UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

MASTER IN ARTIFICIAL INTELLIGENCE

UNSUPERVISED LEARNING

COURSEWORK B: COOPERATIVE BISECTING K-MEANS

Author:

Daniel Hinjos García

June 2021

Contents

1 Introduction and Motivation

Clustering is a data mining method that allows to form meaningful groups of objects, so that the elements within the same group o cluster are similar among them and have certain dissimilarities with respect to the ones outside the cluster. This allows us to gain insight and simplify our reasoning considerably. As a result, it is a technique widely used in many fields. Its increasing relevance has led through the years to the development or refinement of several clustering algorithms with different principles, performance and complexity. Among them, we can briefly introduce the popular K-Means and Bisecting K-Means, key components of this assignment.

K-Means (KM) algorithm consists, at first, in selecting k objects of the dataset randomly as initial centroids for the desired clusters. Then, the data points are assigned to the cluster with the closest centroid, the centroids of the clusters are recomputed as the mean of all objects within each cluster and a new assignation step is carried out. This is repeated until some termination criterion is fulfilled (partitional clustering).

On the other hand, Bisecting K-Means is a variant of the previous one. Starting from a unique cluster containing all data points, it is based on splitting one cluster in two (using K-Means) in each iteration, until reaching the specified number of clusters (k) or some other termination condition (hierarchical clustering). At each step, one cluster is selected to be bisected, and the selection criteria may vary among the different implementations.

Bisecting K-Means has proven to perform better than the classic K-Means, as it is less susceptible to initialization problems and the partitions are made to preserve homogeneity (it carries out several trials for each bisection and saves the best). However, in some cases, through the iterations, some fractions or clusters of the dataset might be left behind with no further considerations or way to re-cluster it in future levels of the hierarchical tree. As a result, a refinement process over the final clusters is necessary many times to overcome this issue. This situation, far away from ideal, suggests the necessity of a method that is able to directly deal with these problems. This is the main motivation for the algorithm that will be presented below.

Combining clustering methods is a good starting point to face performance problems. In special, according to the level of coordination used to approach this methodology, the clustering algorithms are said to cooperate on the intermediate level or at the end-result level. Combining clustering with end-result cooperation has traditionally guaranteed an improvement in clustering quality and performance. For example, often the hybrid BKM-KM with end-result approach is used, which consists in refining the clusters by using the centroids of the resulting clusters of BKM as the initial centroids for classical KM. This is successful because KM is guaranteed to find a clustering that represents the local optima, whereas in BKM we are using the KM locally, resulting in a final set of clusters that does not fall into that problem [1].

Based on the idea of intermediate cooperation, the Cooperative Bisecting Algorithm (CBKM) is presented in [1]. The aim of this assignment is to understand, summarize, implement, reproduce and test this algorithm with different artificial and real datasets.

2 Cooperative Bisecting K-Means (CBKM)

Cooperative Bisecting K-Means is presented as an algorithm based on the principles of BKM, including the idea of splitting clusters at each level of a generated hierarchical tree (from l=2...k). However, it includes the notion of intermediate cooperation between KM and BKM. Specifically, by using cooperative contingency and cooperative merging matrices at each level, resulting from the intersections of KM and BKM and the subclusters merging cohesiveness factors, respectively, CBKM eventually obtains better clustering results than the ones provided by both methods separately.

At each level of the tree, CBKM consists in three phases: the global clustering phase, the cooperative clustering phase and the merging clustering phase. This last phase, at the same time, comprises three steps.

Algorithm: Cooperative Bisecting *k*-means: CBKM (X,ITER,ζ, *k*,*SM*, *NumBins*)

Input: the dataset X, ITER for the bisecting step, homogeneity criterion ζ, number of clusters k, similarity matrix *SM*, and number of bins in the histogram, *NumBins*.

Output: set of k clusters, $S=\{S_0,S_2,..,S_{k-1}\}$

Begin

For number of clusters l* =2 *to k

Phase 1: Global Clustering

$S^{cooperative}(l)=\{\}$, Synchronously generate the two sets $S^{KM}(l)$ and $S^{BKM}(l)$

Phase 2: Cooperative Clustering

Using the two sets $S^{KM}(l)$ and $S^{BKM}(l)$, the $CCM(S^{KM}(l),S^{BKM}(l))$ is constructed and a new set *Sb* of n_{sb} disjoint sub-clusters is generated .

Phase 3: Merging

*Step*1: for each sub-cluster $Sb_i \in Sb$, *Build-Histogram* (Sb_i, *NumBins*, *SM*)

*Step*2: Using the sub-cluster's histograms, each element in the *CMM* matrix, $mcf(Sb_i,Sb_j)$, is calculated. Initially, $S^{cooperative}(l)=Sb$

*Step*3: ***Repeat***

Merge two sub-clusters from the set $S^{cooperative}(l)$ with the highest similarity merging value (*mcf*) in *CMM*; reduce the number of cooperative clusters by one and update the *CMM*.

Until (number of cooperative clusters=l)

*Step*3: Replace the Bisecting clustering $S^{BKM}(l)$ with the $S^{cooperative}(l)$.

End

Return the final set of k desired clusters $S=S^{cooperative}(k)$

End

Figure 1: CBKM Pseudocode

2.1 Phase 1: Global Clustering

Firstly, K-Means and Bisecting K-Means are performed, resulting in two sets containing l clusters for that level. In the first iteration (i.e. *l=2*), both methods will be fed with the whole dataset, whereas for the following iterations, although KM remains the same, BKM will receive as input the *l-1* cooperative clusters resulting from the third phase of the previous level. This process will be detailed later.

2.2 Phase 2: Cooperative Clustering

In this phase, a cooperative contingency matrix (CCM) is built using both sets of clusters generated in the previous phase. In special, this matrix comprises the number of objects of each KM cluster that belongs to each BKM cluster, providing a mapping that identifies the disjoint subclusters that comprehend a intersection between both KM and BKM clusterings.

