_RPS

This function generates basic partition results using K-means as a basic clustering algorithm with Random Parameter Selection (RPS) strategy.

Syntax:

```
IDX = BasicCluster_RPS(Data, r, K, dist, randKi)
```

Input parameters:

`Data` : input data matrix
`r` : the predefined number of basic partitions in the cluster ensemble
`K` : the groundtruth number of clusters for the input data
`dist` : the distance measure used in K-means clustering
`randKi` : the parameter regarding the number of clusters in the basic partitions

Output parameters:

`IDX` : a `n` × `r` cluster labels matrix for `n` data points in `r` basic partitions

Discussion:

The parameter `K` indicates the groundtruth number of clusters for the input data, which is used as the lower bound of the randomized number of cluster for each basic partition when `randKi == 1`. The parameter `randKi` has the following options. If `randKi == 1`, this

function generates a random number of cluster ranging from K to sqrt(n), where n is the number of input data instances; if randKi is set to a r × 1 vector, this function produces r basic partitions of which the i-th BP's number of clusters is RandKi(i); otherwise, this function produces r basic partitions with each partition having equal number (i.e., K) of clusters.

2.3 Preprocess

This function conduct some preprocessing on the input basic partition matrix IDX to produce the input for the final consensus clustering, as well as some auxiliary output variables that can help to save memory and accelerate computations.

Syntax:

```
  Ki, sumKi, binIDX, missFlag, missMatrix, distance, Pvector, weight =
Preprocess(IDX, U, n, r, w, utilFlag)
```

Input parameters:

IDX : the input basic partition matrix
U : the parameter regarding the chosen utility function
n : the number of data instances
r : the number of basic partitions in the cluster ensemble
w : the weight vector for all basic partitions
utiFlag : whether to calculate utility function

Output parameters:

Ki : a row vector indicating the number of clusters in all basic partitions
sumKi : a matrix indicating the starting indexes for all basic partitions
binIDX : a sparse representation of binarization of IDX
missFlag : whether the input IDX matrix contains IBPs or not
missMatrix : indices matrix of the non-zero entries in IDX if there exists IBPs
distance : the distance function corresponding to a specific utility function
Pvector : a row vector calculated from the contingency matrix
weight : an adjusted weight vector adapted for convenient distance calculation

Discussion:

The parameter U is a 1 × 3 cell array. Its first cell U{1,1} defines the chosen type of the KCC utility function. It currently supports four different types of utility functions which correspond to four different K-means point-to-centroid distance functions, i.e., 'U_c' for Euclidean distance, 'U_H' for Kullback-Leibler Divergence, 'U_cos' for cosine similarity, and 'U_Lp' for Lp-norm. It is worth noting that 'U_Lp' corresponds the distance measure in L_p spaces, which are function spaces defined using a natural generalization of the p-norm for finite-dimensional vector spaces. The second cell U{1,2} is a parameter specifying the chosen form of the KCC utility function, i.e., 'std' for the Standard Form, and 'norm' for the Normalized Form. The third cell U{1,3} is only required to be set when 'U_Lp' is chosen as the utility function. The settings of p = 1, p = 2, p → ∞ correspond to

the Manhattan distance, the euclidean distance and the chebyshev distance, respectively. The parameter `w` is `r` × 1 weight vector, of which each entry indicates the weight value assigned to each basic partition in the optimization objective of consensus clustering. The parameter `utiFlag` is a variable indicating whether to calculate utility function during the iterative process of K-means clustering.
The output variable missFlag $\in \{0, 1\}$ indicates whether the input IDX matrix contains incomplete basic partitions (IBPs) or not. The output missMatrix is a mask matrix which represents the indices of the non-zero entries in IDX if there exists IBPs. The output variable distance determines the corresponding distance function to deal with the specific utility function defined by U{1,1}. The output vector Pvector is a 1 × `r` row vector calculated from the contingency matrix, i.e., $P_k^{(i)}$, which can later be used in calculating distance and utility functions. The output vector weight is a `r` × 1 adjusted weight vector adapted for convenient distance calculation in later K-means heuristic.

2.4 KCC

This function employs the K-means heuristic to generate a final consensus partition.

