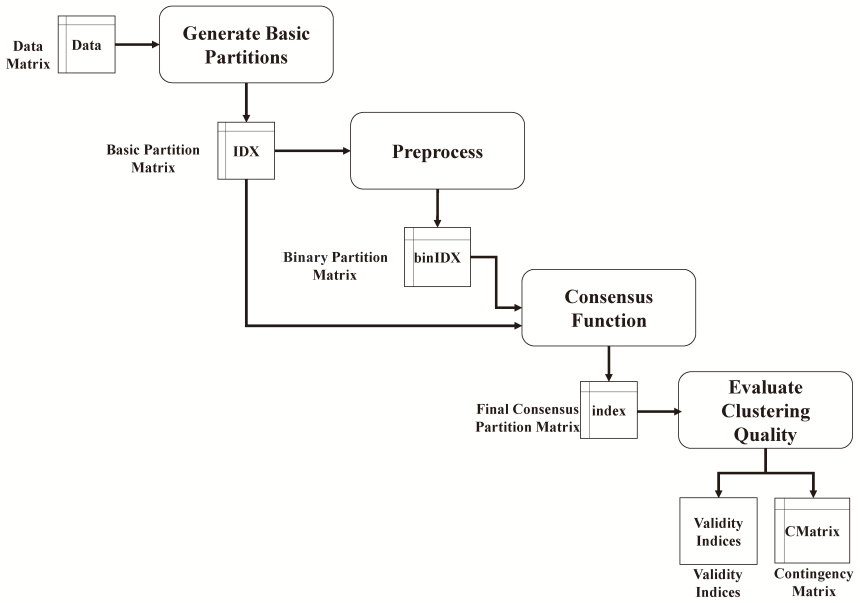

Fig. 1. Diagram of **KCC** package.

Algorithm xxxx: KCC: A MATLAB Package for K-means-based Consensus Clustering

ACM Reference Format:
. 2021. Algorithm xxxx: KCC: A MATLAB Package for K-means-based Consensus Clustering. *ACM Trans. Math. Softw.* ?, ?, Article ? (May 2021), 6 pages. https://doi.org/10.1145/1122445.1122456

1 THE KCC PACKAGE

This section presents a systematic MATLAB **KCC** package that implements the KCC method described in Section **??**, and tackles the issues in Section **??**. Figure 1 illustrates the main functions of the **KCC** package, including basic partitions generation function, consensus clustering preprocessing function, consensus function, and clustering quality evaluation function. We can see from Figure 1 that, 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 $\mathcal{X}^{(b)}$ as introduced in Section **??**, i.e., the binary matrix binIDX. The binary matrix binIDX, along

Author's address:

0098-3500/2021/5-ART? $15.00
https://doi.org/10.1145/1122445.1122456

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.

1.1 Basic partitions generation function

The first step of the KCC framework is to establish a collection of r basic partition results $\Pi = \{\pi_i\}, 1 \leq i \leq r$. In the **KCC** package, the BasicCluster_RFS and BasicCluster_RPS functions are used to generate basic partition results. These two functions employ K-means as a base clustering algorithm, and uses the Random Feature Selection (RFS) strategy and the Random Parameter Selection (RPS) strategy as described in Section **??**, respectively.

For the first basic partition generation function, namely BasicCluster_RFS function, its input argument set consists of Data, r, K, dist and nFeature, which are the input data matrix, the predefined number of basic partitions, the predefined number of clusters in the basic partitions, the distance measure for K-means clustering in p-dimensional space, and the number of randomly selected partial features for RFS, respectively. In particular, Data is an n × p matrix of data, whose rows correspond to n observations, and columns correspond to p features. In RFS strategy, we set the number of clusters across all r basic partitions to a same value K for simplicity. The recommended value for the distance measure is sqEuclidean namely the squared Euclidean distance, and each centroid is the mean of the data points in a cluster. In the RFS process, a nFeature × 1 vector is sampled uniformly at random without replacement from the integers 1 to p, which forms the indices of the selected features.

For the BasicCluster_RPS function, its input argument set consists of Data, r, K, dist and RandKi. Most of the function's input parameters are similar to those of BasicCluster_RFS, except for that RandKi indicates different ways of setting the number of clusters in BPs. Recall that the RPS strategy is to randomly sample different number of cluster for each of the basic partitions. This random sampling strategy can only be activated by setting RandKi to 1 and under the condition of $\sqrt{n} > K$. More specifically, in the RPS strategy, a r × 1 vector is sampled uniformly at random with replacement from the values range from K to $\sqrt{n}$, each entry indicating the number of clusters for the corresponding basic partition. If RandKi is directly 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). If RandKi is set to other values, this function produces r basic partitions with equal number (i.e., K) of clusters.

