Paper Review: K-means-based Consensus Clustering: A Unified View [3]

Alberto Becerra Tomé

1 Introduction

In this report, we present the implementation and evaluation of the K-means-based Consensus Clustering algorithm proposed by Wu et al. [3]. The algorithm is a consensus clustering method that combines multiple K-Means clusterings to obtain a single, more stable one. The paper claims that the algorithm outperforms other consensus clustering methods and is robust to noise and outliers.

Objective: Given a finite set of basic partitionings of the same dataset, obtain a single one which agrees with them as much as possible. The paper employs mathematical demonstrations to derive utility functions, enabling the transformation of the consensus clustering problem into a K-Means problem. Subsequently, the 2-phase algorithm employed in K-Means is utilized for solving the transformed problem.

The main concepts we have to take into account are the followings:

Basic Partitioning: It is the result of applying different clustering algorithm to the dataset. In the case of this work, the basic partitionings are obtained by repeatedly applying the K-means algorithm to the dataset. The number of basic partitionings is a parameter of the algorithm denoted with r.

Utility Function: It is a measure of the agreement between the basic partitionings and the consensus partitioning. The utility function is used to guide the search for the consensus partitioning. The final utility (or consensus function) is the average of the utility functions of the basic partitionings.

$$\Gamma(\pi, \pi_i) = \sum_{i=1}^{n} w_i U(\pi, \pi_i), \tag{1}$$

where π is the consensus partitioning, π_i is the basic partitioning and w_i its corresponding weight.

K-Means loss function: Point-to-centroid function used in K-Means to measure the distance between a point and a centroid. There is a whole family of functions that fit K-Means. All of them are composed as follows:

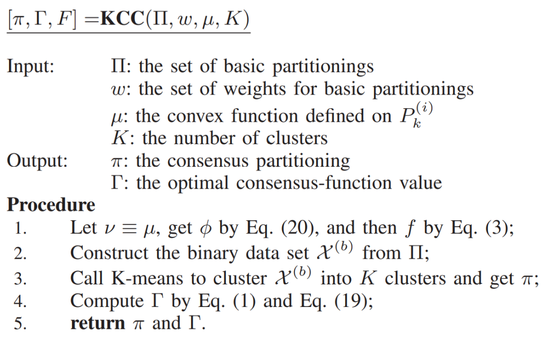
$$f(x, y) = \phi(x) - \phi(y) - (x - y)\nabla(\phi(y)) \tag{2}$$

where ϕ is a differentiable, strictly-convex function.

Binary Dataset ($X^{(b)}$): It is a convenient way to represent the basic partitionings. The binary dataset is a matrix of size $n \times \sum_i K_i$, where n is the number of data points and K_i is the number of clusters of basic partitioning i. It consists in a one-hot-encoding representation of the label of each point for each of the basic partitionings.

Normalized Contingency Matrix: It is a matrix of size $K \times K_i$, where p_{ij} is the proportion of points in cluster i of the consensus partitioning that are in cluster j of the basic partitioning. The normalized contingency matrix is used to compute the utility function.

Creating a correspondence between U and f is the main contribution of the paper. The authors demonstrate that the utility function can be mapped to the K-Means loss function, so that the consensus clustering problem can be solved using the K-Means algorithm. This provides efficiency thanks to the 2-phase heuristic of K-Means and flexibility to use different loss functions.



```
[$\pi, \Gamma, F$] =KCC($\Pi, w, \mu, K$)

Input:    $\Pi$: the set of basic partitionings
          $w$: the set of weights for basic partitionings
          $\mu$: the convex function defined on $P_k^{(i)}$
          $K$: the number of clusters
Output:   $\pi$: the consensus partitioning
          $\Gamma$: the optimal consensus-function value
Procedure
  1.  Let $\nu \equiv \mu$, get $\phi$ by Eq. (20), and then $f$ by Eq. (3);
  2.  Construct the binary data set $\mathcal{X}^{(b)}$ from $\Pi$;
  3.  Call K-means to cluster $\mathcal{X}^{(b)}$ into $K$ clusters and get $\pi$;
  4.  Compute $\Gamma$ by Eq. (1) and Eq. (19);
  5.  return $\pi$ and $\Gamma$.
```

Figure 1: Pseudo-code of the K-means-based Consensus Clustering algorithm.

