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:** Here, we're aiming 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 $d = 2$):

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

     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):

$$P(C = k_1) = P(C = k_2) = 0.5$$

Regarding x_1, we have:

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

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

The (normalized) posteriors can be computed as follows:

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

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

In order to avoid repeating showing such similar calculations for the remaining samples, and after talking with prof. Rui, we'll opt to write the results of all intermediate steps instead, provided that the `python` code required to perform the calculations is available in this report's appendix.

x_2:	x_3:
$P(x_2 \mid C = k_1) = 0.00891057$ $P(x_2 \mid C = k_2) = 0.0482662$	$P(x_3 \mid C = k_1) = 0.0338038$ $P(x_3 \mid C = k_2) = 0.061975$
$\gamma_{2,1} = 0.155843$ $\gamma_{2,2} = 0.844157$	$\gamma_{3,1} = 0.352936$ $\gamma_{3,2} = 0.647064$

- **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.

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

$$\Sigma_k^{nm} = \frac{\sum_{i=1}^{3} P(C = k \mid x_i)(x_{i,n} - \mu_{k,n})(x_{i,m} - \mu_{k,m})^T}{\sum_{i=1}^{3} P(C = k \mid x_i)}$$

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

  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_k's are used in the calculation of the new Σ_k's). The `python` code utilized for these calculations is also available in the appendix.

Regarding k_1:

$$\begin{aligned}
\mu_1 &= \frac{\sum_{i=1}^{3} P(C = k_1 \mid x_i) x_i}{\sum_{i=1}^{3} P(C = k_1 \mid x_i)} \\
&= \frac{0.742788 \cdot \begin{bmatrix} 1 \\ 2 \end{bmatrix} + 0.155843 \cdot \begin{bmatrix} -1 \\ 1 \end{bmatrix} + 0.352936 \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix}}{0.742788 + 0.155843 + 0.352936} \\
&= \begin{bmatrix} 0.750964 \\ 1.31149 \end{bmatrix}
\end{aligned}$$

$$\begin{aligned}
\Sigma_1^{nm} &= \frac{\sum_{i=1}^{3} P(C = k_1 \mid x_i)(x_{i,n} - \mu_{k_1,n})(x_{i,m} - \mu_{k_1,m})^T}{\sum_{i=1}^{3} P(C = k_1 \mid x_i)} \\
&= \begin{bmatrix} 0.436053 & 0.0775726 \\ 0.0775726 & 0.778455 \end{bmatrix}
\end{aligned}$$

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

Regarding k_2:

$$\begin{aligned}
\mu_2 &= \frac{\sum_{i=1}^{3} P(C = k_2 \mid x_i) x_i}{\sum_{i=1}^{3} P(C = k_2 \mid x_i)} \\
&= \frac{0.257212 \cdot \begin{bmatrix} 1 \\ 2 \end{bmatrix} + 0.844157 \cdot \begin{bmatrix} -1 \\ 1 \end{bmatrix} + 0.647064 \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix}}{0.257212 + 0.844157 + 0.647064} \\
&= \begin{bmatrix} 0.0343846 \\ 0.777028 \end{bmatrix}
\end{aligned}$$

$$\begin{aligned}
\Sigma_2^{nm} &= \frac{\sum_{i=1}^{3} P(C = k_2 \mid x_i)(x_{i,n} - \mu_{k_2,n})(x_{i,m} - \mu_{k_2,m})^T}{\sum_{i=1}^{3} P(C = k_2 \mid x_i)} \\
&= \begin{bmatrix} 0.998818 & -0.215305 \\ -0.215305 & 0.467476 \end{bmatrix}
\end{aligned}$$

