Aprendizagem 2022

Homework IV – Group 019

Diogo Gaspar 99207, Rafael Oliveira 99311

Part I: Pen and paper

1. **Perform one epoch of the EM clustering algorithm and determine the new parameters.**

As a side note, we'll be using the k_1 and k_2 notation to represent clusters 1 and 2 - with that, we'll say that $\pi_1 = P(C = k_1)$, with analogous notation for π_2.

EM-Clustering, being an unsupervised learning algorithm intending to calculate the probability of a sample belonging to a certain cluster, is a method that iteratively updates the parameters of the model until convergence is reached (for a given definition of convergence). Here, we'll perform exactly one epoch of the algorithm, which means we'll be going through two steps:

- **E-step:** In this step, our goal is to calculate the **posterior probability** of each sample belonging to each cluster. In order to perform this calculation, we'll be using **Bayes' rule**, of course, to decompose the posterior probability into the product of the **likelihood** and the **prior probability** of the sample belonging to the cluster. Let's try, then, to assign each sample to the cluster that maximizes the posterior probability.

  For starters, we must first note that the likelihood of a sample belonging to a cluster is given by the **multivariate Gaussian distribution**, which can be written as (considering $d = 2$):

$$P(x_i \mid C = k_n) \sim \mathcal{N}(x_i; \mu_n, \Sigma_n) = \frac{1}{\sqrt{(2\pi)^d \det \Sigma_n}} \exp\left(-\frac{1}{2}(x_i - \mu_n)^T \Sigma_n^{-1}(x_i - \mu_n)\right)$$

  We'll also use the $\gamma_{i,j}$ notation to represent the normalized posteriors, where i matches the i-th sample and j the j-th cluster:

$$\gamma_{i,j} = \frac{\pi_j P(x_i \mid C = k_j)}{\sum_{k=1}^{d} \pi_k P(x_i \mid C = k_k)} = \frac{\pi_j \mathcal{N}(x_i \mid \mu_j, \Sigma_j)}{\sum_{k=1}^{d} \pi_k \mathcal{N}(x_i \mid \mu_k, \Sigma_k)}$$

  Moreover, in this step we'll use teal to denote the priors and purple to denote the likelihoods.

As a given, we have that the priors are (for every sample, of course):

$$P(C = k_1) = P(C = k_2) = \pi_1 = \pi_2 = 0.5$$

Regarding x_1, we have:

$$
\begin{aligned}
P(x_1 \mid C = k_1) &= \frac{1}{\sqrt{(2\pi)^d \det \Sigma_1}} \exp\left(-\frac{1}{2}(x_1 - \mu_1)^T \Sigma_1^{-1}(x_1 - \mu_1)\right) \\
&= \frac{1}{\sqrt{(2\pi)^2 \det \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}}} \exp\left(-\frac{1}{2}\left(\begin{bmatrix} 1 \\ 2 \end{bmatrix} - \begin{bmatrix} 2 \\ 2 \end{bmatrix}\right)^T \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}^{-1}\left(\begin{bmatrix} 1 \\ 2 \end{bmatrix} - \begin{bmatrix} 2 \\ 2 \end{bmatrix}\right)\right) \\
&= \frac{e^{-1/3}}{\sqrt{(2\pi)^2 \cdot 3}} \\
&\approx 0.0658407
\end{aligned}
$$

$$
\begin{aligned}
P(x_1 \mid C = k_2) &= \frac{1}{\sqrt{(2\pi)^d \det \Sigma_2}} \exp\left(-\frac{1}{2}(x_1 - \mu_2)^T \Sigma_2^{-1}(x_1 - \mu_2)\right) \\
&= \frac{1}{\sqrt{(2\pi)^2 \det \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}}} \exp\left(-\frac{1}{2}\left(\begin{bmatrix} 1 \\ 2 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right)^T \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}^{-1}\left(\begin{bmatrix} 1 \\ 2 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right)\right) \\
&\approx 0.0227993
\end{aligned}
$$