In order to achieve this, the algorithm proposed by Wu et al. [3] uses a two-phase approach. In the first phase, the centroids are updated using the normalized contingency matrix. In the second phase, the labels are recomputed using the point-to-centroid distance corresponding to the utility function used. The algorithm stops when the sum of these distances (also called inertia) do not change below a tolerance set to 10^{-10} by default, or a maximum number of iterations (default 1000) is reached. However, convergency occurs in less than 15 iterations in most cases. Pseudo-code of the algorithm is shown in Figure 1.

2 Experimental Setup

Replicating the experimental setup of the original paper, datasets from the UCI repository [1] were used.

1. To generate basic partitionings (BPs), we used the K-Means with squared Euclidean distance for UCI data sets. The number of initializations is set to 1, given that the variety in the initializations is associated with an improvement in performance in Consensus Clustering algorithms.
2. We randomized the number of clusters within an interval for each BP within $[2, 2K]$, where K is the number of clusters in consensus partition, set as the number of classes in data.
3. For each dataset, 100 BPs are typically generated for consensus clustering (namely r=100), and the weights of these BPs are exactly the same.
4. In terms of data processing, minimal preprocessing was performed. For the datasets containing missing values, we imputed them with the average of the column. In the case of *wine* dataset, the values of the last attribute were normalized by a scaling factor 100 following the approach of the original authors.

The different utility functions used in the paper were tested and they are shown in Table 1. Also, their normalized versions were tested. The K-Means function corresponding to the normalized version of the utility function is calculated as follows:

$$f_n(x_l^{(b)}, m_k) = \sum_{i=1}^{r} w_i \frac{f(x_l^{(b)}, m_k)}{|\mu(m_k)|} \tag{3}$$

In the second part of the experimental work, the results of the implemented algorithm are compared with some of the most used clustering algorithms. The *Adjusted Rand Index (ARI)* is used to measure the similarity.

The selected algorithms are:

- Hierarchical Clustering (HC): *AgglomerativeClustering* from *sklearn*[2] with its different linkage methods (single, complete, average, ward).
- Partition Methods: *KMeans* from *sklearn* with squared Euclidean distance.
- Soft Clustering: *GaussianMixture* from *sklearn* with their different covariance types (full, tied, diag, spherical).
- Kernel K-Means: *KernelKMeans* from *sklearn* with a *RBF* kernel.

Although Spectral Clustering was initially considered, it was abandoned due to computational inefficiencies, particularly concerning prolonged processing times. It was substitued by Kernel K-Means, which is a similar algorithm but more efficient.

With this selection, we aim to be able to identify the different cluster shapes in the datasets and to compare the results with the ones achieved by the *KCC* algorithm.

For non-deterministic methods (i.e. all of them except HC), the best results out of 10 runs were selected.

For KCC, the point-to-centroid distance corresponding to U_c has been used as it was the most stable and robust to randomness in our experiments.

3 Implementation

The implementation of the algorithm was done in Python. It consists of a main class *ConsensusKMeans* that is initialized with the number of clusters, the number of basic partitionings, the number of clusters within each basic partition, and the utility function type.

The main method of the class is *fit*, which receives the binary dataset, $X^{(b)}$, as input. Centroids are initialized using random pick from $X^{(b)}$ and, iteratively, the algorithm updates the centroids, m_k, using