The dimension of this matrix is l *x* l and thus the number of subclusters identified (n_{sb}) can be, as maximum, l^2.

2.3 Phase 3: Merging Clustering

This is the most extense phase. In it, some steps are followed so that the set of n_{sb} subclusters is transformed into l clusters with better clustering quality than the ones provided by the KM and the BKM separately.

For that matter, a new matrix with dimension $n_{sb}(n_{sb}-1)/2$ is introduced: the cooperative merging matrix (CMM). In this new bi-dimensional data structure, each element represents a merging cohesiveness factor (mcf) between two of the subclusters identified, which will be discussed later. This factor depends on the so-called similarity histograms of both subclusters.

Similarity Histograms

These similarity histograms are individual for each subcluster and comprise a concise statistical representation of the set of pair-wise similarities in the collection of its objects (i.e. the elements within each subcluster). The similarity metric used is the cosine coefficient and thus the histogram is built in the interval [-1, 1], considering a *BinSize* that is calculated based on a number of bins *NumBins* passed as a input parameter, such as $BinSize = 2/NumBins$. This way, for a fixed bin size, each *binid* bin contains the count of similarities included in the interval [(*binid* - (*NumBins*/2))**BinSize*, (*binid* - (*NumBins*/2))**BinSize*+*BinSize*].

With these considerations, the distributions of the similarities of the histogram of a typical cluster is similar to a normal distribution, while a more coherent and ideal cluster will have a histogram where most of similarities fall in high or maximum values/bins. On the other hand, a histogram where most of its similarities are in minimum values will result from a loose cluster.

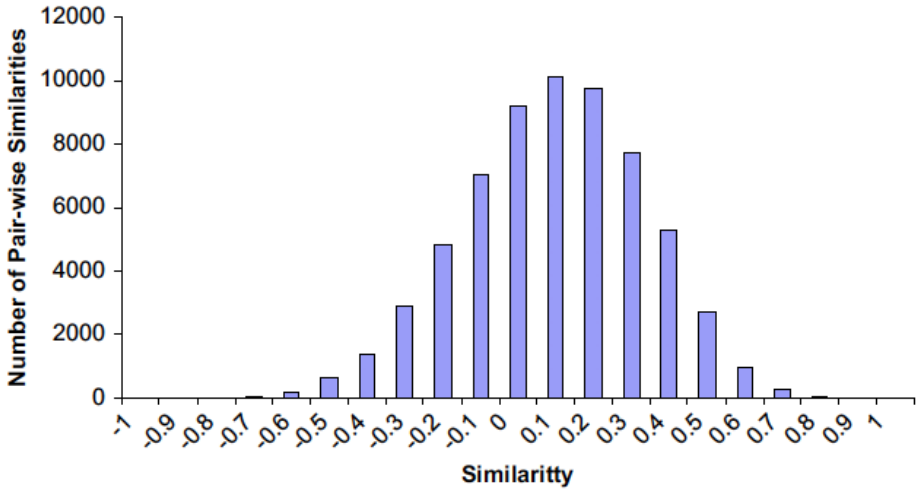

Figure 2: Similarity histogram of subcluster with NumBins = 20

This way, a similarity histogram is created and stored for each subcluster being considered in the phase (**Step 1**).

Merging Cohesiveness Factor

Afterwards, a merging cohesiveness factor (mcf) is assigned to each pair of subclusters present in the cooperative merging matrix (CMM) mentioned before. This factor represents the quality of merging the subclusters into a new cluster, and thus it is very relevant for our purposes. The higher this value, the more cohesive the new generated cluster will be. The mcf is computed considering the coherency of merging the similarity histograms of both subclusters into a new one, which first has to be constructed.

Assuming the same number of bins for all histograms, this new histogram can be conformed by adding the corresponding counts of each bin from both merged histograms and, additionally, adding the count of the new pair-wise similarities resulting from comparing the elements of both subclusters among them that fall into each bin, as the following formula suggests:

$$H_{ij}(bin) = \begin{pmatrix} (H_i(bin) + H_j(bin) + |Sim(\boldsymbol{x},\boldsymbol{y})|), \forall \boldsymbol{x} \in Sb_i, \boldsymbol{y} \in Sb_j, \\ bin = 0, 1, \ldots, NumBins - 1 \\ \text{Such that} \{((bin - (NumBins/2)) * BinSize) < Sim(\boldsymbol{x},\boldsymbol{y}) \\ \leqslant ((bin - (NumBins/2)) * BinSize + BinSize)\} \end{pmatrix}$$

After creating this new histogram, the merging cohesiveness factor between both subclusters is computed considering the ratio of the count of similarities weighted by the bin similarity above a certain bin threshold (manual hyper-parameter) to the total count of similarities in the new merged histogram, such as:

$$mcf(Sb_i, Sb_j) = \frac{\sum_{bin=binThreshold}^{numBins-1} (((bin * binSize) - 1 + (binSize/2)) * H_{ij}(bin))}{n_{Sim}(Sb_i, Sb_j)}$$

Being $n_{sim}(Sb_i, Sb_j)$ the number of additional similarities of merging the two subclusters together, given by the expression $[(|Sb_i| + |Sb_j|) * (|Sb_i| + |Sb_j| - 1)/2]$, where $|Sb_i|$ and $|Sb_j|$ are the number of objects in the subclusters Sb_i and Sb_j, respectively.

Finally, after all the merging cohesiveness factors have been computed and the CMM is completed (**Step 2**), a iterative process starts, which consists in merging the two subclusters with the highest *mcf* into a new one and update the CMM accordingly. This (**Step 3**) is repeated until the number of subclusters is equal to the number of clusters of the corresponding level in the hierarchical tree (l).

When the whole level iteration has finished, the resulting cooperative set of l clusters is used to feed the BKM of the first phase in the next iteration. This cooperative algorithm is repeated for each number of clusters $l = 2... \ k$ until the final number of clusters k is obtained. This way, BKM receives a set of homogeneous clusters to be bisected at each level of the hierarchical tree, which, beside the intermediate cooperation with KM, are refined, providing a clustering with better quality through the iterations.

