Data Mining Summer Term 2020

Institute of Neural Information Processing
PD Dr. F. Schwenker

Assignment 6 (Submission until June 16, 2020)

Exercise 1 (5 points): Fuzzy Clustering [Pen and Paper]

The idea behind fuzzy clustering is to loosen the constraint that each data point gets assigned to exactly one cluster prototype as it is the case with crisp memberships. Instead, we define a membership $f_{\mu,j}$ denoting to which degree the data point $\boldsymbol{p}_\mu$ belongs to the prototype $\boldsymbol{c}_j$. To find clusters in the data, we can adjust the vanilla k-means algorithm to the idea of fuzzy memberships. We want to see how this adjustment works in this exercise.

Mathematically, we adapt the cluster re-calculation step to include the membership $f_{\mu,j}$ and a fuzzifier $b > 1$ leading to

$$\boldsymbol{c}_j = \frac{1}{\sum_{\mu=1}^{n} f_{\mu,j}^b} \sum_{\mu=1}^{n} f_{\mu,j}^b \boldsymbol{p}_\mu. \tag{1}$$

We do not have distinct cluster sets C_j anymore since per definition every data point is assigned to some degree to every cluster. Hence, we have to calculate the new memberships between all point-and-cluster pairs in the re-assignment step

$$f_{\mu,j} = \frac{1}{\sum_{i=1}^{k} \left(\frac{\|\boldsymbol{p}_\mu - \boldsymbol{c}_j\|^2}{\|\boldsymbol{p}_\mu - \boldsymbol{c}_i\|^2} \right)^{\frac{1}{b-1}}}. \tag{2}$$

We are using a simple two-dimensional dataset visually similar to the one we used in crisp k-means clustering

$$\boldsymbol{p}_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \boldsymbol{p}_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad \boldsymbol{p}_3 = \begin{pmatrix} 2 \\ 1 \end{pmatrix} \quad \text{and} \quad \boldsymbol{p}_4 = \begin{pmatrix} 2 \\ 2 \end{pmatrix}. \tag{3}$$

We use $k = 2$ and start with the initial cluster centres

$$\boldsymbol{c}_1(0) = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad \text{and} \quad \boldsymbol{c}_2(0) = \begin{pmatrix} 1 \\ 2 \end{pmatrix}. \tag{4}$$

Like before, $\boldsymbol{c}_i(t)$ denotes the location of the i-th cluster centre in the iteration t. The initial situation is also shown in Figure 2.

1. Since we started with initial cluster centres, the first step of the k-means algorithm is to assign each point to the clusters. In the fuzzy world, we use a membership matrix

$$F = \begin{pmatrix} 5/7 & 2/7 \\ f_{2,1} & f_{2,2} \\ 2/3 & f_{3,2} \\ f_{4,1} & f_{4,2} \end{pmatrix}$$

where each row marks a data point and each column a cluster prototype. Calculate the missing values of the matrix F. Use a fuzzifier of $b = 2$. Hint: it is possible to calculate all five values and use Equation 2 only twice.

2. The next step is to re-calculate the new cluster centres. In the fuzzy world, the cluster prototype is a weighted average of all data points. Calculate $\boldsymbol{c}_1(1)$ and $\boldsymbol{c}_2(1)$ (using again $b = 2$) and draw the new cluster centres into Figure 2.

3. The algorithm now repeats the re-assignment and re-calculation steps until convergence. This is a bit cumbersome to do by hand, so Figure 3 shows the result after 100 iterations with the cluster prototypes

$$\boldsymbol{c}_1(100) = \begin{pmatrix} 0.485528 \\ 0.00456674 \end{pmatrix} \quad \text{and} \quad \boldsymbol{c}_2(100) = \begin{pmatrix} 1.99543 \\ 1.51447 \end{pmatrix}.$$

   Does this differ much from the expected result of k-means with crisp membership?