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

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 formulas utilized in the previous question's **E-Step**, we're not going to repeat them here, opting instead to just write the final results for each intermediate step. 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:
$P(x_1 \mid C = k_1) = 0.1957$ $P(x_1 \mid C = k_2) = 0.01352$	$P(x_2 \mid C = k_1) = 0.0081953$ $P(x_2 \mid C = k_2) = 0.14365$	$P(x_3 \mid C = k_1) = 0.077166$ $P(x_3 \mid C = k_2) = 0.10478$
$\gamma_{1,1} = \underline{0.91198}$ $\gamma_{1,2} = 0.088017$	$\gamma_{2,1} = 0.039237$ $\gamma_{2,2} = \underline{0.96076}$	$\gamma_{3,1} = 0.34519$ $\gamma_{3,2} = \underline{0.65481}$

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

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

(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, therefore, 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 biggest associated prior value (π_k). As was 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:

$$\begin{aligned}
a_2 &= \frac{\|x_2 - x_3\|_2}{2-1} \\
&= \left\| \begin{bmatrix} -1 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right\|_2 = 2.2361 \\
b_2 &= \min\left\{ \frac{\|x_2 - x_1\|_2}{1} \right\} \\
&= \left\| \begin{bmatrix} -1 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 \\ 2 \end{bmatrix} \right\|_2 = 2.2361 \\
s_2 &= \frac{b_2 - a_2}{\max\{a_2, b_2\}} \\
&= \frac{2.2361 - 2.2361}{\max\{2.2361, 2.2361\}} = 0
\end{aligned}$$

x_3:

$$\begin{aligned}
a_3 &= \frac{\|x_3 - x_2\|_2}{2-1} \\
&= \left\| \begin{bmatrix} 1 \\ 0 \end{bmatrix} - \begin{bmatrix} -1 \\ 1 \end{bmatrix} \right\|_2 = 2.2361 \\
b_3 &= \min\left\{ \frac{\|x_3 - x_1\|_2}{1} \right\} \\
&= \left\| \begin{bmatrix} 1 \\ 0 \end{bmatrix} - \begin{bmatrix} 1 \\ 2 \end{bmatrix} \right\|_2 = 2 \\
s_3 &= \frac{b_3 - a_3}{\max\{a_3, b_3\}} \\
&= \frac{2 - 2.2361}{\max\{2.2361, 2\}} = -0.11803
\end{aligned}$$

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

$$s(k_2) = \frac{s_2 + s_3}{2} = -0.059015$$

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 could lead to possible different convergence points.

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.

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 the `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 utilized 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

  return Sigma_1 / np.sum(k1_posteriors), Sigma_2 / np.sum(k2_posteriors)

def update_priors(k1_posteriors, k2_posteriors):
```

```
  total = np.sum(k1_posteriors) + np.sum(k2_posteriors)
  return np.sum(k1_posteriors) / total, np.sum(k2_posteriors) / total

mu_1_vector = mu_1.transpose()[0]
mu_2_vector = mu_2.transpose()[0]

priors = np.array([0.5, 0.5])

p_x_1_given_k_1, p_x_1_given_k_2 = calc_likelihoods(x_1, mu_1_vector, mu_2_vector,
    Sigma_1, Sigma_2)
p_x_2_given_k_1, p_x_2_given_k_2 = calc_likelihoods(x_2, mu_1_vector, mu_2_vector,
    Sigma_1, Sigma_2)
p_x_3_given_k_1, p_x_3_given_k_2 = calc_likelihoods(x_3, mu_1_vector, mu_2_vector,
    Sigma_1, Sigma_2)

posteriors_x_1 = calc_posteriors(priors, [p_x_1_given_k_1, p_x_1_given_k_2])
posteriors_x_2 = calc_posteriors(priors, [p_x_2_given_k_1, p_x_2_given_k_2])
posteriors_x_3 = calc_posteriors(priors, [p_x_3_given_k_1, p_x_3_given_k_2])

k1_posteriors = np.array([posteriors_x_1[0], posteriors_x_2[0], posteriors_x_3[0]])
k2_posteriors = np.array([posteriors_x_1[1], posteriors_x_2[1], posteriors_x_3[1]])