3 Implementation Details

The implementation of this algorithm itself is mainly covered by a class called CBKM, which has been created following the sklearn conventions. We can visualize the class diagram to have a wider vision of the variables and methods involved.

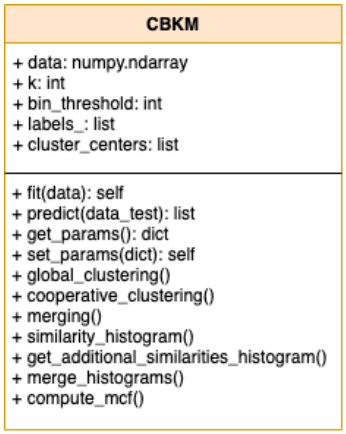

Figure 3: CBKM Class

The main methods are fit and predict, although some other methods relevant in sklearn such as *get_params* and *set_params* have been implemented as well. Note that other typical methods such as $transform$, $fit\_predict$ or $fit\_transform$ have not been considered relevant for the case.

- **fit(data)**: When calling this method, the whole CBKM algorithm is executed over the dataset specified and the clusters are generated. Specifically, the labels of all data points and the centroids of all clusters generated are stored in two global variables of the class called $labels\_$ and $cluster_centers\_$, respectively. From this method, the rest of methods of the class are eventually called.
- **predict(data_test)**: This method provides the label(s) of the closest cluster (closest centroid) for each sample in *data_test*.

The algorithm itself has been implemented following the steps and guidelines indicated in [1], which can be observed in the pseudocode displayed in the previous section. Moreover, it is relevant to indicate that, whereas the sklearn implementation of KM algorithm has been directly used, BKM has been implemented from scratch. For that matter, another class BKM has been constructed in the same way than the previous one, with similar methods (fit and predict) and variables.

One of the difficulties in implementing BKM is to decide how to choose which cluster to split among all the available choices (note that as the algorithm proceeds it creates new clusters, increasing the number of available clusters to choose from). It was considered that an accurate criteria for making this decision would be to choose the cluster with the **highest mean of squared error (MSE)**, so that the total MSE of the set would be reduced as the algorithm progresses. As mentioned beforee, the real interest in this algorithm lies in the fact that it is less susceptible to problems derived from random initialization of centroids because it incorporates a process of several "bisecting" trials in order to determine which will be the best split for each cluster to be divided (and therefore, which will be the best random centroids for it) [2]. This number of trials has been set to 10 by default.

Leaving aside BKM, we can focus on CBKM implementation. Specifically, in the first phase both KM and BKM are executed (Phase 1). Once we have both sets of clusters, the cooperative contingency matrix (CCM) is constructed with the intersections of all clusters (Phase 2) and it is displayed in the following way:

```
(l = 3)
Cooperative Contingency Matrix (CCM)
     S0   S1   S2
S0  454    0    0
S1    0  126    3
S2   12    1  103
```

Figure 4: CCM constructed in the second level with *l=3*

From this matrix, we obtain the subclusters to examine. At this point, the phase 3 starts and the similarity histograms must be computed. **A typical number of bins chosen for the construction of similarity histograms is 20, resulting in a *Bin-Size* of 0.1**. With these parameters and following the pseudocode illustrated in the paper, the histograms are built for each subcluster, following the paper indications. We can appreciate in the next Figure two histograms of two subclusters and the histogram resulting from merging both of them in a future step.

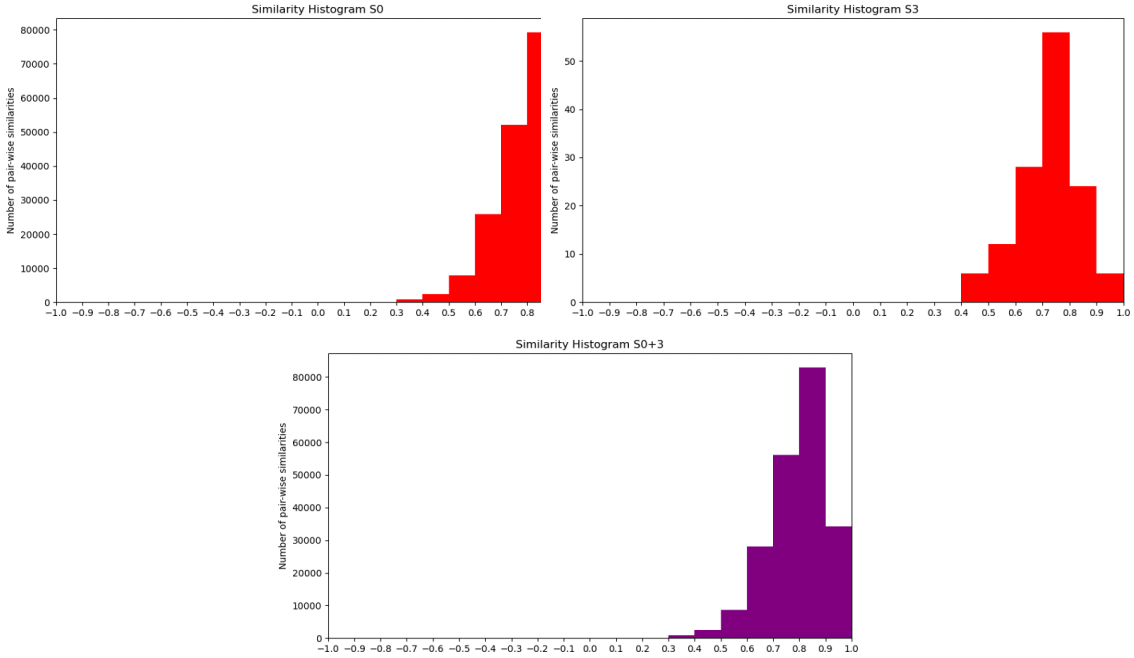

Figure 5: In red, histograms of two individual subclusters; in purple, histogram of the cluster resulting from their merging[1]

Afterwards, the merging cohesiveness factor is calculated for each pair of subclusters by means of their histograms, and all of these values are stored in the symmetrical cooperative merging matrix (CMM). For the *mcf* computation, each new histogram is computed and the formula described is applied as indicated in the previous section. **The *binthreshold* chosen has been set to 12, as it is the bin corresponding to similarities of 0.2**. In [1], a range of similarity threshold comprising 0.2 was used for all datasets considered, so this value was considered appropriate enough.