4. The fuzzifier b controls the amount of overlap between the cluster assignments. That is, whether a data point contributes mainly to one cluster or "fuzzifies" its membership between multiple clusters. In terms of Figure 2, it controls whether from one data point one very thick and one very thin line goes to the cluster prototypes or if both lines have a significant thickness. Note that the fuzzifier b occurs both in Equation 1 and in Equation 2. In the former, it is used as an exponent to soften the memberships and in the latter, it is required to find the optimal membership value itself.

   a) Let's see about the effect of the fuzzifier b. To analyse how the contribution to cluster prototypes changes with increasing value of b, we use the initial cluster centres of Equation 4 and the membership of the first data point $\boldsymbol{p}_1$ to these clusters. Figure 1 shows the values $f_{1,1}^b$ and $f_{1,2}^b$ as a function of b. To say something about the contribution to each cluster compared to the overall contribution, also the re-scaled values

$$\tilde{f}_{1,j} = \frac{f_{1,j}^b}{f_{1,1}^b + f_{1,2}^b} \tag{5}$$

   are depicted in Figure 1.

      i. If you look at the values $f_{1,j}^b$, you can see that both get very small. Does this necessarily mean that our algorithm does not work anymore?

      ii. Based on the scaled values $\tilde{f}_{1,j}$, how does the mentioned overlap between the clusters change for larger values for b? How does the thickness of the lines change?

   b) If you look at the left part of Figure 1, you see that we have a near crisp membership for values of b very close to 1. To make this more precise, determine the limiting values

$$\lim_{b \to 1+} f_{1,1}^b \quad \text{and} \quad \lim_{b \to 1+} f_{1,2}^b$$

and interpret the result. Hint: it might be helpful to rewrite Equation 2 as

$$f_{\mu,j} = \frac{1}{\sum_{i=1}^{k} \left(\frac{\|\boldsymbol{p}_\mu - \boldsymbol{c}_j\|^2}{\|\boldsymbol{p}_\mu - \boldsymbol{c}_i\|^2} \right)^{\frac{1}{b-1}}} = \frac{1}{1 + \sum_{\substack{i=1, \\ i \neq j}}^{k} \left(\frac{\|\boldsymbol{p}_\mu - \boldsymbol{c}_j\|^2}{\|\boldsymbol{p}_\mu - \boldsymbol{c}_i\|^2} \right)^{\frac{1}{b-1}}}.$$

c) The situation for increasing value of b is a bit more complicated. You can already see from Figure 1 that the membership values $f_{1,j}^b$ tend to approximate zero but the more interesting question is where the scaled version $\tilde{f}_{1,j}$ has its limits. Figure 1 indicates that this is not zero but it does look like as if there is indeed an asymptotic value. It is a bit harder to show but in the limits we get

$$\lim_{b \to \infty} \tilde{f}_{1,1} = 1 - \frac{\|\boldsymbol{p}_1 - \boldsymbol{c}_1\|^2}{\|\boldsymbol{p}_1 - \boldsymbol{c}_1\|^2 + \|\boldsymbol{p}_1 - \boldsymbol{c}_2\|^2} \quad \text{and} \quad \lim_{b \to \infty} \tilde{f}_{1,2} = 1 - \frac{\|\boldsymbol{p}_1 - \boldsymbol{c}_2\|^2}{\|\boldsymbol{p}_1 - \boldsymbol{c}_1\|^2 + \|\boldsymbol{p}_1 - \boldsymbol{c}_2\|^2}.$$

Interpret these expressions and calculate both values explicitly by plugging in the initial cluster centroids $\boldsymbol{c}_1(0)$ and $\boldsymbol{c}_2(0)$ as well as the first data point $\boldsymbol{p}_1$.

d) Based on your knowledge of the fuzzifier b, how would the result after 100 iterations (Figure 3) change if we used a higher value for b?