# update parameters

mu_1_after_update, mu_2_after_update = update_means(k1_posteriors, k2_posteriors)
Sigma_1_after_update, Sigma_2_after_update = update_covs(k1_posteriors,
    k2_posteriors, mu_1_after_update, mu_2_after_update)

# update the priors

priors_after_update = update_priors(k1_posteriors, k2_posteriors)
prior_1_update, prior_2_update = priors_after_update

### Question 2a ###

mu_1_after_update_vector = mu_1_after_update.transpose()[0]
mu_2_after_update_vector = mu_2_after_update.transpose()[0]

updated_p_x_1_given_k_1, updated_p_x_1_given_k_2 = calc_likelihoods(x_1,
    mu_1_after_update_vector, mu_2_after_update_vector, Sigma_1_after_update,
    Sigma_2_after_update)
updated_p_x_2_given_k_1, updated_p_x_2_given_k_2 = calc_likelihoods(x_2,
    mu_1_after_update_vector, mu_2_after_update_vector, Sigma_1_after_update,
    Sigma_2_after_update)
updated_p_x_3_given_k_1, updated_p_x_3_given_k_2 = calc_likelihoods(x_3,
    mu_1_after_update_vector, mu_2_after_update_vector, Sigma_1_after_update,
    Sigma_2_after_update)

updated_posteriors_x_1 = calc_posteriors(priors_after_update, [
    updated_p_x_1_given_k_1, updated_p_x_1_given_k_2])
updated_posteriors_x_2 = calc_posteriors(priors_after_update, [
    updated_p_x_2_given_k_1, updated_p_x_2_given_k_2])
updated_posteriors_x_3 = calc_posteriors(priors_after_update, [
    updated_p_x_3_given_k_1, updated_p_x_3_given_k_2])

hard_assignments = np.argmax(np.array([updated_posteriors_x_1,
```

```
    updated_posteriors_x_2, updated_posteriors_x_3]), axis=1) + 1

### Question 2b ###

# Calculate the norm between x1 and x2, x2 and x3, x1 and x3

norm_x1_x2 = np.linalg.norm(x_1 - x_2)
norm_x2_x3 = np.linalg.norm(x_2 - x_3)
norm_x1_x3 = np.linalg.norm(x_1 - x_3)
```

Code utilized to answer the questions in the Programming and critical analysis section:

```
import numpy as np
import pandas as pd
from matplotlib import rc
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.io.arff import loadarff
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.metrics.cluster import contingency_matrix
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
sns.set_style('darkgrid')
rc('font', **{'family': 'serif', 'serif': ['Computer Modern']})
rc('text', usetex=True)

# Load the data
data = loadarff('data/pd_speech.arff')
df = pd.DataFrame(data[0])
df['class'] = df['class'].str.decode('utf-8')

def calculate_scores(kmeans):
    def purity_score(y_true, y_pred):
        confusion_matrix = contingency_matrix(y_true, y_pred)
        return np.sum(np.amax(confusion_matrix, axis=0)) / np.sum(confusion_matrix)

    silhouette_scores = np.array([])
    purity_scores = np.array([])
    for k in kmeans:
        silhouette_scores = np.append(silhouette_scores, silhouette_score(X_scaled,
    k.labels_))
        purity_scores = np.append(purity_scores, purity_score(labels, k.labels_))
    print(f'Silhouette scores: {silhouette_scores}')
    print(f'Purity scores: {purity_scores}')

