WORK 1

CLUSTERING EXERCISE

Group members:

SANTIAGO BERNAL, RODRIGO ARIAS, ANA COCHO

Implementation Details

For our implementation, we decided to use the following datasets: wine, vehicles and segment. The code can be executed by running the command: python clustering.py [filepath], where the file path is the path of the .arff file relative to the current position. For example: **python clustering.py datasets/wine.arff**

K-means:

The K-means algorithm is implemented by following the steps of the slides.

```
1.Decide on a value for k
2.Select k random instances {s1, s2,… sk} as seeds.
3.(Decide the class membership)
For each instance x_i:
    Assign x_i to the cluster c_j such that d(xi, sj) is minimal.
4.(Update the seeds to the centroid of each cluster)
For each cluster c_j
    s_j= μ(c_j)
5.(Until clustering converges or other stopping criterion):
If none of the N instances changed membership, exit
Otherwise, go to step 3
```

The stopping criteria used is when the number of modified classes reaches zero, or when the number of iterations is greater than a threshold. The function is called do_kmeans, and has the parameters:

- **X, Xn**: The numeric and nominal part of the input dataset.
- **Iterations**: The maximum number of iterations. By default 500.
- **N_clusters**: The number of clusters or classes.
- **Gamma**: The coefficient γ used to weight the nominal part of the dataset, when computing the distances. By default is 1.1

In addition, if we found that two clusters are very close together, we move one to a new random position. Also if one of the clusters has no classes assigned, a new position is randomly assigned.

The formula to compute the distance between a data-point and a cluster is based on the k-prototypes algorithm, with p numerical classes, and m-p nominal classes. The δ function computes the number of class matches, and the distance from the numeric part is the Euclidean distance:

$$d(X,\ Y) = \sum_{j=1}^{p}(x_j - y_j)^2 + \gamma \sum_{j=p+1}^{m} \delta(x_j, y_j)$$

The classes returned by the function are just indexes in the range of the number of classes. But they can be wrongly matched, like the classes A and B can be swapped. So after the classification, we swap the indexes to maximize the matches with the original classification.

K-harmonic means:

The k-harmonic means introduces a variation in the way the centers of the clusters are computed. The harmonic mean is used to select the new position. The formula to compute the centers is

$$\alpha_i = \frac{1}{(\sum_{l=1}^{K} \frac{1}{d_{i,l}^{\ 2}})^2}, \quad q_{i,k} = \frac{\alpha_i}{d_{i,k}^{\ 3}}, \quad q_i = \sum_{k=1}^{K} q_{i,k}, \quad p_{i,k} = \frac{q_{i,k}}{q_i}, \quad m_k = \sum_{i=1}^{N} p_{i,k} x_i.$$

Where m_k is the new position of the cluster k. The algorithm is very similar to the original k-means, but the centroids are updated differently. In order to stop the algorithm, a measure of the performance is used, simply as the mean euclidean distance between points and clusters. If the mean performance difference over the last 10 iterations is lower than a threshold the computation is stopped.

After some tests, it seem that the implementation is wrong, as the centroids don't converge to the centers of the data, instead they go away.

Fuzzy C-means:

For the fuzzy clustering algorithms we choose Fuzzy C-means. For this, we created a "do_fuzzyCMeans" method with the parameters:

- X: which is the input dataset, with no class and scaled. This parameter is calculated by using the input dataset (using scipy's arff.loadarff method), and removing the classes we then scale it using the sklearn scale method.
- C: which is the number of clusters to be calculated and to be used as the "c" of fuzzy C-Means. For the calculations of the optimal C, we tested on each dataset using a different number and evaluated the results.
- Fuzzyness: the degree of fuzzyness of the values to be calculated (m). We used 2 as the default value, to avoid a crisp fuzzyness.
- Iterations: which is the amount of total maximum iterations. For this we chose 100 so that it would not take too much trying to find the clusters.
- Termination threshold: (or epsilon) which is the threshold to be compared to the termination measure to determine if we finished clustering the data. For this we chose 0.01 as the default value.

Once the parameters were set, we implemented the following algorithm:

1. Calculate the first centroid vector by creating a random matrix that followed a normal distribution.
2. Calculate the membership matrix which specifies the percentage of membership of each sample to a cluster using Formula 1
3. Re calculate the centroids using the new matrix. For this we used Formula 2.
4. After calculating both elements, we would check if the termination threshold has been met by using Formula 3, and if it was, then we finished the processing, if not we would go back to step 2.
5. This algorithm was repeated until the threshold had been reached or until we reached the total amount of iterations allowed.

The Formulas were:

$$U_{ik} = \left(\sum_{j=1}^{c} \left(\frac{\| x_k - v_i \|}{\| x_k - v_j \|} \right)^{2/(m-1)} \right)^{-1} \quad \forall i, \forall k$$

Formula 1

$$v_i = \frac{\sum_{k=1}^{n} (U_{ik})^m x_k}{\sum_{k=1}^{n} (U_{ik})^m}$$

Formula 2

$$\| V_t - V_{t-1} \|$$

Formula 3

To apply these formulas we considered that: m = degree of fuzzyness; c = number of clusters to calculate; n = number of samples in the dataset; x = input dataset; V = centroid vector; and U = the membership matrix.

For the optimization of the value of C for Fuzzy C-Means, we tested different values to discover which ones worked best, using as maximum value the number of classes plus one:

- Segments dataset: we noticed how lower values like 2 gave us an output with an error of around 71%, and as we increased we saw it slowly decreasing the error, but incrementing the computational time required, we found that at around 6 or 7 the error reaches its lowest value, but the time it takes still increases in any C value after that, so the optimal C would be one of those two values, we chose 7 since it has a lower avg time.
- Wine dataset: the wine dataset got the best results when using C=3, having the least amount of error % (close to 0) and still not taking much time to

execute. Every C value higher than 3, would start taking more time and the output would have more error.