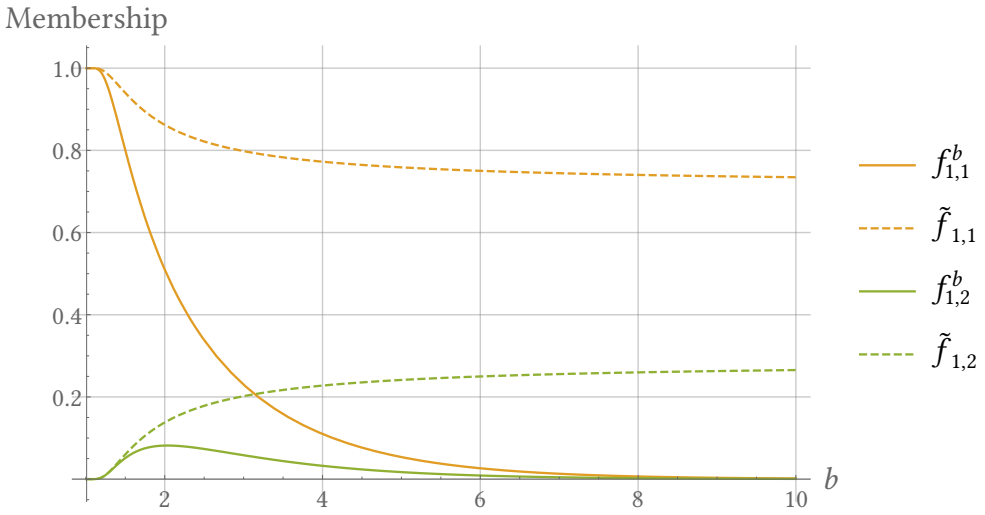

Figure 1: Analysis of the default $f_{1,j}^b$ and scaled membership $\tilde{f}_{1,j}$ as a function of the fuzzifier b.

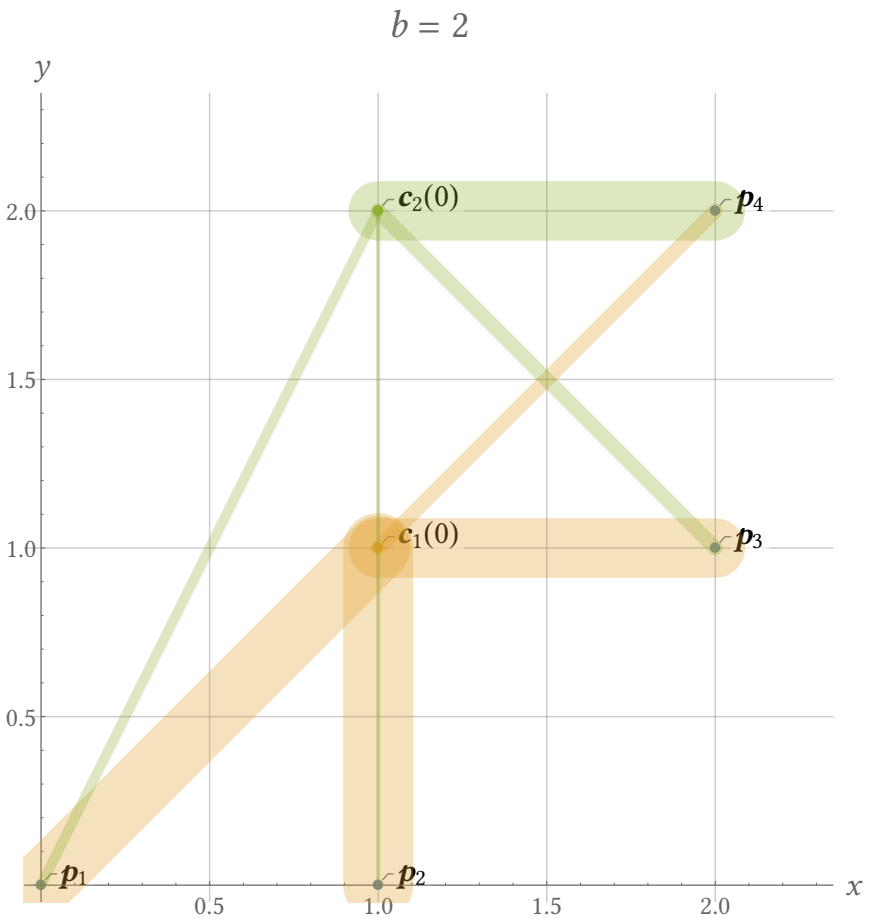

Figure 2: Example dataset (Equation 3) and initial cluster prototypes (Equation 4) used in Exercise 1. The thickness of the lines between the data points and the cluster prototypes corresponds to the strength of the scaled membership $\tilde{f}_{\mu,j}$ (Equation 5).

Exercise 2 (5 points): Crisp vs. Fuzzy k-means

We discussed in previous exercises how the k-means algorithm works and that we can use either the crisp or fuzzy method. Whereas in the former each data point is assigned only to its nearest cluster centre, we have a notion of membership values in the latter. These values define that each point belongs, at least some degree, to all cluster centres. We covered both concepts but what is left is a comparison. This exercise tries to fill this gap.