It is at this point where the iterative sub-process starts in which the pair of subclusters with highest *mcf* are merged and the CMM is updated with the *mcf* values of the new resulting cluster with respect to each of the remaining subclusters in the matrix (which means computing the internal histogram of the resulting merging cluster and constructing a new histogram as indicated in the previous section for each lasting subcluster in the CMM). This is repeated until the number of clusters in the matrix is l.

[1]Note that this merging histogram is not the same as the histogram constructed to compute the merging cohesiveness factor of the subclusters, but this is just a representation of the internal similarities of the resulting merged cluster

As clarification, we can observe the merging phase of level *l=3* for the CCM of Figure 4. Notice how the size of the CMM is decreased as the merging takes place.

```
Cooperative Merging Matrix (CMM)
          Sb0       Sb1       Sb2       Sb3       Sb4       Sb5
Sb0 -1.000000  1.158605  1.391545  1.364051  1.397897  1.179225
Sb1  1.158605 -1.000000  1.462405  1.359881  1.485291  1.092867
Sb2  1.391545  1.462405 -1.000000  1.070295  1.031746  1.415770
Sb3  1.364051  1.359881  1.070295 -1.000000  1.186813  1.301064
Sb4  1.397897  1.485291  1.031746  1.186813 -1.000000  1.440823
Sb5  1.179225  1.092867  1.415770  1.301064  1.440823 -1.000000

Sub-clusters merged: Sb1 and Sb4
Cooperative Merging Matrix (CMM)
            Sb0       Sb2       Sb3       Sb5     Sb1+4
Sb0   -1.000000  1.391545  1.364051  1.179225  1.157481
Sb2    1.391545 -1.000000  1.070295  1.415770  1.461362
Sb3    1.364051  1.070295 -1.000000  1.301064  1.359677
Sb5    1.179225  1.415770  1.301064 -1.000000  1.092698
Sb1+4  1.157481  1.461362  1.359677  1.092698 -1.000000

Sub-clusters merged: Sb2 and Sb1+4
Cooperative Merging Matrix (CMM)
              Sb0       Sb3       Sb5   Sb2+1+4
Sb0     -1.000000  1.364051  1.179225  1.154241
Sb3      1.364051 -1.000000  1.301064  1.360646
Sb5      1.179225  1.301064 -1.000000  1.093642
Sb2+1+4  1.154241  1.360646  1.093642 -1.000000

Sub-clusters merged: Sb0 and Sb3
Cooperative Merging Matrix (CMM)
              Sb5   Sb2+1+4     Sb0+3
Sb5     -1.000000  1.093642  1.180289
Sb2+1+4  1.093642 -1.000000  1.155248
Sb0+3    1.180289  1.155248 -1.000000

Merge phase finished with l=3 sub-clusters
```

Figure 6: CMM through the merge phase

When this condition is fulfilled (number of cooperative subclusters is l), the iterative sub-process finishes and the next level iteration starts for the next value of l (if $l!=k$) and these resulting cooperative clusters will be used to feed the BKM. By repeating this process, we eventually obtain refined clusters with higher quality and more homogeneity by means of intermediate cooperation between different methods.

Note that the rest of implementation details not mentioned in this section are strictly followed as indicated in [1] . Note also that some other files with several auxiliar methods and the implementation of the specific metrics used are part of this project as well, not only the principal algorithm classes.

4 Experiments and Results

Before diving into results, it is necessary to provide some insight about the experimental procedure. By trying to replicate the methodology of [1], the Cooperative Bisecting K-Means (CBKM) implemented is compared with K-Means (KM), Bisecting K-Means (BKM) and Single Linkage Agglomerative Clustering (SL). KM and BKM implementations have been mentioned before and SL is executed from the sklearn method AgglomerativeClustering with linkage='single'.

As KM, BKM and CBKM are subject to the implicit randomness (sklearn's KM number of iterations has been set to 1), several trials of each one of them must be carried out for a specific k value and a dataset. Note that the k values chosen are fixed for each dataset, as finding the most optimal value of k for each case is not believed to be within the scope of this project. As SL algorithm doesn't depend on any randomization, it only generates one solution and thus is not necessary to perform more than one episode.

This way, for each of these algorithms and each dataset, 10 runs have been executed and the mean and standard deviation values for all metrics have been stored. Moreover, as to illustrate the real significance of the results obtained, a statistical t-test analysis is carried out for the mean values of each measure to reveal significant differences between KM-CBKM and BKM-CBKM. Specifically, we use a critical t_value of 2.024 at degree of freedom 38 within a 95% confidence interval, as it has been specified in the paper. The null hypothesis H_0, "no significant difference", is accepted if the $t\_value < t\_critical$ and rejected otherwise.

The execution of each dataset will result in an output .txt file containing the information about the different metrics and the results of the statistical analysis for the different algorithms.

4.1 Evaluation Metrics

In order to evaluate the quality of the clusters generated in diverse ways, several external and internal metrics are proposed in [1]. Specifically, the external quality measures suggested are F-measure, entropy, purity and normalized mutual information (NMI), whereas separation index (SI) is also presented as internal metric. Moreover, the total execution time is also considered.