	$\mu(m_{k,i})$	$U_\mu(\pi, \pi_i)$	$f(x_l^{(b)}, m_k)$
U_c	$\|m_{k,i}\|_2^2 - \left\|P^{(i)}\right\|_2^2$	$\sum_{k=1}^K p_{k+} \left\|P_k^{(i)}\right\|_2^2 - \left\|P^{(i)}\right\|_2^2$	$\sum_{i=1}^r w_i \left\|x_{l,i}^{(b)} - m_{k,i}\right\|_2^2$
U_H	$(-H(m_{k,i})) - \left(-H(P^{(i)})\right)$	$\sum_{k=1}^K p_{k+} \left(-H(P_k^{(i)})\right) - \left(-H(P^{(i)})\right)$	$\sum_{i=1}^r w_i D\left(x_{l,i}^{(b)} \| m_{k,i}\right)$
$U_{\cos}$	$\|m_{k,i}\|_2 - \left\|P^{(i)}\right\|_2$	$\sum_{k=1}^K p_{k+} \left\|P_k^{(i)}\right\|_2 - \left\|P^{(i)}\right\|_2$	$\sum_{i=1}^r w_i \left(1 - \cos\left(x_{l,i}^{(b)}, m_{k,i}\right)\right)$
U_{L_p}	$\|m_{k,i}\|_p - \left\|P^{(i)}\right\|_p$	$\sum_{k=1}^K p_{k+} \left\|P_k^{(i)}\right\|_p - \left\|P^{(i)}\right\|_p$	$\sum_{i=1}^r w_i \left(1 - \frac{\sum_{j=1}^K x_{l,ij}^{(b)} (m_{k,ij})^{p-1}}{\|m_{k,i}\|_p^{p-1}}\right)$

D - KL-divergence; H - Shannon entropy; L_p - L_p norm.

Table 1: Sample KCC Utility Functions

Data Sets	**#Objects**	**#Attributes**	**#Classes**
breast	699	9	2
ecoli	332	7	6
iris	150	4	3
pendigits	10992	16	10
satimage†	6435	36	6
dermatology	358	33	6
wine‡	178	13	3

Table 2: UCI Datasets Information

†: In the original paper there were 4435 entries.

‡: the values of the last attribute were normalized by a scaling factor 100.

$$m_{k,i} = \left(\frac{p_{k1}^{(i)}}{p_{k+}}, \ldots, \frac{p_{kK_i}^{(i)}}{p_{k+}}\right), \qquad (4)$$

where p_{k+} is the sum of the k-th row of the normalized contingency matrix.

The labels are then recomputed using as point-to-centroid distance the f corresponding to the utility function used (see Table 1). The algorithm stops when the sum of these distances (also called inertia) do not change below a tolerance set to 10^{-10} by default, or a maximum number of iterations (default 1000) is reached. However, convergency occurs in less than 15 iterations in most cases.

The f function is calculated using **vectorized operations**. For euclidean, cosine and p-distances, the use of *scipy* library and its *cdist* function were key to achieve good performance. In the case of the KL-divergence, the function was implemented from scratch using numpy operations and array shape broadcasting properties. Previous attempts calculating the point-to-centroid function using loops were not efficient, taking more than 10x the time of the vectorized version, which given the number of iterations, was not acceptable. The version of f for the normalized utility function was implemented in a similar way but weighting each point-to-centroid distance with the inverse of μ.

The algorithm was tested using a testbed of datasets from the UCI repository. The results of the clustering were evaluated using the Adjusted Rand Index (ARI).

4 Results

4.1 Algorithm Reproducibility

The results of the algorithm were compared with the original results of the paper. The ARI was calculated for each dataset and each utility function. The results are shown in Tables 4 and 5. Visual representation of these tables can be seen in Figures 2 and 3 for each dataset and utility function.

The results of the new implementation of the algorithm are consistent with those from the original one. We can observe that there are significant differences in *breast* for some of the utility functions. All of the results are higher in the new implementation, and the same independently of the utility funciton used. KCC relies on transforming the consensus clustering problem into a K-means problem. Any changes in the implementation's logic, such as initialization, random seed settings, or convergence criteria, can affect the resulting clusters and, consequently, the consensus partitioning. We have checked that, for some random seeds, this doesn't happen. However, we have considered to include this case, showing the potential convergency issues. If the new implementation uses different parameter settings or initialization methods, it can lead to significant differences in results. However, for the rest of the datasets, the results are consistent with the original ones except some differences possibly due to stochasticity and numerical errors.