1. [Pen and Paper] In your opinion, argue whether crisp or fuzzy k-means has the better computational performance. Then, search the web to see whether you can find some benchmarks which support your theory.
2. [Pen and Paper] We now want to analyse how the two methods behave on an artificially generated dataset with increasing noise. For this, use the corresponding animation[1] where

[1] https://milania.de/showcase/Crisp_vs._fuzzy_k-means_clustering

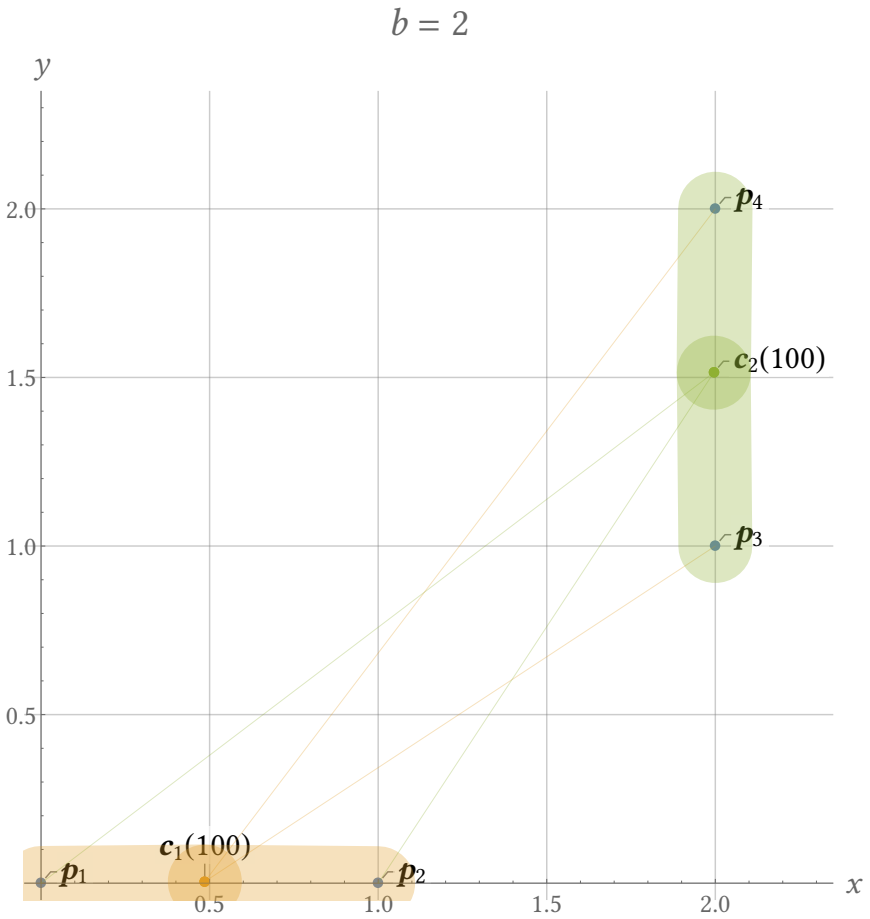

Figure 3: Result of the fuzzy k-means algorithm after 100 iterations (converged).

you can control the number of new (noise) points which are added to the bottom right area of the two-dimensional dataset.

a) By analysis of the first two plots, describe how the two methods react differently to increasing noise and explain how these differences arise.

b) The last of the three plots shows the movement of the cluster centres subject to increasing noise. For both methods, the cluster centres move to the bottom right corner since this is where the new points are generated. Despite the jumps which occur on high noise influence, it seems that the crisp cluster centres move ahead of the fuzzy cluster centres. Explain why this is the case.

3. Let's apply k-means clustering to a more realistic application. We take the opportunity and explore another field where k-means clustering is used (crisp and fuzzy): image segmentation. The idea is to take all pixels of an image, extract a long list of three-dimensional vectors corresponding to all colours and then apply one of our clustering algorithms to it. As a result, we get the main colours of an image which hopefully

correspond to the main objects seen in the image. What is more, we can replace each colour by its corresponding cluster prototype to retrieve a colour-reduced version of the image.