As we want to reproduce the methodology of the paper, the different metrics have been implemented following its specific guidelines and formulas. For that reason, only a superficial description of each one of them is going to be presented in this report:

- **F-measure**: combining the ideas of precision and recall of each cluster S_j with respect to each classification class R_i, the F-measure of a class is defined as:

$$F(R_i) = \max_j \frac{2 * precision(R_i, S_j) * recall(R_i, S_j)}{precision(R_i, S_j) + recall(R_i, S_j)}$$

where the cluster S_j with the highest F-measure is considered to be the cluster that is mapped to that class. The overall F-masure for the whole k clusters is the weighted average of the F-measure for each class, such as:

$$F\text{-}measure = \frac{\sum_{i=0}^{k-1} (|R_i| \times F(R_i))}{\sum_{i=0}^{k-1} |R_i|}$$

The higher the F-measure, the better the clustering because the mapping from the clusters generated to the real classes is more accurate.

- **Entropy**: this metric measures the homogeneity of the cluster. For every cluster S_j, its entropy is computed by means of the probabilities that a member of the cluster belongs to each class R_i, such as:

$$E(S_j) = -\sum_{i=0}^{k-1} pr_{ij} \log(pr_{ij}), \quad j = 0, 1, \ldots, k-1$$

The overall entropy for the set of k clusters is the sum of the entropies for each cluster weighted by the cluster size:

$$entropy = \sum_{j=0}^{k-1} \left(\frac{|S_j|}{n} \right) \times E(S_j)$$

The lower the entropy, the more similar will be the data points within each cluster. Note that, after obtaining the probabilities, *scipy.stats* library has been used to speed up the computation.

- **Purity**: the purity of a cluster S_j is the ratio of the number of objects that belong to the dominant class of the cluster to the total amount of objects in the same cluster. The overall purity of a set of k clusters is the sum of the purities of all clusters divided by the total number of objects in the dataset, such as:

$$purity(k) = \frac{1}{|n|} \sum_{j=0}^{k-1} \max_{i=0,1\ldots k-1, j \neq i} (L_{ij})$$

where L_{ij} is the number of objects from class R_i in cluster S_j. Note that the higher the purity, the more suitable is the partition of the data points into the clusters generated.

- **Normalized Mutual Information**: in information theory, Mutual Information is a measurement of the mutual dependence between two random variables. NMI is a normalization of MI, so that the results are scaled between 0 (no mutual information at all) and 1 (perfect correlation). For this metric, the sklearn's method $sklearn.metrics.normalized\_mutual\_info\_score$ has been directly used.

- **Separation Index**: by considering cluster centroids c_i, this measure is an indicator of the dissimilarity between clusters, as well as between the objects within each cluster with respect to their corresponding centroid. Formally, it is the ratio of average within-cluster variance to the minimum pair-wise dissimilarity measured with cosine coefficient:

$$SI(k) = \frac{\sum_{i=1}^{k} \sum_{\forall \boldsymbol{x}_j \in S_i} 1 - CosSim(\boldsymbol{x}_j, c_i)}{n * \min_{r,s=1,\ldots,k, r \neq s} \{1 - CosSim(c_r, c_s)\}}$$

  The smallest this index, the more correctly separated are the clusters.

4.2 Preprocessing

Right after loading the dataset in the main file, a small preprocessing step is carried out in order to handle some issues. Typically, a personalized preprocessing pipeline would be created for each considered dataset according to its necessities, but as preprocessing is not the relevant part of this assignment, only some generals adjustment have been made.

Specifically, the numerical attributes have been normalized to the range [0,1] with sklearn's *MinMaxScaler* method. The possible missing values have been replaced by the mode of the attribute for categorical attributes and the mean of the attribute for numerical ones. For nominal attributes, a decoding step into 'utf8' and a discretization using sklearn's *LabelEncoder* method has been applied as well.

4.3 Datasets

In order to test the performance of the implemented algorithm, some experiments have been carried out with five different datasets that will be introduced in this section, two artificial and three real, attending to the requirements of the assignment: DS1, DS2, Heart-C, Breast Cancer Wisconsing and CMC. Note that no test nor validation sets have been generated for these tests, as it has not been considered necessary for the objectives.

4.3.1 Artificial Datasets

The artificial datasets have been generated with the sklearn method *make_blobs*. Specifically, the first one (DS1) is composed of 811 samples, 3 centroids and 2 features; the second one (DS2) is formed by 2530 samples, 6 centroids and 2 features as well. These values are not arbitrary, but are set as to replicate the conditions of the artificial datasets used in the paper.

In the following tables, the different results for the datasets are displayed. Consider that the values shown are the mean values of the 10 iterations (except for SL) and the values inside the parenthesis are the standard deviations.

	KM	BKM	SL	CBKM
F-Measure ↑	0.9988 (0)	0.9988 (0)	0.7778	0.9988 (0)
Entropy ↓	0.0074 (0)	0.0074 (0)	0.4193	0.0074 (0)
Purity ↑	0.9988 (0)	0.9988 (0)	0.6683	0.9988 (0)
NMI ↑	0.9926 (0)	0.9926 (0)	0.7313	0.9926 (0)
SI ↓	0.0371 (0)	0.0371 (0)	0.4867	0.0371 (0)
Time ↓	9.3774 (0.0181)	9.4313 (0.0245)	10.109	171.5113 (0.3559)

Table 1: Performance of KM, BKM, SL and CBKM for DS1 dataset with k=3

	KM	BKM
F-Measure ↑	t = 0.0, H0: Accepted	t = 0.0, H0: Accepted
Entropy ↓	t = 0.0, H0: Accepted	t = 0.0, H0: Accepted
Purity ↑	t = 0.0, H0: Accepted	t = 0.0, H0: Accepted
NMI ↑	t = nan, H0: Accepted	t = nan, H0: Accepted
SI ↓	t = 0.0, H0: Accepted	t = 0.0, H0: Accepted
Time ↓	t = 1365.0151, H0: Rejected	t = 1363.1066, H0: Rejected

Table 2: Results of T-Test for DS1 dataset

For DS1 dataset, k has been set to 3, as that is its number of centroids specified in its generation. Note how the mean values for all metrics except the computation time is exactly the same for CBKM, KM and BKM, along with a standard deviation of 0. These values, although representative of a very good performance close to optimal, raise certain suspicious. However, if we take into consideration that the dataset has a very simple nature (recall that it only has two attributes!), it's not crazy to think that all three algorithms perform in such way for all iterations executed. The results provided by SL algorithm are pretty worse than the rest. It is also remarkable the time cost, as CBKM time is 18 times slower than KM, BKM's or even SL's.