- Vehicle dataset: For this dataset, the error always remained around the same when changing the value of C, but would increase in the computational time it would take as the value of C kept growing. So the optimal C for this case would be 1.

All of the datasets with the results for each C can be found on the following table:

Dataset	C	Error	Avg Time
Segments	2	71.43%	3311ms
	3	57.19	8217.798ms
	4	45.11%	38639.562ms
	5	40.48%	32615.354ms
	6	37.66%	93946.296ms
	7	36.67%	80850.299ms
	8	36.82%	125696.869ms
Wine	1	71.11%	16.324ms
	2	40.89%	218.642ms
	3	4.30%	590.001ms
	4	28.43%	1770.205ms
Vehicles	1	74.33%	73.013ms
	2	73.43%	932.411ms
	3	74.31%	3008.378ms
	4	74.12%	9747.261ms
	5	79.31%	20042.438ms

Evaluation Details

For the evaluation we chose to use: adjusted rand index (ARI) and the Davies-Bouldin index (DB). We also compare the error percentage by comparing the input with the computed output, the time it takes to execute the function and by using a confusion matrix. For the fuzzy C-means we chose the C that is optimal for each dataset when evaluating the results. The output was:

Algorithm	Dataset	Error %	ARI	DB	Time
Kmeans	Segment	46.43%	0.44	0.65	1728.894ms
	Wine	6.04%	0.85	0.844	66.739ms
	Vehicles	73.88%	0.08	0.11	598.559ms
KHmeans	Segment	86.71%	0.0	1.0	25125.004ms
	Wine	71.11%	0.0	1.0	66.739ms
	Vehicles	75.23%	0.0	1.0	2081.069ms
Cmeans	Segment	36.67%	0.52	0.62	80850.299ms
	Wine	4.30%	0.90	0.87	590.001ms
	Vehicles	74.33%	0.0	1.0	73.013ms

The confusion matrix results are as follows:

Segment:

K-means:

Predicted class	Actual class							
		Brickface	Sky	Foliage	Cement	Window	Path	Grass
	Brickface	53	0	0	0	266	11	0
	Sky	0	330	0	0	0	0	0
	Foliage	18	0	20	5	262	19	6
	Cement	49	6	2	0	22	251	0

	Window	14	0	0	0	249	29	38
	Path	36	0	0	0	0	294	0
	Grass	2	0	0	0	1	1	326

KH-means:

Predicted class	Actual class							
		Brickface	Sky	Foliage	Cement	Window	Path	Grass
	Brickface	0	0	330	0	0	0	0
	Sky	0	0	330	0	0	0	0
	Foliage	0	0	330	0	0	0	0
	Cement	0	0	330	0	0	0	0
	Window	0	0	330	0	0	0	0
	Path	0	0	330	0	0	0	0
	Grass	0	0	330	0	0	0	0

Fuzzy C-means:

Predicted class	Actual class							
		Brickface	Sky	Foliage	Cement	Window	Path	Grass
	Brickface	150	0	42	0	138	0	0
	Sky	0	330	0	0	0	0	0
	Foliage	42	0	229	20	29	10	0
	Cement	37	5	1	155	13	119	0
	Window	86	0	89	15	140	0	0
	Path	13	0	0	65	0	252	0
	Grass	0	0	0	1	2	0	327

Wine:

K-means:

Predicted class	Actual class			
		1	2	3
	1	59	0	0
	2	8	62	1
	3	0	0	48

KH-means:

Predicted class	Actual class			
		1	2	3
	1	0	59	0
	2	0	71	0
	3	0	48	0

Fuzzy C-means:

Predicted class	Actual class			
		1	2	3
	1	59	0	0
	2	3	65	3
	3	0	0	48

Vehicle:
K-means:

Predicted class	Actual class				
		Opel	Saab	Bus	Van
	Opel	96	40	36	40
	Saab	91	45	39	42
	Bus	38	40	86	54
	Van	0	32	2	84

KH-means:

Predicted class	Actual class				
		Opel	Saab	Bus	Van

	Opel	0	0	212	0
	Saab	0	0	217	0
	Bus	0	0	218	0
	Van	0	0	199	0

Fuzzy C-means:

Predicted class	Actual class				
		Opel	Saab	Bus	Van
	Opel	103	38	34	37
	Saab	99	44	36	38
	Bus	43	42	84	49
	Van	3	44	70	82

Comparing the results of the different datasets with the different algorithms we can see how the results can be inaccurate when data is too close together, for example in both the Vehicle and Segment datasets. If we see the Segment datasets, the classes that are further apart from the rest, like the sky, has more correct elements (330 in K-means and C-means) than other classes that are probably close to each other, like the window, that can get confused with the brickface or the foliage. The same can be seen in the Vehicle dataset, where in this case, we can see that all the data is grouped together since there is no case like the sky from the Segment dataset, and that there are a lot of mistakes in each class but with no real tendency that can tell us how the data is grouped, which implies that it can all be grouped together in one cluster. This statement can also be validated by the fact that the optimized number of clusters for the fuzzy C-means algorithm for the Vehicle datasets is 1. The wine dataset on the other, is clearly divided by the different classes, since we can see the that the algorithms don’t struggle to separate the data into clusters, giving us a confusion matrix with more correct values than errors, and that the optimized number of clusters for the Fuzzy C-Means algorithm was the same as the number of classes.

Overall, the algorithms can tell us how close together the data is, and how easily it can be grouped together and classified. For the case of algorithms like the Fuzzy C-Means, it can also tell us the number of clusters that should be created, which tells us even more how the data is distributed, even though this algorithm is noticeably slower than the others.