The (normalized) posteriors can be computed as follows:

$$
\begin{aligned}
\gamma_{1,1} &= \frac{P(C = k_1)P(x_1 \mid C = k_1)}{P(C = k_1)P(x_1 \mid C = k_1) + P(C = k_2)P(x_1 \mid C = k_2)} \\
&= \frac{0.5 \cdot 0.0658407}{0.5 \cdot 0.0658407 + 0.5 \cdot 0.0227993} \\
&= 0.742788
\end{aligned}
$$

$$
\begin{aligned}
\gamma_{1,2} &= \frac{P(C = k_2)P(x_1 \mid C = k_2)}{P(C = k_1)P(x_1 \mid C = k_1) + P(C = k_2)P(x_1 \mid C = k_2)} \\
&= \frac{0.5 \cdot 0.0227993}{0.5 \cdot 0.0658407 + 0.5 \cdot 0.0227993} \\
&= 0.257212
\end{aligned}
$$

As to avoid presenting here the repetitive calculations for the remaining samples, for which the results in question are calculated in the same manner, we include here only the intermediate values of such calculations, as discussed with Prof. Rui Henriques. The respective Python code is available in this report's appendix.

x_2:

$$P(x_2|C = k_1) = 0.00891057$$
$$P(x_2|C = k_2) = 0.0482662$$

$$\gamma_{2,1} = 0.155843$$
$$\gamma_{2,2} = 0.844157$$

x_3:

$$P(x_3|C = k_1) = 0.0338038$$
$$P(x_3|C = k_2) = 0.061975$$

$$\gamma_{3,1} = 0.352936$$
$$\gamma_{3,2} = 0.647064$$

- **M-step: Having calculated the posteriors, we can now update the parameters of the cluster-defining distributions.**

  For each cluster, we'll want to find the new distribution parameters: in this case, μ_k and Σ_k (for every cluster k). For likelihoods, we'll need to update both μ_k and Σ_k, using all samples weighted by their respective posteriors, as can be seen below; for priors, we'll need to perform a weighted mean of the posteriors.

$$\mu_\alpha = \frac{\sum_{i=1}^{3} P(C = k_\alpha \mid x_i)x_i}{\sum_{i=1}^{3} P(C = k_\alpha \mid x_i)}$$

$$\Sigma_\alpha = \frac{\sum_{i=1}^{3} P(C = k_\alpha \mid x_i)(x_i - \mu_\alpha)(x_i - \mu_\alpha)^T}{\sum_{i=1}^{3} P(C = k_\alpha \mid x_i)}$$

$$P(C = k_\alpha) = \frac{\sum_{i=1}^{3} P(C = k_\alpha \mid x_i)}{\sum_{c=1}^{2} \sum_{i=1}^{3} P(C = k_c \mid x_i)}$$

In the equations stated above, we're considering $x_{i,n}$ as the n-th feature's value of the i-th sample, and $\mu_{k,n}$ as the n-th index of centroid μ_k.

We can now estimate the new parameters of the distributions (and the new priors) as shown in the next page (note that the updated μ_α's are used in the calculation of the new Σ_α's). The Python code used for these calculations is also available in the appendix.

Regarding k_1:

$$\mu_1 = \frac{\sum_{i=1}^{3} P(C = k_1 \mid x_i) x_i}{\sum_{i=1}^{3} P(C = k_1 \mid x_i)}$$

$$= \frac{0.742788 \cdot \begin{bmatrix} 1 \\ 2 \end{bmatrix} + 0.155843 \cdot \begin{bmatrix} -1 \\ 1 \end{bmatrix} + 0.352936 \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix}}{0.742788 + 0.155843 + 0.352936}$$

$$= \begin{bmatrix} 0.750964 \\ 1.31149 \end{bmatrix}$$

$$\Sigma_1 = \frac{\sum_{i=1}^{3} P(C = k_1 \mid x_i)(x_i - \mu_1)(x_i - \mu_1)^T}{\sum_{i=1}^{3} P(C = k_1 \mid x_i)}$$

$$= \begin{bmatrix} 0.436053 & 0.0775726 \\ 0.0775726 & 0.778455 \end{bmatrix}$$

$$\pi_1 = P(C = k_1) = \frac{\sum_{i=1}^{3} P(C = k_1 \mid x_i)}{\sum_{c=1}^{2} \sum_{i=1}^{3} P(C = k_c \mid x_i)} = 0.417189$$