Regarding the statistical analysis, this situation leads to the acceptance of all null hypothesis except for the time one, with a t_value of 0 or nan. This outcome is reasonable, as we are statistically comparing values that are exactly the same.

	KM	BKM	SL	CBKM
F-Measure ↑	0.9478 (0.0582)	0.9678 (0)	0.6381	0.9595 (0.0434)
Entropy ↓	0.0714 (0.0394)	0.0693 (0)	0.4357	0.068 (0.0298)
Purity ↑	0.948 (0.0578)	0.968 (0)	0.5	0.9597 (0.0431)
NMI ↑	0.934 (0.0287)	0.9309 (0)	0.7193	0.9348 (0.022)
SI ↓	12.5525 (13.4785)	9.242 (0)	37.2827	7.9068 (2.4438)
Time ↓	139.1121 (2.0001)	140.7594 (0.2528)	152.2455	7715.0948 (5368.06)

Table 3: Performance of KM, BKM, SL and CBKM for DS2 dataset with k=6

	KM	BKM
F-Measure ↑	t = 0.4849, H0: Accepted	t = -0.5738, H0: Accepted
Entropy ↓	t = -0.2051, H0: Accepted	t = -0.1319, H0: Accepted
Purity ↑	t = 0.4869, H0: Accepted	t = -0.5804, H0: Accepted
NMI ↑	t = 0.0655, H0: Accepted	t = 0.5303, H0: Accepted
SI ↓	t = -1.0174, H0: Accepted	t = -1.6391, H0: Accepted
Time ↓	t = 4.2339, H0: Rejected	t = 4.233, H0: Rejected

Table 4: Results of T-Test for DS2 dataset

For DS2 dataset, k parameter has been set to 6, respecting its specified centroids and thus the values used in the original paper as well. This dataset is bigger than the previous one, but we must keep in mind that the number of attributes is the same (2), which emphasizes again the simplicity of the dataset.

It can be appreciated how the mean values, although similar, are not the same for the different algorithms. CBKM performance overcomes KM, and it's very balanced with the values of BKM, providing slightly better results for some of the metrics (specially Separation Index). However, as it is later verified by the statistical analysis, there is no significant differences (all null hypothesis are accepted except for the time) between the performances of the three algorithms, although it might seem different. SL performance is, again, not relevant. On the other hand, the computation times in this scenario are more differentiated. CBKM seems to be much slower than the rest for big datasets.

It is interesting to remark that the standard deviation of BKM metrics is 0. BKM itself performs KM for a fixed number of times in order to partition the chosen cluster, so this situation is reasonable. However, it also puts on the table the fact that the dataset is very "easy" to clusterize, as the best partition returned by KM seems to be always the same.

Overall, the results for the artificial datasets are very high, but our CBKM doesn't contribute significantly. However, the values seem skewed by the fact of the datasets' cleanness and lack of difficult information to clusterize. Therefore, it is necessary to verify the performance of the different algorithms in real data situations before reaching any conclusion.

4.3.2 Real Datasets

Three real datasets have been obtained from the UCI Machine Learning Repository, so as to cover three cases considered interesting to explore, regarding their size: a small (Heart-C), a medium (Breast Cancer) and a larger (CMC) dataset. They have all been loaded and processed in the .arff format, but the option to load data in the .csv format has been implemented as well.

The smallest dataset, **Heart-C**, is a medical dataset that comprises data from different patients with the aim of predicting the presence of heart disease in the individual. It has 303 samples and 14 mixed attributes (6 of them numerical and 8 of them categorical,

including the class labels). There are 2 different classes in which the instances are categorized: >50_1 and <50. Moreover, missing values are present in 0.17 % of the whole dataset.

The medium dataset, **Breast Cancer Wisconsing**, has been included in this study because a bigger version of it is indeed used in the original experiments. Features are extracted from images fine needle aspirates (FNA) of breast mass, describing characteristics of the cell nuclei present in the image. It has 699 samples and 9 numerical attributes plus class labels. The instances are categorized into two classes: 'benign' and 'malign'. Also, it is relevant to remark that it has 0.25% of missing values.

On the other hand, **CMC** has also been considered relevant for the purposes, as it is a larger and mixed dataset. It contains 1473 instances and 9 attributes plus class, from which 7 of them are numerical and 2 of them categorical. It doesn't have any missing values, which is a positive point.

	KM	**BKM**	**SL**	**CBKM**
F-Measure ↑	0.4809 (0.027)	0.5674 (0.0011)	0.6667	0.4842 (0.0237)
Entropy ↓	0.4805 (0.0084)	0.4876 (0.0008)	0.4924	0.4842 (0.0095)
Purity ↑	0.602 (0.0132)	0.5855 (0.0034)	0.5479	0.5809 (0.0106)
NMI ↑	0.0238 (0.0108)	0.0183 (0.0013)	0.0175	0.0186 (0.0134)
SI ↓	0.965 (0.1379)	0.8702 (0.1252)	1.5461	8.5715 (2.8094)
Time ↓	1.8794 (0.0039)	1.8996 (0.0044)	1.8144	74.1232 (6.3842)

Table 5: Performance of KM, BKM, SL and CBKM for Heart-C dataset with k=4

	KM	**BKM**
F-Measure ↑	t = 1.0924, H0: Accepted	t = -9.2641, H0: Accepted
Entropy ↓	t = 0.8787, H0: Accepted	t = -1.0858, H0: Accepted
Purity ↑	t = -3.734, H0: Accepted	t = -1.2418, H0: Accepted
NMI ↑	t = -0.907, H0: Accepted	t = 0.067, H0: Accepted
SI ↓	t = 8.1128, H0: Rejected	t = 8.2156, H0: Rejected
Time ↓	t = 33.9483, H0: Rejected	t = 33.9388, H0: Rejected

Table 6: Results of T-Test for Heart-C dataset

For Heart-C dataset, k parameter has been arbitrarily set to 4, although the dataset has only two classes. The results are not really good and very far away from those obtained for the artificial dataset. The values for most of the metrics are, in fact, quite bad. However, this reality reveals the true possible performance of these datasets in real-world data, so it's a very interesting and necessary approach.