The output produced from the above two functions is a n × r cluster labels matrix for n data points in r basic partitions, i.e., IDX. In practice, the BasicCluster_RFS function is executed as follows:

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

Similarly, the BasicCluster_RPS function is executed as follows:

```
> IDX = BasicCluster_RPS(Data, r, K, dist, RandKi)
```

It is worth noting that, both BasicCluster_RFS and BasicCluster_RPS are non-deterministic functions. Each call to each of them may yield a different output matrix IDX, due to the random feature sampling process in RFS, the random parameter selection process in RPS, and the random initializations in the K-means algorithm. In addition, users can implement their own basic partition generation strategies under the framework of **KCC** package, or directly import their existing basic partitions for later computation.

1.2 Consensus clustering preprocessing function

With the basic partition matrix produced from Section 1.1, some preprocessing techniques are required to produce the input for the final consensus clustering, as well as some auxiliary variables that can help to save memory and accelerate computations. The **KCC** package offers a Preprocess function for this purpose. The main input argument is basic partition matrix IDX, which can either be obtained using any user-defined base clustering algorithm or the BasicCluster_RFS/BasicCluster_RPS function. One of the main outputs of the Preprocess function is the data matrix binIDX, which is the sparse representation as introduced in Section **??**, and serves as one important input to the final K-means algorithm. The Preprocess function can be called using the following command:

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

Input arguments of the Preprocess function contain IDX, U, w and utilFlag, which are the basic partition matrix, an argument related to the chosen utility function, a weight vector related to the objective function of the consensus clustering, and a flag indicator, respectively. The argument U is a 1×3 parameter cell array. More specifically, the first parameter 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 parameter 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 parameter 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, $p \to \infty$ correspond to the Manhattan distance, the euclidean distance and the chebyshev distance, respectively. The argument w is a $r \times 1$ weight vector, of which each entry indicates the weight value assigned to each basic partition in the objective of consensus clustering. The last input argument utilFlag is a flag indicating whether to calculate utility function during the iterative computation of K-means.

The outputs of the Preprocess function contain Ki, sumKi, binIDX, missFlag, missMatrix, distance, Pvector and weight. The motivation of producing Ki, sumKi, and binIDX is to reduce memory usage, and accelerate the centroid and distance computation, which has been introduced in Section **??**. The output variable missFlag $\in \{0, 1\}$ indicates whether the input IDX matrix contains incomplete basic partitions (IBPs) or not, and a mask matrix missMatrix represent 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 $1 \times r$ row vector calculated from the contingency matrix, i.e., $P_k^{(i)}$ in Equation (**??**), which can later be used in calculating distance and utility functions. The output vector weight is a $r \times 1$ adjusted weight vector adapted for convenient distance calculation in later K-means heuristic.

1.3 Consensus function

Consensus function is the core function of a consensus clustering implementation, which aims at finding the final consensus partition. In the **KCC** package, a KCC function is defined as the consensus function. Several auxiliary functions for distance and utility calculation are also provided here to serve as a clue to potentially extend the **KCC** package to include more utility functions and

distance functions. A RunKCC function is further defined to combine the Preprocess function and the KCC function for easy usage.

1.3.1 KCC function. In the KCC function, we employ the K-means heuristic to conduct a final clustering. More specifically, a KCC utility function is used to construct the objective function of the consensus clustering, while a K-means heuristic with the centroid update and distance calculation phases is utilized to find the consensus partition. The KCC function accepts three main input arguments including the basic partition matrix IDX, the sparse representation matrix binIDX, and the number of clusters in the final consensus partition K. Input arguments that are also the inputs of the Preprocess function contain U, w and utilFlag. Input arguments that are produced by the Preprocess function include Ki, sumKi, binIDX, missFlag, missMatrix, distance, Pvector and weight. Other input arguments are maxIter, minThres, n and r, which are the maximum number of iterations for the convergence, the minimum reduced threshold of objective function, the number of data points, and the number of basic partitions, respectively. Particularly, maxIter is the hard criterion for stopping iteration while minThres is the soft criterion for stopping iteration. The outputs of the KCC function include the final clustering label vector index, the optimal value of the consensus clustering's objective function sumbest, the iterative values of objective function converge, and the final utility function value utility. The KCC function can be called as follows:

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

For convenience of computation, we implement the KCC function under four different conditions: (a) utility calculation is enabled and there exist IBPs in IDX (i.e., utilFlag==1 && missFlag==1); (b) utility calculation is enabled and there are no IBPs in IDX (i.e., utilFlag==1 && missFlag==0); (c) utility calculation is disabled and there exist IBPs in IDX (i.e., utilFlag==0 && missFlag==1); (d) utility calculation is disabled and there are no IBPs in IDX (i.e., utilFlag==0 && missFlag==0). As the KCC algorithm has been described in Section **??**, we only present two main parts of KCC function in the next paragraphs, namely its auxiliary functions for distance calculation and utility calculation while omit other details.