When analyzing the results by utility function, we can see that the results are consistent with the

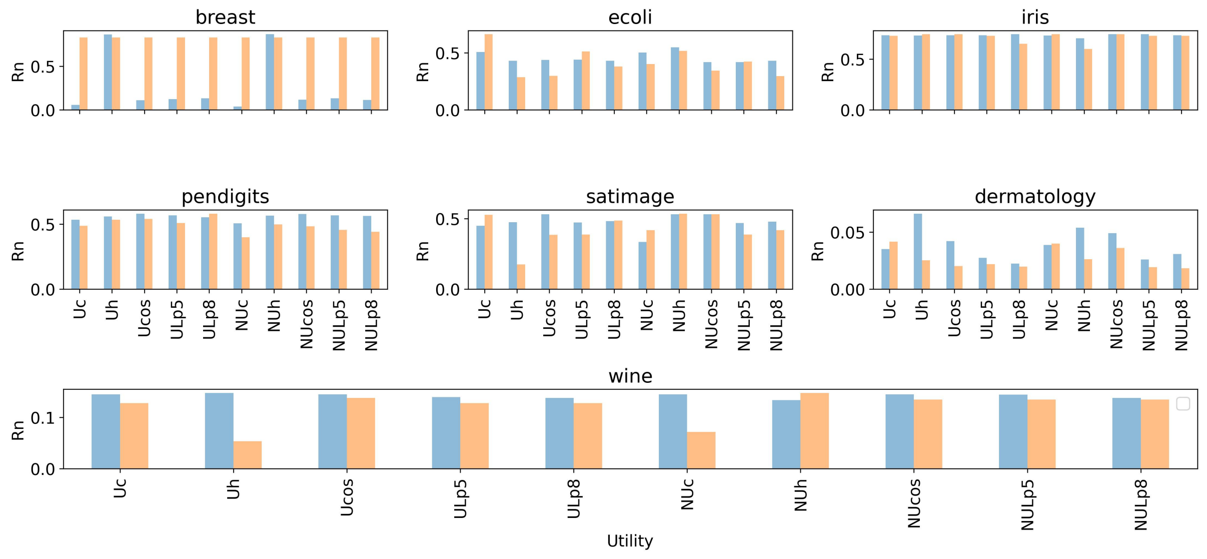


Figure 2: Comparison of performance between datasets (blue for original results). Each one with its scale.

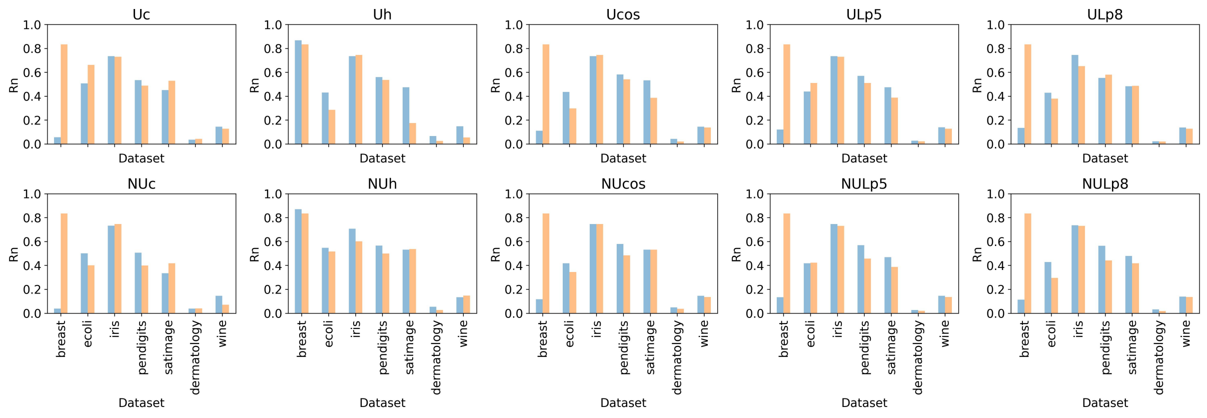


Figure 3: Comparison of performance by metrics (blue for original results). All of them in the same scale.

original ones. In Figure 3, in some cases as for *satimage*, *dermatology* and *wine*, there are some small discrepancies between original and implemented U_h. In Figure 4 no significative differences are observed in the average ARI for each utility function. The distribution of the ARI for each dataset is also consistent with the original results.

The training time for the algorithm was also measured for each dataset and utility function. The comparison is shown in Table 3. From the results we can see that, in general, the training time is consistent with the original results. There is no significant difference between the original and the implemented algorithm in terms of training time. In the case of *pendigits* and *satimage*, which are the biggest datasets, the training time is lower than the original results. This can be due to the different hardware used in the experiments.