From the three principal algorithms, BKM seems to slightly outperform the rest. However, as it can be verified again, most of the null hypothesis of the t-test are accepted, so there's no really significative difference among the metrics, except for the Separation Index and the time, for which CBKM values are actually the worse. Nevertheless, the computation time for CBKM has not the biggest difference with the rest of algorithms found up until this point.

In this case, it is relevant to consider that SL performance is very competitive, overcoming even all of the algorithms regarding F-measure.

	KM	BKM	SL	CBKM
F-Measure ↑	0.8127 (0.0332)	0.8276 (0.0013)	0.6952	0.7675 (0.062)
Entropy ↓	0.097 (0.0117)	0.1226 (0.0001)	0.4613	0.1982 (0.0406)
Purity ↑	0.9652 (0.004)	0.9571 (0)	0.6595	0.9166 (0.025)
NMI ↑	0.5997 (0.0345)	0.5809 (0)	0.0136	0.4596 (0.0811)
SI ↓	1.8663 (0.269)	2.2415 (0.0361)	1.105	2.0017 (0.1137)
Time ↓	6.9939 (0.1229)	7.0409 (0.0188)	6.8445	362.8144 56.01)

Table 7: Performance of KM, BKM, SL and CBKM for Breast Cancer Wisconsin dataset with k=4

	KM	BKM
F-Measure ↑	t = -1.9251, H0: Accepted	t = -2.9029, H0: Accepted
Entropy ↓	t = 7.1831, H0: Rejected	t = 5.5878, H0: Rejected
Purity ↑	t = -5.7679, H0: Accepted	t = -4.8652, H0: Accepted
NMI ↑	t = -4.7714, H0: Accepted	t = -4.4884, H0: Accepted
SI ↓	t = 1.3903, H0: Accepted	t = -6.0306, H0: Accepted
Time ↓	t = 19.0584, H0: Rejected	t = 19.0559, H0: Rejected

Table 8: Results of T-Test for Breast Cancer dataset

For Breast Cancer dataset, the k parameter has been selected as 4 as well, based on empirical knowledge and the indications of the paper for the bigger version of this algorithm.

This time the performance is pretty great for the principal algorithms, not so much for SL agglomerative clustering. However, it is true that, overall, CBKM returns worse values than KM and BKM and a time complexity much higher.

Most null hypothesis are accepted, so in this case the sensation of worse performance of CBKM is not really grounded, except for the case of entropy and time, which are both rejected for KM and BKM, meaning that there is indeed a significant difference between CBKM values and the rest. Nevertheless, it is relevant to consider that, although entropy null hypothesis is rejected, the real difference between the smallest entropy (KM) and CBKM entropy value is only of 0.1. Therefore, if a conclusion can be extracted from this case is that CBKM performance is kind of static, in the sense that it does not outperform nor falls much behind the rest of the principal algorithms, except for the time factor.

	KM	BKM	SL	CBKM
F-Measure ↑	0.403(0.0011)	0.4428(0)	0.5159	0.4449(0.0034)
Entropy ↓	0.9464 (0.001)	0.9477 (0)	0.9732	0.951 (0.0028)
Purity ↑	0.443 (0.0045)	0.4318 (0)	0.4325	0.4412 (0.0054)
NMI ↑	0.0322 (0.001)	0.0302 (0)	0.0024	0.0276 (0.0023)
SI ↓	6.2842 (0.2033)	11.1496 (0)	0.4	23.1408 (9.0308)
Time ↓	56.7354 (0.0942)	56.622 (0.0571)	55.1777	980.3281 (68.0148)

Table 9: Performance of KM, BKM, SL and CBKM for DS2 dataset with k=3

	KM	BKM
F-Measure ↑	t = 34.861, H0: Rejected	t = 1.838, H0: Accepted
Entropy ↓	t = 4.5232, H0: Rejected	t = 3.4678, H0: Rejected
Purity ↑	t = -0.8046, H0: Accepted	t = 5.1907, H0: Rejected
NMI ↑	t = -5.4774, H0: Accepted	t = -3.4156, H0: Accepted
SI ↓	t = 5.5982, H0: Rejected	t = 3.9834, H0: Rejected
Time ↓	t = 40.7378, H0: Rejected	t = 40.7429, H0: Rejected

Table 10: Results of T-Test for CMC dataset

For the CMC dataset, k parameter has been set to an arbitrary value of 3, despite comprising two classification classes as well.

The results perceived are not really good for the principal algorithms. Four out of six null hypothesis are rejected, which at first instance is a great sign! Even if this affects negatively to our CBKM regarding mostly time and Separation Index mostly, its F-measure turns out to be the greatest significant one, meaning that the mapping of the clusters generated to the classes is the most accurate for our algorithm (although the value itself is pretty low). For the rest of metrics, such as entropy, purity and NMI, either if their null hypothesis are rejected or accepted, the differences with respect to KM and BKM are not that big, although they perjudicates our algorithm, overall. On the other hand, SL algorithm outstands, overcoming the rest in all metrics.

It is relevant to remark that even if the time costs of CBKM is much higher than the rest, its ratio is lower than the one of the large artificial dataset. It is true that that dataset had more samples, but it also had many less attributes.

5 Discussion

After these experiments, several points can be discussed. The first thing to highlight is that the implemented version of CBKM is very time consuming in all cases. However, this is normal considering that, apart from executing KM and BKM itself in its first phase, CBKM carries out many more calculations with the different subclusters examined in each step, so it's reasonable that the time complexity is significantly higher than the rest in all datasets tested. However, it must be taken into account the efficiency of the sklearn implementations used for the comparisons, which are unmatched within the AI community. Moreover, it is also true that the temporal cost is much higher than the one illustrated in the original paper, but it must be considered as well the abilities and expertise of the authors, which are much superior to mine.