In order to compare the clustering algorithms, we are adding artificially generated noise to an image. We then analyse the segmentation results of the two methods and see how they behave differently. For the test script, please pay also attention to Table 1.

a) [Python] Load the example image[2] `flower.jpg`, transform it to the range $[0; 1]$ and scale it down to 25 % of its original size to reduce computational complexity. Useful functions: `load_sample_image` and `rescale` from the `sklearn.datasets` and `skimage.transform` package, respectively.

b) [Python] Generate three different versions of the image by adding random salt & pepper noise which replaces random pixels with extreme values. Use the noise ratios $r \in \{0, 0.02, 0.05\}$ where the first version denotes the original image. The function `random_noise` from the `skimage.util` is helpful for this task (set the `seed` parameter to 1).

c) [Python] Write a function `cluster_image(image,n_clusters,fuzzifier)` which segments the input `image` with `n_clusters` colours and the `fuzzifier` parameter $b > 1$.

   i. We need to remove the positional information of the pixels in the image before we can cluster the colour data. Reshape the image matrix so that you retrieve a long list of vectors corresponding to all pixel colours in the image. The result should be a $n \times 3$ matrix for an image with n pixels.

   ii. Apply fuzzy k-means to the pixel data by using the `cmeans` function from the `skfuzzy` package (you may need to install this package first). Set the stopping error to 0.0001, run the algorithm for at most 2000 iterations and set the random seed to 1. The goal is to retrieve the cluster centroids. Hint: pay attention to the `data` argument and its expected shape.

   iii. We need to find the nearest cluster centroid for each colour in the image. For this, associate each colour vector with its closest cluster label.

   iv. Replace each colour in the image with the corresponding colour of the cluster centroid and reshape the result back to an image. This result is the segmented image.

d) [Python] Visualize the clustering result for the different noise ratios and the fuzzifiers $b \in \{1.01, 3\}$ ($b = 1.01$ serves as an approximation for the crisp method). An example output can be seen in Figure 4.

e) [Pen and Paper] Let's interpret the result.

[2] Alternatively, you can also test some of your own images.

i. How does the clustering result between the crisp and the fuzzy method differ?

ii. The amount of visible noise does not seem to be reduced in the fuzzy results. Is this a contradiction to our general observation that fuzzy clustering is more stable against noise influence or do you have a different explanation?

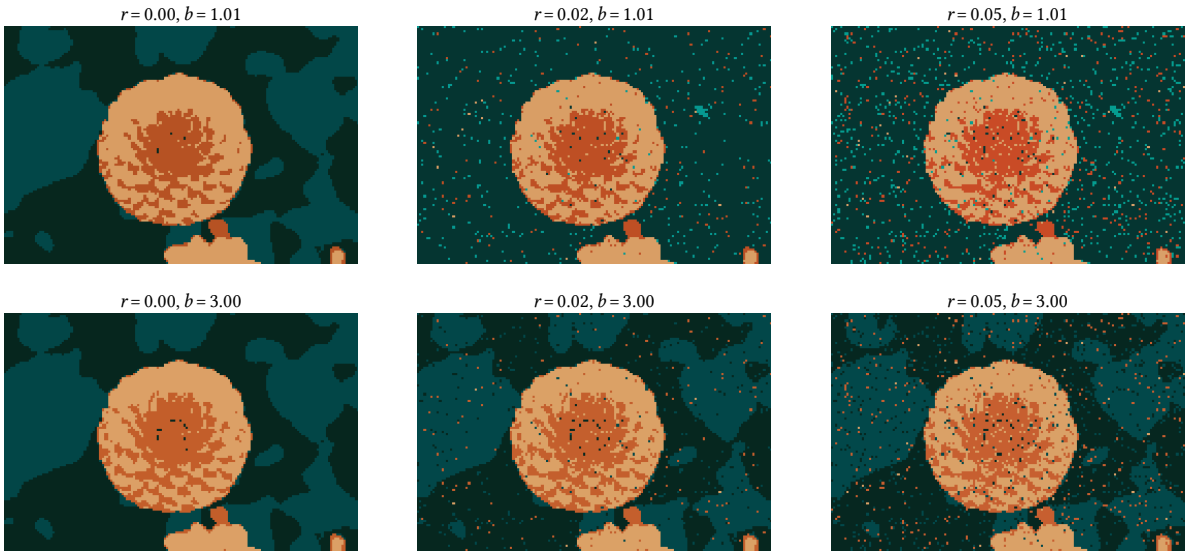

Figure 4: Comparison of the crisp and fuzzy k-means method for an example image subject to noise.

Table 1: Variable names for the notebook `CrispVsFuzzy.ipynb` (Exercise 2). You have to name your variables exactly as noted here so that the test script can check their content.

Name	Type	Description
`image`	Matrix	The down-scaled version of the example image.
`noise_ratios`	List	The noise ratios r.
`images_noise`	List	The different noisy versions of the example image (corresponding to the noise ratios r).