Regarding k_2:

$$\mu_2 = \frac{\sum_{i=1}^{3} P(C = k_2 \mid x_i) x_i}{\sum_{i=1}^{3} P(C = k_2 \mid x_i)}$$

$$= \frac{0.257212 \cdot \begin{bmatrix} 1 \\ 2 \end{bmatrix} + 0.844157 \cdot \begin{bmatrix} -1 \\ 1 \end{bmatrix} + 0.647064 \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix}}{0.257212 + 0.844157 + 0.647064}$$

$$= \begin{bmatrix} 0.0343846 \\ 0.777028 \end{bmatrix}$$

$$\Sigma_2 = \frac{\sum_{i=1}^{3} P(C = k_2 \mid x_i)(x_i - \mu_2)(x_i - \mu_2)^T}{\sum_{i=1}^{3} P(C = k_2 \mid x_i)}$$

$$= \begin{bmatrix} 0.998818 & -0.215305 \\ -0.215305 & 0.467476 \end{bmatrix}$$

$$\pi_2 = P(C = k_2) = \frac{\sum_{i=1}^{3} P(C = k_2 \mid x_i)}{\sum_{c=1}^{2} \sum_{i=1}^{3} P(C = k_c \mid x_i)} = 0.582811$$

2. **Given the updated parameters computed in previous question:**

(a) **Perform a hard assignment of observations to clusters under a MAP assumption.**

*Note that, since all calculations follow the same general expressions used in the previous question's **E-Step**, we have opted not to repeat them here, showing just the final results for each intermediate step instead. The code required for these calculations is, once again, available in the appendix.*

Just like in the first question's answer, we'll need to compute the posterior probabilities of each sample belonging to each cluster (now utilizing the newly updated parameters); however, instead of proceeding to the **M-Step**, we'll just assign each sample to the cluster with the highest posterior probability.

The priors have been updated in the previous question's answer to:

$$\pi_1 = 0.417189, \quad \pi_2 = 0.582811$$

Moreover, we've also updated the means and covariances of the distributions to:

$$\mu_1 = \begin{bmatrix} 0.750964 \\ 1.31149 \end{bmatrix} \qquad \Sigma_1 = \begin{bmatrix} 0.436053 & 0.0775726 \\ 0.0775726 & 0.778455 \end{bmatrix}$$

$$\mu_2 = \begin{bmatrix} 0.0343846 \\ 0.777028 \end{bmatrix} \qquad \Sigma_2 = \begin{bmatrix} 0.998818 & -0.215305 \\ -0.215305 & 0.467476 \end{bmatrix}$$

Therefore, for each sample, we'll have:

x_1:	x_2:	x_3:
$P(x_1\|C = k_1) = 0.195695$	$P(x_2\|C = k_1) = 0.00819528$	$P(x_3\|C = k_1) = 0.0771661$
$P(x_1\|C = k_2) = 0.0135196$	$P(x_2\|C = k_2) = 0.143645$	$P(x_3\|C = k_2) = 0.104784$
$\gamma_{1,1} = \underline{0.911983}$	$\gamma_{2,1} = 0.0392368$	$\gamma_{3,1} = 0.345186$
$\gamma_{1,2} = 0.0880168$	$\gamma_{2,2} = \underline{0.960763}$	$\gamma_{3,2} = \underline{0.654814}$