Many times, this slowness of the implemented version of CBKM makes its use not worth it, because the performance differences with other less costly clustering algorithms are not that higher nor significant enough. In some cases, even a simple SL agglomerative clustering is much more efficient than the rest.

Anyhow, it is acknowledged that there might be some flaws in the implemented CBKM, both in the time and in the clustering quality capability domains. The original paper was not very clear in some mathematical and procedural parts (especially in some formulas or specific steps of the algorithm), and thus these issues might be derived from personal decisions and interpretations.

Regarding general behavior, it seems that all algorithms tested provide better performance for artificial and simpler datasets. The performance of the algorithms doesn't really seem to depend entirely on the artificial or real nature of the dataset, but in many more factors. However, the only case in which we have obtained a significantly better results with CBKM for one of the metrics at least (F-measure) has been with the real and large dataset, CMC. This fact is very encouraging, because in the original paper the greatest performance improvements by CBKM take place in real and huge datasets, with the greatest cases being datasets of up to 18800 samples and 91000 attributes.

Therefore, we could set up the hypothesis that CBKM algorithm itself works better in real and bigger datasets with a simple nature, as both the results of the original paper and the results of this project point out so. Hence, our implemented algorithm could still be efficient and provide relevant performance. However, more tests should be carried out in order to corroborate this and reach solid conclusions about the real quality of the personal implementation, because the particular limitations of hardware and time haven't allowed me to execute tests with much larger datasets. However, if doing future executions, time and memory factors must be taken into account, as the algorithm might be very time-consuming for some datasets.

Nevertheless, the evaluation methodology followed is very robust, as both several iterations to overcome the randomness problems of the algorithms and a statistical t-test analysis have been used. Thus, even though we can't extract definitive conclusions with these limited amount of results, it can be said that everything declared in this section is pretty grounded.

6 Conclusions

As a summarization, at first sight the implemented version of CBKM does not seem to provide significant results with respect to the rest of the algorithms with which it's compared. Moreover, the time complexity is much higher than the other's, decreasing its viability as a real option. As aa result, the possible flaws and implementation errors are recognized.

However, better but slight results regarding significance contributions start to be visible with real and large datasets, which is matched with the performance and improvements of the original paper. Due to the lack of resources and time, it has not been possible to test the algorithm over larger datasets to corroborate the final effectiveness of the implementation. Further tests with different datasets and values of k parameter, along with a refinement of the code to reduce the time complexity and balance the trade-off between cost and performance, is proposed as future work.

Nevertheless, a state-of-the-art unsupervised learning algorithm has been studied, understood and interpreted. Then, it has been implemented following sklearn conventions, and its performance and properties have been compared with respect to some other well-known clustering methods, in order to extract conclusions and gain insight. Therefore, the objectives of the project have been fulfilled and the work done has been successful.

I feel that this assignment has provided me a wider vision about the real functioning of classical machine learning algorithms and techniques. In addition, it has been impressive to experiment first hand with the generation, intersection, organization and merging of clusters. It has also been inspiring to be able to observe how the implemented algorithm overcomes the weaknesss of the traditional BKM that leaves parts of the dataset behind at each level of the hierarchical tree with no way to reconsider it again, using intermediate cooperation with KM, matrices to handle the creation and merging of subclusters and many other thoughtful techniques through the process. Regarding the paper, it has also been interesting to challenge my analytical, comprehension and research abilities. Overall, a very captivating coursework and an invigorating perspective of unsupervised learning!

References

[1] R. Kashef and M. Kamel, "Enhanced bisecting k-means clusteringusingintermediate-cooperation."

[2] "Chapter 8: Cluster analysis: Basic concepts and algorithms."

Appendix: How to run the code

This project has been developed using Python v3.6 as programming language and PyCharm as IDE. In order to execute the project, simply open the project with PyCharm and run "main.py". You will be greeted with a menu that will provide you the opportunity of selecting between several options:

- "1": Processing DS1 dataset.
- "2": Processing DS2 dataset.
- "3": Processing Heart-C dataset.
- "4": Processing Breast Cancer dataset.
- "5": Processing CMC dataset.
- "6": Process another dataset.
- "7": Exit.

Before anything else, note that the PyCharm project is composed by five .py files and two folders: "datasets" and "results". In order for the adjusted execution to work properly, the dataset files should be located inside a folder called "datasets" within the root path. The folder called "results" should also exist previously to running.

Once the project is properly set, by selecting "1", "2", "3", "4" or "5" you will start the processing of the studied datasets with CBKM, KM, BKM and SL. By selecting option number "4" you will be able to process another dataset of your choice. For that matter, you will have to provide the filename of your dataset, and verify that said file is located inside a folder "datasets" as just mentioned. In addition, only .csv or .arff files will be properly loaded. Last option is just exiting the execution.

Select a studied dataset and, if it's real, some basic relevant information about it will be displayed in a first instance. Afterwards, the CBKM will start executing for the specified number of iterations and with a fixed k value that can be changed within the main file. After the n runs of CBKM, n runs of KM and BKM will be executed as well. Finally, only one execution of SL will be performed. When the whole process is finished, the relevant information about the metrics for the different algorithms will be stored in the corresponding .txt file inside the folder "results".

Moreover, if you want only the relevant classes and methods of the algorithm in order to integrate it in your library, you must extract the files "CBKM.py", "BKM.py" and "utils.py" from the project. The first two contain the classes for CBKM and BKM, whereas the third one comprises relevant methods used by the main and by both classes. Therefore, be careful to include the important methods of "utils.py", which can be easily integrated into the classes if desired. Mostly all relevant imports are in "utils.py" as well, so make sure to correctly reference them.

Moreover, in a separate way, if you want to test the algorithm's performance with the metrics introduced, you should also consider all the methods of the "metrics.py" file.