Syntax:

```
sumbest, index, converge, utility = KCC(IDX, K, U, w, weight, distance,
maxIter, minThres, utilFlag, missFlag, missMatrix, n, r, Ki, sumKi, binIDX,
Pvector)
```

Input parameters:

`IDX` : the input basic partition matrix
`K` : the number of clusters in the consensus partition
`U` : the parameter regarding the chosen utility function
`w` : the weight vector for all basic partitions
`weight` : an adjusted weight vector adapted for convenient distance calculation
`distance` : the distance function corresponding to a specific utility function
`maxIter` : the maximum number of iterations for the convergence
`minThres` : the minimum reduced threshold of objective function
`utiFlag` : whether to calculate utility function
`missFlag` : whether there exist IBPs in IDX
`missMatrix` : the indices matrix of the non-zero entries in IDX if there exists IBPs
`n` : the number of data points
`r` : the number of basic partitions in the cluster ensemble
`Ki` : a row vector indicating the number of clusters in all basic partitions
`sumKi` : a matrix indicating the starting indexes for all basic partitions
`binIDX` : a sparse representation of binarization of IDX
`Pvector` : a row vector calculated from the contingency matrix

Output parameters:

`sumbest` : the optimal value of the consensus clustering's objective function
`index` : the label vector for the final consensus partition

`converge` : the iterative values of objective function
`utility` : the final utility function value

2.5 distance_euc

This function performs point-to-centroid distance calculation using euclidean distance measure.

Syntax:

```
D = distance_euc(U, C, weight, n, r, K, sumKi, binIDX)
```

Input parameters:

`U` : the 1 × 3 utility parameter cell array
`C` : the centroid matrix
`weight` : an r × 1 adjusted weight vector
`n` : the number of data points
`r` : the predefined number of basic partitions in the cluster ensemble
`K` : the predefined number of clusters in the consensus partition
`sumKi` : a matrix indicating the starting indexes for all basic partitions
`binIDX` : a sparse representation of binarization of IDX

Output parameters:

`D` : an n × K point-to-centroid distance matrix

2.6 distance_cos

This function performs point-to-centroid distance calculation using cosine distance measure.

Syntax:

```
D = distance_cos(U, C, weight, n, r, K, sumKi, binIDX)
```

Input parameters:

`U` : the 1 × 3 utility parameter cell array
`C` : the centroid matrix
`weight` : an r × 1 adjusted weight vector
`n` : the number of data points
`r` : the predefined number of basic partitions in the cluster ensemble
`K` : the predefined number of clusters in the consensus partition
`sumKi` : a matrix indicating the starting indexes for all basic partitions
`binIDX` : a sparse representation of binarization of IDX

Output parameters:

`D` : an n × K point-to-centroid distance matrix

2.7 distance_kl

This function performs point-to-centroid distance calculation using KL-divergence measure.

Syntax:

```
D = distance_kl(U, C, weight, n, r, K, sumKi, binIDX)
```

Input parameters:

`U` : the 1×3 utility parameter cell array

`C` : the centroid matrix

`weight` : an `r` $\times$ 1 adjusted weight vector

`n` : the number of data points

`r` : the predefined number of basic partitions in the cluster ensemble

`K` : the predefined number of clusters in the consensus partition

`sumKi` : a matrix indicating the starting indexes for all basic partitions

`binIDX` : a sparse representation of binarization of IDX

Output parameters:

`D` : an `n` $\times$ `K` point-to-centroid distance matrix

2.8 distance_lp

This function performs point-to-centroid distance calculation using L_p norm measure.

Syntax:

```
D = distance_lp(U, C, weight, n, r, K, sumKi, binIDX)
```

Input parameters:

`U` : the 1×3 utility parameter cell array

`C` : the centroid matrix

`weight` : an `r` $\times$ 1 adjusted weight vector

`n` : the number of data points

`r` : the predefined number of basic partitions in the cluster ensemble

`K` : the predefined number of clusters in the consensus partition

`sumKi` : a matrix indicating the starting indexes for all basic partitions

`binIDX` : a sparse representation of binarization of IDX

Output parameters:

`D` : an `n` $\times$ `K` point-to-centroid distance matrix

2.9 distance_euc_miss

This function performs point-to-centroid distance calculation over data with IBPs using euclidean distance measure.