4.2 Comparison with other clustering algorithms

The results of the implemented algorithm were compared with other clustering algorithms. The ARI was calculated for each dataset and each clustering algorithm, as well as the training times. The results are shown in Tables 6 and 7 and visualizations in Figure 5.

As we can observe, the implemented algorithm

	breast	ecoli	iris	pendigits	satimage	dermatology	wine
Original (U_c)	1.95	1.40	0.33	81.19	32.47	1.26	0.56
Results (U_c)	0.69	2.31	0.52	73.61	25.15	1.96	0.45

Table 3: Training times values for each dataset.

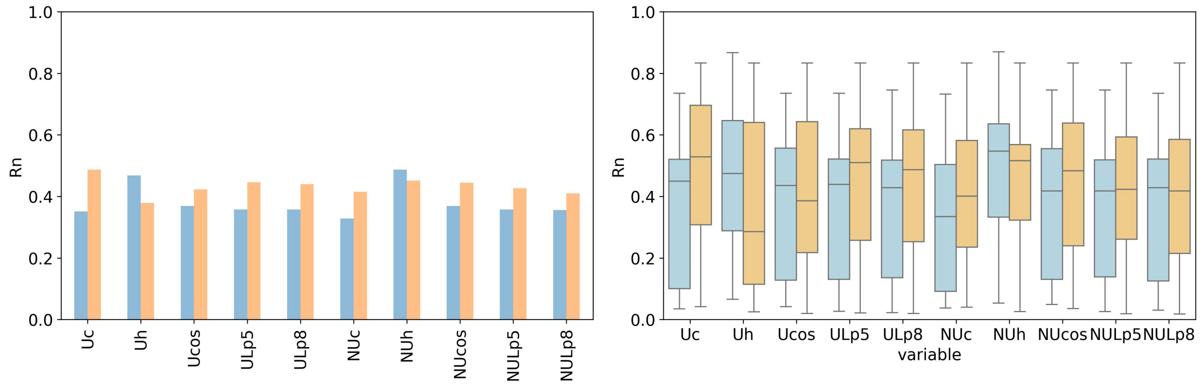


Figure 4: Comparison of statistical performance by metrics (blue for original results) across all the datasets. Average comparison in the left. Distribution for each dataset in the right.

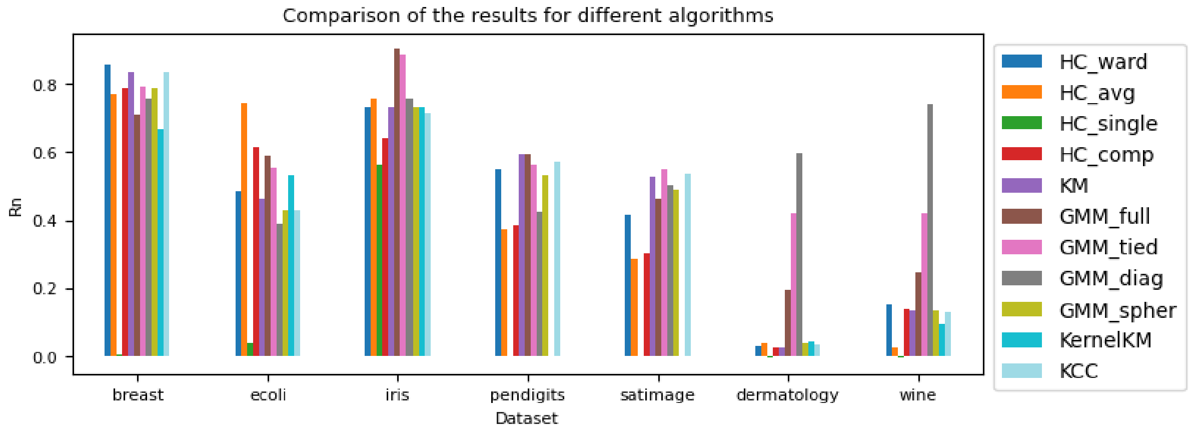


Figure 5: Comparison of performance (ARI) for different clustering algorithms. Our implementation of KCC in the rightmost column.

performance is comparable to some of the most used clustering algorithms in the literature. For the rest of the datasets, the behavior is similar to the rest of them, highlighting the poor performance in *dermatology* and *wine* datasets in comparison with GMMs, that are clearly superior.