After performing these calculations, under a MAP (*Maximum A Posteriori*) assumption, we'll assign each sample to the cluster with the highest posterior probability, as underlined above:

x_1:	x_2:	x_3:
$\mathrm{MAP}(x_1) \mapsto k_1$	$\mathrm{MAP}(x_2) \mapsto k_2$	$\mathrm{MAP}(x_3) \mapsto k_2$

(b) **Compute the silhouette of the larger cluster using the Euclidean distance.**

As we know, the silhouette of a given sample x_i is defined as

$$s_i = \frac{b_i - a_i}{\max\{a_i, b_i\}},$$

where a_i is the average distance between x_i and all other samples in the same cluster, and b_i is the average distance between x_i and all other samples in its **neighboring cluster** - the neighboring cluster being, here, the cluster minimizing such average distance.

Moreover, the silhouette of a given cluster k_n, with m assigned samples, is defined as:

$$s(k_n) = \frac{\sum_{i=1}^{m} s_i}{m}$$

Here, the **largest cluster** will be the cluster with the greatest associated prior value (π_k). As we computed in 1., $\pi_2 = 0.582811 > 0.417189 = \pi_1$, hence the larger cluster will be k_2. Its assigned samples, considering a MAP assumption, are x_2 and x_3 (as asserted in the previous question's answer), so we'll have the following:

x_2:

$$a_2 = \frac{\|x_2 - x_3\|_2}{2 - 1}$$

$$= \left\|\begin{bmatrix} -1 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \end{bmatrix}\right\|_2 = 2.23607$$

$$b_2 = \min\left\{\frac{\|x_2 - x_1\|_2}{1}\right\}$$

$$= \left\|\begin{bmatrix} -1 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 \\ 2 \end{bmatrix}\right\|_2 = 2.23607$$

$$s_2 = \frac{b_2 - a_2}{\max\{a_2, b_2\}}$$

$$= \frac{2.23607 - 2.23607}{\max\{2.23607, 2.23607\}} = 0$$

x_3:

$$a_3 = \frac{\|x_3 - x_2\|_2}{2 - 1}$$

$$= \left\|\begin{bmatrix} 1 \\ 0 \end{bmatrix} - \begin{bmatrix} -1 \\ 1 \end{bmatrix}\right\|_2 = 2.23607$$

$$b_3 = \min\left\{\frac{\|x_3 - x_1\|_2}{1}\right\}$$

$$= \left\|\begin{bmatrix} 1 \\ 0 \end{bmatrix} - \begin{bmatrix} 1 \\ 2 \end{bmatrix}\right\|_2 = 2$$

$$s_3 = \frac{b_3 - a_3}{\max\{a_3, b_3\}}$$

$$= \frac{2 - 2.23607}{\max\{2.23607, 2\}} = -0.105573$$

With this, we can compute the silhouette of the larger cluster:

$$s(k_2) = \frac{s_2 + s_3}{2} = -0.052786$$

6

Part II: Programming and critical analysis

The code utilized to answer the following questions is available in this report's appendix.

Recall the `pd_speech.arff` dataset from earlier homeworks, centered on the Parkinson diagnosis from speech features. For the following exercises, normalize the data using `sklearn`'s `MinMaxScaler`.

3. **Using `sklearn`, apply k-means clustering fully unsupervisedly (without targets) on the normalized data with $k = 3$ and three different seeds (using random $\in$ $\{0, 1, 2\}$). Assess the silhouette and purity of the produced solutions.**

   `sklearn.metrics` offers us the `silhouette_score` method, which computes the silhouette of a given clustering solution. `purity_score` is a custom method defined in the appendix of this report, which computes the purity of a given clustering solution.