Syntax:

```
D = distance_euc_miss(U, C, weight, n, r, K, sumKi, binIDX, missMatrix)
```

Input parameters:

`U` : the 1×3 utility parameter cell array

`C` : the centroid matrix

`weight` : an $r \times 1$ adjusted weight vector

`n` : the number of data points

`r` : the predefined number of basic partitions in the cluster ensemble

`K` : the predefined number of clusters in the consensus partition

`sumKi` : a matrix indicating the starting indexes for all basic partitions

`binIDX` : a sparse representation of binarization of IDX

`missMatrix` : the indices matrix of the non-zero entries in IDX if there exists IBPs

Output parameters:

`D` : an $n \times K$ point-to-centroid distance matrix

2.10 distance_cos_miss

This function performs point-to-centroid distance calculation over data with IBPs using cosine distance measure.

Syntax:

```
D = distance_cos_miss(U, C, weight, n, r, K, sumKi, binIDX, missMatrix)
```

Input parameters:

`U` : the 1×3 utility parameter cell array

`C` : the centroid matrix

`weight` : an $r \times 1$ adjusted weight vector

`n` : the number of data points

`r` : the predefined number of basic partitions in the cluster ensemble

`K` : the predefined number of clusters in the consensus partition

`sumKi` : a matrix indicating the starting indexes for all basic partitions

`binIDX` : a sparse representation of binarization of IDX

`missMatrix` : the indices matrix of the non-zero entries in IDX if there exists IBPs

Output parameters:

`D` : an $n \times K$ point-to-centroid distance matrix

2.11 distance_kl_miss

This function performs point-to-centroid distance calculation over data with IBPs using KL-divergence measure.

Syntax:

```
D = distance_kl_miss(U, C, weight, n, r, K, sumKi, binIDX, missMatrix)
```

Input parameters:

`U` : the 1×3 utility parameter cell array

`C` : the centroid matrix

`weight` : an `r` $\times$ 1 adjusted weight vector

`n` : the number of data points

`r` : the predefined number of basic partitions in the cluster ensemble

`K` : the predefined number of clusters in the consensus partition

`sumKi` : a matrix indicating the starting indexes for all basic partitions

`binIDX` : a sparse representation of binarization of IDX

`missMatrix` : the indices matrix of the non-zero entries in IDX if there exists IBPs

Output parameters:

`D` : an `n` $\times$ `K` point-to-centroid distance matrix

2.12 distance_lp_miss

This function performs point-to-centroid distance calculation over data with IBPs using L_p norm measure.

Syntax:

```
D = distance_lp_miss(U, C, weight, n, r, K, sumKi, binIDX, missMatrix)
```

Input parameters:

`U` : the 1×3 utility parameter cell array

`C` : the centroid matrix

`weight` : a `r` $\times$ 1 adjusted weight vector

`n` : the number of data points

`r` : the number of basic partitions in the cluster ensemble

`K` : the predefined number of clusters

`sumKi` : a matrix indicating the starting indexes for all basic partitions

`binIDX` : the sparse representation matrix

`missMatrix` : the indices matrix of the non-zero entries in IDX if there exists IBPs

Output parameters:

`D` : an `n` $\times$ `K` point-to-centroid distance matrix

2.13 UCompute

This function performs utility calculation on data sets without missing values.

Syntax:

```
util = UCompute(index, U, w, C, n, r, K, sumKi, Pvector)
```

Input parameters:

`index` : an `n` × 1 cluster assignment matrix
`U` : the 1 × 3 utility parameter cell array
`w` : the `r` × 1 adjusted weight parameter vector
`C` : the centroid matrix
`n` : the number of data points
`r` : the number of basic partitions in the cluster ensemble
`K` : the predefined number of clusters in the consensus partition
`sumKi` : a matrix indicating the starting indexes for all basic partitions
`Pvector` : a 1 × `r` row vector calculated from the contingency matrix

Output parameters:

`util` : the utility values calculated from the optimization objective of consensus clustering

Discussion:

The output `util` is a 2 × 1 cell array including a utility gain and an adjusted utility value.

2.14 UCompute_miss

This function performs utility calculation on data sets with missing values.