1.3.2 Auxiliary functions for distance calculation. Various distance measures are adapted for the point-to-centroid distance calculation. In **KCC** package, we implement these distance functions with or without missing values (IBPs), including distance_euc, distance_euc_miss, distance_cos, distance_cos_miss, distance_kl, distance_kl_miss, distance_lp, and distance_lp_miss. Their main input arguments are binIDX and the centroid matrix C. The point-to-centroid distance matrix D, acquired as the output of these functions, is used to reassign each data point to its nearest cluster in the K-means heuristic.

- distance_X function: For data sets without missing values (without IBPs), distance_euc, distance_cos, distance_kl and distance_lp function are adopted for distance calculation. We denote each of these functions as distance_X function. Input arguments for these functions consist of U, C, weight, n, r, K, sumKi and binIDX, which are the 1×3 utility parameter cell array, the centroid matrix, a $r \times 1$ adjusted weight vector, the number of data points, the number of basic partitions, the predefined number of clusters, the starting index matrix for basic partitions, and the sparse representation matrix, respectively. The output D is an $n \times K$ point-to-centroid distance matrix. Accordingly, these functions can be called by using the following commands:

```
> D = distance_euc(U, C, weight, n, r, K, sumKi, binIDX)
or
> D = distance_cos(U, C, weight, n, r, K, sumKi, binIDX)
or
> D = distance_kl(U, C, weight, n, r, K, sumKi, binIDX)
or
> D = distance_lp(U, C, weight, n, r, K, sumKi, binIDX)
```

- distance_X_miss function: For data sets with missing values (with IBPs), distance_euc_miss, distance_cos_miss, distance_kl_miss and distance_lp_miss function are employed for distance calculation. We denote each of these functions as distance_X_miss function. While the first eight input arguments of distance_X_miss function (i.e., U, C, weight, n, r, K, sumKi and binIDX) are the same as those of the distance_X function, the additional input variable missMatrix is required to serve as an indicator matrix showing non-zero entries in IDX. The output of these functions is also an $n \times K$ point-to-centroid distance matrix. The commands used to execute these functions are:

```
> D = distance_euc_miss(U, C, weight, n, r, K, sumKi, binIDX, missMatrix)
or
> D = distance_cos_miss(U, C, weight, n, r, K, sumKi, binIDX, missMatrix)
or
> D = distance_kl_miss(U, C, weight, n, r, K, sumKi, binIDX, missMatrix)
or
> D = distance_lp_miss(U, C, weight, n, r, K, sumKi, binIDX, missMatrix)
```

1.3.3 Auxiliary functions for utility calculation. Two functions including UCompute and UCompute_miss are applied for utility calculation on data sets with and without missing values (with and without IBPs), respectively.

- UCompute function. Input arguments for UCompute function consist of an $n \times 1$ cluster assignment matrix index, the 1×3 utility parameter cell array U, the $r \times 1$ adjusted weight parameter vector w, the centroid matrix C, the number of data points n, the number of basic partitions r, the predefined number of clusters K, the starting index matrix for basic partitions sumKi, and the $1 \times r$ vector Pvector. The output is a 2×1 cell array util containing a utility gain and an adjusted utility value. The UCompute function can be executed using the following command:

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

- UCompute_miss function. The input arguments of UCompute_miss function are similar to UCompute function, except for a mask matrix indicating the indices of the non-zero entries M in IDX. The output of UCompute_miss function is a 2×1 cell array util similar to UCompute function. The UCompute_miss function can be executed using the following command:

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

1.3.4 RunKCC function. The **KCC** package also provides a RunKCC function to combine the two functions, i.e., Preprocess and KCC, for finding the consensus partition in one step. In practice, the RunKCC function is executed as follows:

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

It is also worth noting that this function obtains the final consensus partition matrix index by selecting the best result among those generated by rep times of KCC experiments. The output t records the calculation time cost of the whole process.

1.4 Clustering quality evaluation function

After acquiring the final consensus partition, users may need to assess its clustering quality. The **KCC** package provides a function exMeasure for such purpose. Particularly, based on the contingency matrix, this function implements five external validity indices, including classification accuracy [**?**] (CA), the normalized mutual information [**?**] (NMI), the normalized Rand statistic [**?**] (R_n), the normalized van Dongen criterion [**?**] (VD_n), and the normalized Variation of Information [**?**] (VI_n). These indices correspond to the five variables produced by the exMeasure function, i.e., Acc, NMI, Rn, VIn, and VDn. Other outputs include the number of unique labels in the groundtruth data labelnum, the number of clusters returned by the algorithm ncluster, a ncluster $\times$ labelnum contingency matrix. This function requires two input arguments including cluster and true_label. The first argument cluster is a n $\times$ 1 data matrix, of which each row indicates the corresponding data point's cluster label in the final consensus partition, while the second argument true_label is the true class labels for the data points. In implementing the exMeasure function, we utilize a function called bestMap written by **?**], in which the Hungarian method [**?**] is employed to resolve the label assignment issue in clustering. To evaluate the clustering results, the exMeasure function can be called as follows:

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