   Regarding the k-means clustering solutions for $k = 3$, for each of the given seeds, we were able to gather the following scores:

random_state	Silhouette score	Purity score
0	0.11362028	0.76719577
1	0.11403554	0.76322751
2	0.11362028	0.76719577

   Table 1: Silhouette and Purity scores for each clustering solution

4. **What is causing the non-determinism?**

   The non-determinism present in the k-means clustering solutions gathered in the previous exercise is caused by the fact that the algorithm is inherently random: `sklearn`'s KMeans class sets up the centroids' initial positions in a randomly generated fashion, thus leading to possible different convergence points for the same data and number of clusters. `random_state`, here, works as a mere manner of controlling the random seed used to generate the initial centroid positions: for the same seed, the same initial centroids' positions will be generated, thus leading to the same convergence point. For different seeds, different initial centroid positions will be generated, which will generally lead to different convergence points. In the case seen above, with seeds 0 and 2, the convergence point reached was practically the same (in terms of evaluated), even though the final centroids' positions were different - by calculating the norm of the difference between the final centroids' positions for each run, we were able to gather the following results:

Difference - seeds 0 and 1	Difference - seeds 0 and 2	Difference - seeds 1 and 2
3.1566	4.4269	3.3163

Table 2: Norm of the difference between the final centroids' positions for each run

5. **Using a scatter plot, visualize side-by-side the labeled data using as labels: i) the original Parkinson diagnoses, and ii) the previously learned $k = 3$ clusters (`random = 0`). To this end, select the two most informative features as axes and color observations according to their label. For feature selection, select the two input variables with highest variance on the MinMax normalized data.**

   In order to select the two most informative features, we'll use the custom method `select_most_informative_features` defined in the appendix of this report. After gathering that the two input variables presenting the highest variance are both `tqwt_entropy_shannon_dec_16` and `tqwt_kurtosisValue_dec_34`, we were able to gather the scatter plots present in Figure 1 (see Appendix page 15).

6. **The fraction of variance explained by a principal component is the ratio between the variance of that component (i.e., its eigenvalue) and total variance (i.e., sum of all eigenvalues). How many principal components are necessary to explain more than 80% of variability?**

   The PCA class from `sklearn.decomposition` can be used to compute the **principal components** of a dataset. Moreover, its `n_components` parameter, if set to a value between 0 and 1, will automatically select the number of components necessary to explain a fraction of variability greater than the given value - in our case, 0.8 - which can be consulted in the `n_components_` attribute.

   After running our `calculate_pca` method, present in the latter section of this report's appendix, we were able to gather that the number of principal components necessary to explain 80% of variability is 31.