Something important to note is that the training time for the implemented algorithm is significantly higher than the rest of the algorithms. Most of these algorithms are really optimized to run in low level languages, so it's difficult to compete with them in terms of performance. Apart from that, given the nature of the algorithm, it wasn't expected to be the fastest one.

As a summary, the implemented algorithm is able to compete with some of the most used clustering algorithms in the literature used in this work. The main drawback is the training time, which is significantly higher than the rest of the algorithms. This is something that should be improved in future work.

5 Conclusions

In conclusion, this work successfully implemented and evaluated the K-means-based Consensus Clustering algorithm proposed by Wu et al. [3]. The algorithm, designed to obtain a stable consensus clustering from multiple K-Means clusterings, was thoroughly examined and tested using datasets from the UCI repository.

The primary objective of the algorithm is to find a consensus partitioning that aligns with a set of basic partitionings obtained through repeated application of the K-means algorithm. The paper establishes a mathematical framework, introducing utility functions and demonstrating their correspondence with K-Means loss functions. This mapping allows the consensus clustering problem to be efficiently solved using the K-Means algorithm.

The experimental setup closely followed the methodology outlined in the original paper, utilizing UCI datasets and generating basic partitionings through repeated K-means applications. Various utility functions were tested, both in their original and normalized versions, with the consensus clustering algorithm performing consistently across different datasets.

The implementation of the algorithm in Python demonstrated efficiency and accuracy. The results were evaluated using the Adjusted Rand Index (ARI), and the algorithm's performance was compared with the original paper's results. Overall, the implemented algorithm produced comparable results, maintaining consistency across different utility functions and datasets.

The comparison with other clustering algorithms demonstrated the competitiveness of the implemented algorithm. The ARI results were comparable to those of other clustering algorithms, with the exception of the *dermatology* dataset, where the implemented algorithm underperformed. The training times for the implemented algorithm were significantly higher than the rest, a limitation that should be addressed in further developments.

However, KCC has several notable weaknesses. The most significant is its high training time, which can be significantly longer than other clustering algorithms. This could be due to inefficiencies in the algorithm or the implementation in high-level languages like Python. Additionally, since KCC relies on K-means, it inherits some limitations, such as sensitivity to initial centroids and an assumption of spherical clusters, which can impact clustering performance, especially with varying cluster shapes and densities.

Convergence issues are another concern; the algorithm's convergence might vary depending on initializations, requiring careful tuning of parameters. The complexity of mapping utility functions to K-means loss functions might also complicate the implementation process for those without a deep mathematical background. Furthermore, the stochastic nature of K-means can lead to variability in results, affecting consistency.

To improve KCC, focus on optimization to reduce training time, potentially by using lower-level languages or optimizing Python-based implementations with libraries like NumPy and SciPy. Robust initialization techniques could mitigate sensitivity to initial centroids, and integrating with other clustering methods could overcome K-means' inherent limitations. Adding more utility functions may also offer greater flexibility and robustness.

In summary, while KCC's consensus approach and solid theoretical grounding make it a promising clustering method, high training times and inherited limitations from K-means suggest that further optimization and enhancements are needed to improve its applicability and performance. In summary, the implemented K-means-based Consensus Clustering algorithm demonstrates robustness and reliability for diverse datasets. The alignment with the original paper's results, along with the flexibility to accommodate different utility functions, underscores the algorithm's utility in practical clustering applications.

References

[1] Dheeru Dua and Casey Graff. Uci machine learning repository, 2017.

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[3] Junjie Wu, Hongfu Liu, Hui Xiong, Jie Cao, and Jian Chen. K-means-based consensus clustering: A unified view. *IEEE Transactions on Knowledge and Data Engineering*, 27(1):155–169, 2015.