X = df.iloc[:, :-1].values
labels = df.iloc[:, -1].values

# Normalize the data
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

kmeans = [KMeans(n_clusters=3, random_state=seed).fit(X_scaled) for seed in range
    (3)]

calculate_scores(kmeans)
```

```
44
45 # Exercise 4 below
46
47 def select_most_informative_features(kmeans, n):
48   variances = np.var(X_scaled, axis=0)
49   variance_indexes = np.argsort(variances)[::-1][:n]
50   return variance_indexes, [df.columns[i] for i in variance_indexes]
51
52 variance_indexes, most_informative_features = select_most_informative_features(
       kmeans, 2)
53
54 # Plot the data
55 fig, axes = plt.subplots(1, 2, figsize=(12, 6))
56 x = X_scaled[:, variance_indexes[0]]
57 y = X_scaled[:, variance_indexes[1]]
58
59 print(most_informative_features)
60
61 plt.suptitle('Ground truth vs $k$-means clustering of Parkinson\'s dataset',
       fontsize=15)
62
63 # TODO: change legend to show the actual class names/clusters
64
65 for i, ax in enumerate(axes):
66   if i == 0:
67     sns.scatterplot(x=x, y=y, hue=labels, ax=ax, palette='Set2')
68     ax.set_title('Original labels')
69   else:
70     sns.scatterplot(x=x, y=y, hue=kmeans[0].labels_, ax=ax, palette='Set2')
71     ax.set_title("Cluster labels")
72   ax.set_xlabel(f'Feature: {most_informative_features[0]}')
73   ax.set_ylabel(f'Feature: {most_informative_features[1]}')
74 plt.savefig('assets/parkinsons.png')
75 plt.show()
76
77 # Exercise 6 below
78
79 def calculate_pca(X, n_components):
80     pca = PCA(n_components=n_components)
81     X_pca = pca.fit(X)
82     return X_pca
83
84 pca = calculate_pca(X_scaled, 0.8)
85 print(f"Number of principal components: {pca.n_components_}")
```

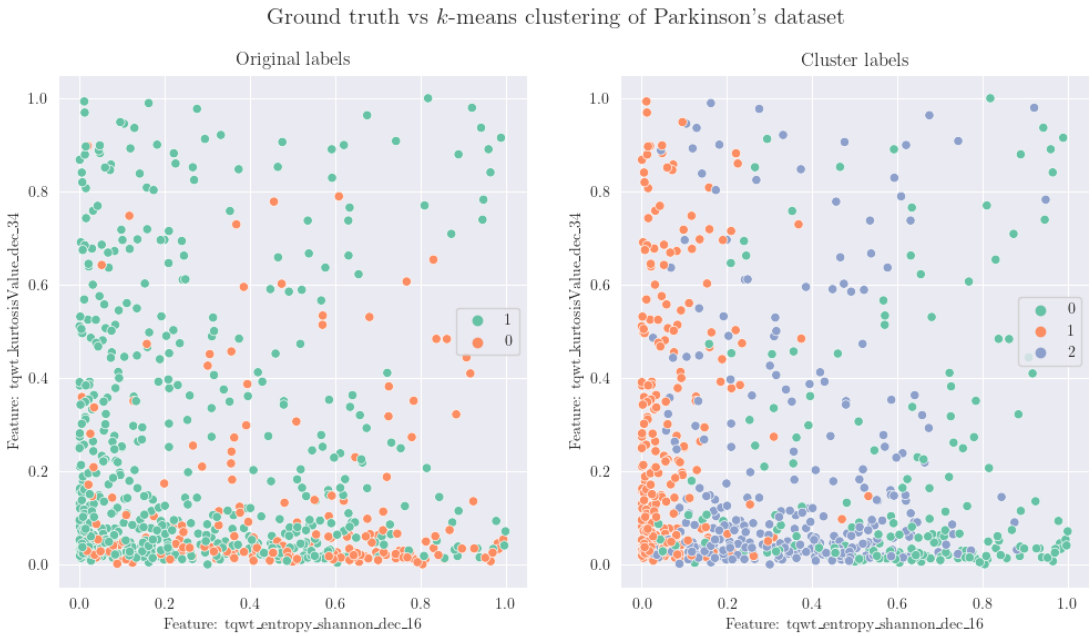

Figure 1: Ground truth vs k-means clustering of Parkinson's dataset