Appendix

Code used to answer the questions in the Pen-and-Paper section:

```python
import numpy as np

### Question 1 ###

# Initial data

x_1 = np.array([[1], [2]])
x_2 = np.array([[-1], [1]])
x_3 = np.array([[1], [0]])

mu_1 = np.array([[2], [2]])
mu_2 = np.array([[0], [0]])

Sigma_1 = np.array([[2, 1], [1, 2]])
Sigma_2 = np.array([[2, 0], [0, 2]])

from scipy.stats import multivariate_normal


def calc_likelihoods(x, mu_1, mu_2, Sigma_1, Sigma_2):
    likelihood_1 = multivariate_normal(mu_1, Sigma_1).pdf(x.T)
    likelihood_2 = multivariate_normal(mu_2, Sigma_2).pdf(x.T)
    return np.array([likelihood_1, likelihood_2])


def calc_posteriors(priors, likelihoods):
    posteriors = np.array([])
    for i in range(len(priors)):
        posteriors = np.append(posteriors, priors[i] * likelihoods[i])

    return posteriors / np.sum(posteriors)  # normalize


def update_means(k1_posteriors, k2_posteriors):
    mu_1 = np.zeros((2, 1), dtype=float)
    mu_2 = np.zeros((2, 1), dtype=float)

    for i in range(len(k1_posteriors)):
        x = eval(f"x_{i+1}")
        mu_1 += k1_posteriors[i] * x
        mu_2 += k2_posteriors[i] * x

    return mu_1 / np.sum(k1_posteriors), mu_2 / np.sum(k2_posteriors)


def update_covs(k1_posteriors, k2_posteriors, mu_1, mu_2):
    Sigma_1 = np.zeros((2, 2), dtype=float)
    Sigma_2 = np.zeros((2, 2), dtype=float)

    for i in range(len(k1_posteriors)):
        x = eval(f"x_{i+1}")
        Sigma_1 += k1_posteriors[i] * (x - mu_1) @ (x - mu_1).T
        Sigma_2 += k2_posteriors[i] * (x - mu_2) @ (x - mu_2).T
```

```
54
55      return Sigma_1 / np.sum(k1_posteriors), Sigma_2 / np.sum(k2_posteriors)
56
57
58  def update_priors(k1_posteriors, k2_posteriors):
59      total = np.sum(k1_posteriors) + np.sum(k2_posteriors)
60      return np.sum(k1_posteriors) / total, np.sum(k2_posteriors) / total
61
62
63  mu_1_vector = mu_1.transpose()[0]
64  mu_2_vector = mu_2.transpose()[0]
65
66  priors = np.array([0.5, 0.5])
67
68  p_x_1_given_k_1, p_x_1_given_k_2 = calc_likelihoods(
69      x_1, mu_1_vector, mu_2_vector, Sigma_1, Sigma_2
70  )
71  p_x_2_given_k_1, p_x_2_given_k_2 = calc_likelihoods(
72      x_2, mu_1_vector, mu_2_vector, Sigma_1, Sigma_2
73  )
74  p_x_3_given_k_1, p_x_3_given_k_2 = calc_likelihoods(
75      x_3, mu_1_vector, mu_2_vector, Sigma_1, Sigma_2
76  )
77
78  posteriors_x_1 = calc_posteriors(priors, [p_x_1_given_k_1, p_x_1_given_k_2])
79  posteriors_x_2 = calc_posteriors(priors, [p_x_2_given_k_1, p_x_2_given_k_2])
80  posteriors_x_3 = calc_posteriors(priors, [p_x_3_given_k_1, p_x_3_given_k_2])
81
82  k1_posteriors = np.array([posteriors_x_1[0], posteriors_x_2[0], posteriors_x_3[0]])
83  k2_posteriors = np.array([posteriors_x_1[1], posteriors_x_2[1], posteriors_x_3[1]])
84
85  # update the parameters
86
87  mu_1_after_update, mu_2_after_update = update_means(k1_posteriors, k2_posteriors)
88  Sigma_1_after_update, Sigma_2_after_update = update_covs(
89      k1_posteriors, k2_posteriors, mu_1_after_update, mu_2_after_update
90  )
91
92  # update the priors
93
94  priors_after_update = update_priors(k1_posteriors, k2_posteriors)
95  prior_1_update, prior_2_update = priors_after_update
96
97  ### Question 2a ###
98
99  mu_1_after_update_vector = mu_1_after_update.transpose()[0]
100 mu_2_after_update_vector = mu_2_after_update.transpose()[0]
101
102 updated_p_x_1_given_k_1, updated_p_x_1_given_k_2 = calc_likelihoods(
103     x_1,
104     mu_1_after_update_vector,
105     mu_2_after_update_vector,
106     Sigma_1_after_update,
107     Sigma_2_after_update,
108 )
109 updated_p_x_2_given_k_1, updated_p_x_2_given_k_2 = calc_likelihoods(
```

```
110      x_2,
111      mu_1_after_update_vector,
112      mu_2_after_update_vector,
113      Sigma_1_after_update,
114      Sigma_2_after_update,
115  )
116  updated_p_x_3_given_k_1, updated_p_x_3_given_k_2 = calc_likelihoods(
117      x_3,
118      mu_1_after_update_vector,
119      mu_2_after_update_vector,
120      Sigma_1_after_update,
121      Sigma_2_after_update,
122  )
123
124  updated_posteriors_x_1 = calc_posteriors(
125      priors_after_update, [updated_p_x_1_given_k_1, updated_p_x_1_given_k_2]
126  )
127  updated_posteriors_x_2 = calc_posteriors(
128      priors_after_update, [updated_p_x_2_given_k_1, updated_p_x_2_given_k_2]
129  )
130  updated_posteriors_x_3 = calc_posteriors(
131      priors_after_update, [updated_p_x_3_given_k_1, updated_p_x_3_given_k_2]
132  )
133
134  hard_assignments = (
135      np.argmax(
136          np.array(
137              [updated_posteriors_x_1, updated_posteriors_x_2, updated_posteriors_x_3
138          ]
138          ),
139          axis=1,
140      )
141      + 1
142  )
143
144  ### Question 2b ###
145
146  # calculate the norm between x1 and x2, x2 and x3, x1 and x3
147
148  norm_x1_x2 = np.linalg.norm(x_1 - x_2)
149  norm_x2_x3 = np.linalg.norm(x_2 - x_3)
150  norm_x1_x3 = np.linalg.norm(x_1 - x_3)
151
152  # calculate the silhouette
153
154  s_2 = (norm_x1_x2 - norm_x2_x3) / norm_x1_x2
155  s_3 = (norm_x1_x3 - norm_x2_x3) / max(norm_x1_x3, norm_x2_x3)
156
157  s_k_2 = (s_2 + s_3) / 2
```