Syntax:

```
util = UCompute_miss(index, U, w, C, n, r, K, sumKi, Pvector, M)
```

Input parameters:

`index` : an `n` × 1 cluster assignment matrix
`U` : the 1 × 3 utility parameter cell array
`w` : the `r` × 1 adjusted weight parameter vector
`C` : the centroid matrix
`n` : the number of data points
`r` : the number of basic partitions in the cluster ensemble
`K` : the predefined number of clusters in the consensus partition
`sumKi` : a matrix indicating the starting indexes for all basic partitions
`Pvector` : a 1 × `r` row vector calculated from the contingency matrix
`M` : a mask matrix indicating the indices of the non-zero entries in IDX

Output parameters:

`util` : the utility values calculated from the optimization objective of consensus clustering

2.15 RunKCC

This function combines the two functions, i.e., Preprocess and KCC, for finding the consensus partition in one step.

Syntax:

```
[pi_sumbest, pi_index, pi_converge, pi_utility, t] = RunKCC(IDX, K, U, w,
rep, maxIter, minThres, utilFlag)
```

Input parameters:

`IDX` : the basic partition matrix
`K` : the predefined number of clusters in the consensus partition
`U` : the 1×3 utility parameter cell array
`w` : the $r \times 1$ adjusted weight parameter vector
`rep` : the number of repeated KCC experiments for selecting the best result
`maxIter` : the maximum number of iterations for the convergence
`minThres` : the minimum reduced threshold of objective function
`utiFlag` : whether to calculate utility function

Output parameters:

`pi_sumbest` : the value of objective function for the best KCC experiment
`pi_index` : the cluster assignment matrix in the consensus partition for the best KCC experiment
`pi_converge`: the iterative values of objective function for the best KCC experiment
`pi_utility` : the utility values for the best KCC experiment
`t` : the execution time cost of the whole process for this function

2.16 exMeasure

This function assesses the clustering quality of the results obtained by a clustering algorithm.

Syntax:

```
[Acc, Rn, NMI, VIn, VDn, labelnum, ncluster, cmatrix] = exMeasure(cluster,
true_label)
```

Input parameters:

`cluster` : a $n \times 1$ cluster assignment matrix returned by a clustering algorithm
`true_label` : the true class labels for the data points

Output parameters:

`Acc` : the classification accuracy
`Rn` : the normalized Rand statistic
`NMI` : the normalized mutual information
`VIn` : the normalized Variation of Information
`VDn` : the normalized van Dongen criterion
`labelnum` : the number of unique labels in the groundtruth data

`ncluster` : the number of clusters returned by the algorithm
`cmatrix` : a ncluster × labelnum contingency matrix

Discussion:

This function implements five external validity indices, including classification accuracy [Nguyen and Caruana 2007] (CA), the normalized mutual information [Cover and Thomas 2012] (NMI), the normalized Rand statistic [Rand 1971] (R_n), the normalized van Dongen criterion [Dongen 2000] (VD_n), and the normalized Variation of Information [Cover and Thomas 2012] (VI_n). In implementing the exMeasure function, we utilize a function called bestMap written by [Cai et al. 2005], in which the Hungarian method [Carpaneto and Toth 1980] is employed to resolve the label assignment issue in clustering.

REFERENCES

D. Cai, X. He, and J. Han. 2005. Document Clustering Using Locality Preserving Indexing. *IEEE Trans. Knowl. Data Eng.* 17, 12 (Dec 2005), 1624–1637.

Giorgio Carpaneto and Paolo Toth. 1980. Algorithm 548: Solution of the assignment problem [H]. *ACM Transactions on Mathematical Software (TOMS)* 6, 1 (1980), 104–111.

Thomas M Cover and Joy A Thomas. 2012. *Elements of Information Theory*. John Wiley & Sons.

Stijn Dongen. 2000. *Performance Criteria for Graph Clustering and Markov Cluster Experiments*. Technical Report. Amsterdam, The Netherlands, The Netherlands.

Nam Nguyen and Rich Caruana. 2007. Consensus Clusterings. In *Proceedings of the 2007 Seventh IEEE International Conference on Data Mining (ICDM '07)*. IEEE Computer Society, Washington, DC, USA, 607–612. https://doi.org/10.1109/ICDM.2007.73

William M Rand. 1971. Objective Criteria for the Evaluation of Clustering Methods. *J. Am. Stat. Assoc.* 66, 336 (1971), 846–850.