	U_c		U_H		$U_{\cos}$		U_{L5}		U_{L8}	
	Original	Results	Original	Results	Original	Results	Original	Results	Original	Results
breast	0.0556	0.8337	0.8673	0.8337	0.1111	0.8337	0.1212	0.8337	0.1333	0.8337
ecoli	0.5065	0.6613	0.4296	0.2857	0.4359	0.2973	0.4393	0.5098	0.4284	0.2948
iris	0.7352	0.7302	0.7338	0.7455	0.7352	0.7455	0.7352	0.6013	0.7455	0.7302
pendigits	0.5347	0.4882	0.5596	0.5348	0.5814	0.5404	0.5692	0.4834	0.5527	0.4411
satimage	0.4501	0.5282	0.4743	0.1753	0.5322	0.3857	0.4738	0.5310	0.4834	0.4181
dermatology	0.0352	0.0416	0.0661	0.0254	0.0421	0.0203	0.0274	0.0361	0.0223	0.0192
wine	0.1448	0.1278	0.1476	0.0537	0.1448	0.1379	0.1397	0.1348	0.1379	0.1348

Table 4: KCC Clustering Results (by Rn) (Us)

	NU_c		NU_H		$NU_{\cos}$		NU_{L5}		NU_{L8}	
	Original	Results	Original	Results	Original	Results	Original	Results	Original	Results
breast	0.0380	0.8337	0.8694	0.8337	0.1173	0.8337	0.1329	0.8337	0.1126	0.8337
ecoli	0.5012	0.3788	0.5470	0.4005	0.4179	0.5165	0.4174	0.3440	0.4281	0.2948
iris	0.7325	0.7455	0.7069	0.6013	0.7455	0.7455	0.7455	0.7302	0.7352	0.7302
pendigits	0.5060	0.5803	0.5652	0.3995	0.5789	0.4991	0.5684	0.4834	0.5639	0.4569
satimage	0.3349	0.4181	0.5323	0.5361	0.5318	0.5309	0.4691	0.3873	0.4797	0.4181
dermatology	0.0386	0.0399	0.0537	0.0262	0.0490	0.0361	0.0259	0.0192	0.0309	0.0183
wine	0.1448	0.0714	0.1336	0.1475	0.1449	0.1348	0.1447	0.1348	0.1379	0.1348

Table 5: KCC Clustering Results (by Rn) (NUs)

Dataset	**HC ward**	**HC avg**	**HC single**	**HC comp**	**KM**	**GMM full**	**GMM tied**	**GMM diag**	**GMM spher**	**KernelKM**	**KCC**
breast	**0.856**	0.771	0.003	0.786	0.834	0.709	0.792	0.758	0.788	0.666	0.834
ecoli	0.486	**0.745**	0.040	0.616	0.466	0.590	0.554	0.391	0.429	0.533	0.430
iris	0.731	0.759	0.564	0.642	0.730	**0.904**	0.886	0.759	0.730	0.730	0.716
pen-based	0.552	0.372	0.000	0.387	0.595	0.591	0.562	0.424	0.531	0.000	**0.573**
statlog	0.415	0.288	0.000	0.302	0.529	0.464	**0.550**	0.501	0.491	0.000	0.537
dermatology	0.031	0.038	-0.005	0.025	0.026	0.194	0.418	**0.598**	0.039	0.042	0.035
wine	0.151	0.027	-0.004	0.137	0.134	0.245	0.419	**0.741**	0.133	0.096	0.130

Table 6: Clustering Methods Comparison in terms of ARI

Dataset	**HC ward**	**HC avg**	**HC single**	**HC comp**	**KM**	**GMM full**	**GMM tied**	**GMM diag**	**GMM spher**	**KernelKM**	**KCC**
breast	0.011	0.006	0.003	0.005	0.060	0.106	0.071	0.056	0.057	0.149	**0.686**
ecoli	0.001	0.002	0.001	0.002	0.059	0.213	0.091	0.072	0.058	0.031	**2.306**
iris	0.000	0.001	0.001	0.000	0.054	0.126	0.052	0.037	0.039	0.018	**0.523**
pen-based	2.167	2.301	0.556	2.388	0.104	14.728	2.546	3.133	0.838	41.091	**73.608**
statlog	0.927	0.984	0.300	0.950	0.115	8.640	2.187	1.021	0.660	9.488	**25.154**
dermatology	0.003	0.002	0.003	0.002	0.068	0.677	0.693	0.083	0.077	0.061	**1.961**
wine	0.001	0.000	0.001	0.001	0.055	0.167	0.101	0.051	0.061	0.024	**0.450**

Table 7: Clustering Methods Comparison in terms of Timing (seconds)