Listing 1: Pen-and-Paper section code

Code used to answer the questions in the Programming and Critical Analysis section:

```
1  import numpy as np
2  import pandas as pd
3  from matplotlib import rc
4  import matplotlib.pyplot as plt
5  import seaborn as sns
6  from scipy.io.arff import loadarff
7  from sklearn.cluster import KMeans
8  from sklearn.metrics import silhouette_score
9  from sklearn.metrics.cluster import contingency_matrix
10 from sklearn.preprocessing import MinMaxScaler
11 from sklearn.decomposition import PCA
12
13 sns.set_style("darkgrid")
14 rc("font", **{"family": "serif", "serif": ["Computer Modern"]})
15 rc("text", usetex=True)
16
17 # Load the data
18 data = loadarff("data/pd_speech.arff")
19 df = pd.DataFrame(data[0])
20 df["class"] = df["class"].str.decode("utf-8")
21
22 # Exercise 1 below
23
24
25 def calculate_scores(kmeans):
26     def purity_score(y_true, y_pred):
27         confusion_matrix = contingency_matrix(y_true, y_pred)
28         return np.sum(np.amax(confusion_matrix, axis=0)) / np.sum(confusion_matrix)
29
30     silhouette_scores = np.array([])
31     purity_scores = np.array([])
32     for k in kmeans:
33         silhouette_scores = np.append(
34             silhouette_scores, silhouette_score(X_scaled, k.labels_)
35         )
36         purity_scores = np.append(purity_scores, purity_score(labels, k.labels_))
37     print(f"Silhouette scores: {silhouette_scores}")
38     print(f"Purity scores: {purity_scores}")
39
40
41 X = df.iloc[:, :-1].values
42 labels = df.iloc[:, -1].values
43
44 ## Normalize the data
45 scaler = MinMaxScaler()
46 X_scaled = scaler.fit_transform(X)
47
48 kmeans = [KMeans(n_clusters=3, random_state=seed).fit(X_scaled) for seed in range
       (3)]
49
50 calculate_scores(kmeans)
51
52 # get the cluster centroids
53 centroids = np.array([k.cluster_centers_ for k in kmeans])
54
```

```python
55  # show that the centroids between all seeds are different
56  print(f'Norm between centroids for seeds 0 and 1: {np.linalg.norm(centroids[0] -
        centroids[1]):.5g}')
57  print(f'Norm between centroids for seeds 0 and 2: {np.linalg.norm(centroids[0] -
        centroids[2]):.5g}')
58  print(f'Norm between centroids for seeds 1 and 2: {np.linalg.norm(centroids[1] -
        centroids[2]):.5g}')
59
60  # Exercise 4 below
61
62
63  def select_most_informative_features(kmeans, n):
64      variances = np.var(X_scaled, axis=0)
65      variance_indexes = np.argsort(variances)[::-1][:n]
66      return variance_indexes, [df.columns[i] for i in variance_indexes]
67
68
69  variance_indexes, most_informative_features = select_most_informative_features(
70      kmeans, 2
71  )
72
73  ## Plot the data
74  fig, axes = plt.subplots(1, 2, figsize=(12, 6))
75  x = X_scaled[:, variance_indexes[0]]
76  y = X_scaled[:, variance_indexes[1]]
77
78  print(most_informative_features)
79
80  plt.suptitle("Ground truth vs $k$-means clustering of Parkinson's dataset",
        fontsize=15)
81
82  for i, ax in enumerate(axes):
83      if i == 0:
84          sns.scatterplot(x=x, y=y, hue=labels, ax=ax, palette="Set2")
85          ax.set_title("Original labels")
86          ax.get_legend().set_title("Diagnosis")
87      else:
88          sns.scatterplot(x=x, y=y, hue=kmeans[0].labels_, ax=ax, palette="Set2")
89          ax.set_title("Cluster labels")
90          ax.get_legend().set_title("Cluster")
91      ax.set_xlabel(f"Feature: {most_informative_features[0]}")
92      ax.set_ylabel(f"Feature: {most_informative_features[1]}")
93  plt.savefig("assets/parkinsons.png")
94  plt.show()
95
96  # Exercise 6 below
97
98
99  def calculate_pca(X, n_components):
100     pca = PCA(n_components=n_components)
101     X_pca = pca.fit(X)
102     return X_pca
103
104
105 pca = calculate_pca(X_scaled, 0.8)
```

13

```
106  print(f"Number of principal components: {pca.n_components_}")
```
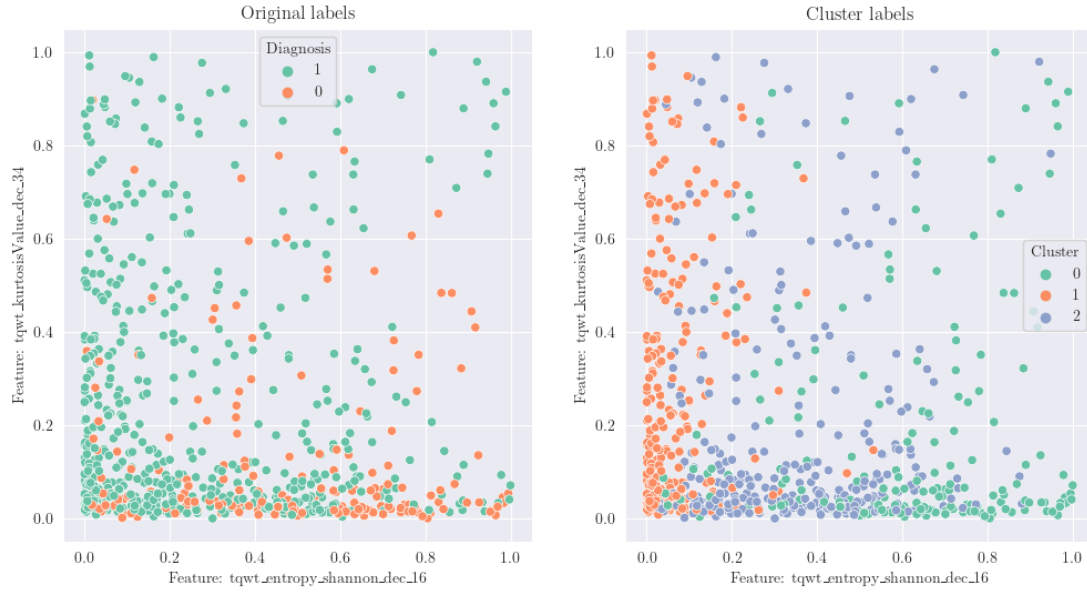
Listing 2: Programming section code

Figure 1: Ground truth vs *k*-means clustering of Parkinson's dataset