TÉCNICO LISBOA

Aprendizagem 2021/22

Homework I – Group 081

I. Pen-and-paper

Given the bivariate observations $\{\binom{1}{2}, \binom{-1}{1}, \binom{1}{0}\}$,

and the multivariate Gaussian mixture

$$\mathbf{u}_1 = \binom{2}{2}, \mathbf{u}_2 = \binom{0}{0}, \mathbf{\Sigma}_1 = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}, \mathbf{\Sigma}_2 = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}, \pi_1 = 0.5, \pi_2 = 0.5.$$

Pergunta 1:

1) [7v] Perform one epoch of the EM clustering algorithm and determine the new parameters. Indicate all calculus step by step (you can use a computer, however disclose intermediary steps).

EM clustering (descrição teórica)

```
Nota: EM clustering (em código):
Para confirmar os calculos e garantir a integridade dos resultados decidi fazer todos
os passos acima referidos utilizando um simples programa em python.O código acompanha
o decumento e está devidamente sinalizado.
```

- P(xn | ck) = N(x | **μk,Ek**) =

N(x | **μk,Ek**) = $\frac{1}{(2\cdot\pi)^{D/2}} \cdot \frac{1}{|\Sigma_k|^{1/2}} \cdot \exp\left(-\frac{1}{2}\cdot(\mathbf{x}_\eta - \boldsymbol{\mu}_k)^T \Sigma_k^{-1} \cdot (\mathbf{x}_\eta - \boldsymbol{\mu}_k)\right)$

Para calcular a probabilidade de um ponto (xk) pertencer a um certo cluster, segundo o algoritmo EM clustering, basta seguir a distribuição normal correspondente

```
def norm_calc(x, sigma, mu):
    return (1 / ((2 * math.pi) * np.linalg.det(sigma)**(0.5)) *
            math.e**(-0.5 * (
                (x - mu).transpose().dot(np.linalg.inv(sigma)).dot(x - mu)))
```

- P (ck = 1 , xn) = πk * N(x | **μk,Ek**)

Usando fórmulas estatisticas, não é dificil chegar a esta igualdade

```
xi_ck = PI_k * xi_mid_ck
```

- P (xn) = $\sum_{k=1}^{K} p(c_k = 1, \mathbf{x}_\eta)$

Após fazer os passos anteriores, para todos os clusters em vigor, basta agora somar todos

```
x1 = x1_c1 + x1_c2
```

- Fazemos agora uma média ponderada:

$$\frac{p(c_k = 1, \mathbf{x}_\eta)}{p(\mathbf{x}_\eta)}$$

Com isto conseguimos ter os **γ's**

```
gamma_kn = xn_ck / xn
>>> Nota: n1 = gamma_11 + gamma_12 + gamma_13
```

- Podemos agora atualizar os cluster's seguindo o seguinte método

Ck_new = $\frac{\gamma k1 * x1 + \gamma k2 * x2 + \gamma k3 * x3}{\gamma k1 + \gamma k2 + \gamma k3}$

```
(exemplo)
mu_1_new = (1 / n1) * (gamma_11 * X_1 + gamma_12 * X_2 + gamma_13 * X_3)
```

- Podemos também atualizar as matrizes de covariância

CovK_new = $\Sigma_k = \frac{1}{N_k} \cdot \sum_{\eta=1}^{N} \gamma(c_{\eta k}) \cdot (\mathbf{x}_\eta - \boldsymbol{\mu}_k) \cdot (\mathbf{x}_\eta - \boldsymbol{\mu}_k)^T$

```
(exemplo)
    sigma_1_new = (1 / n1) * (gamma_11 * ((X_1 - mu_1_new).dot(
        (X_1 - mu_1_new).transpose())) + gamma_12 * ((X_2 - mu_1_new).dot(
            (X_2 - mu_1_new).transpose())) + gamma_13 * ((X_3 - mu_1_new).dot(
                (X_3 - mu_1_new).transpose())))
```

- Teremos também que atualizar os pi's

  Basta fazer a média dos **γk's**

  Ou seja:

  Pi_k = $\frac{\gamma k1 * x1 + \gamma k2 * x2 + \gamma k3 * x3}{3}$

```
pi_1_new = (n1 / 3)
```

Com isto chegaremos aos resultados seguintes:

>>> Novos clusters:
mu_1 = [[0.75096381],[1.31149108]]
mu_2 = [[0.03438459], [0.77702808]]

>>> Novas matrizes de covariância:
sigma_1 = [[0.43605335, 0.07757255],[0.07757255 0.77845521]]
sigma_2 = [[0.9988177 ,-0.21530512],[-0.21530512 0.46747582]]

>>> Novos pi's:
pi_1 = 0.41718869
pi_2 = 0.58281131

Pergunta 2:

2) Given the updated parameters computed in previous question:
   a. [1.5v] perform a hard assignment of observations to clusters under a MAP assumption.
   b. [2.5v] compute the silhouette of the larger cluster using the Euclidean distance.

a) O objetivo é saber agora, qual dos clusters se adequa mais a cada ponto, dado que é hard assigment temos que decidir concretamte qual é o mais adequado

ci = max * P(ci | x) = $\frac{P(ci)*P(x|ci)}{P(x)}$

Por exemplo para x1:
X1 = (1, 2)
C1 - > 0.417 * 0.196
C2 -> 0.583 * 0.0135

C1 > C2
X1 irá ser hard assigment para C1

Fazer o mesmo para x2:
X2:
C1 = 0.417 * 0.0082
C2 = 0.583 * 0.143
Escolher C2

X3:

C1 = 0.417 * 0.0077

C2 = 0.583 * 0.104

Escolher C2

Pergunta 3:

Objetivo: Calcular Silhoete value

Como é que isso se faz?

S(X) = 1 - d(med_entre_cluster) / d(med_cluster_mais_prox)

Como saber qual é o 'larger cluster'

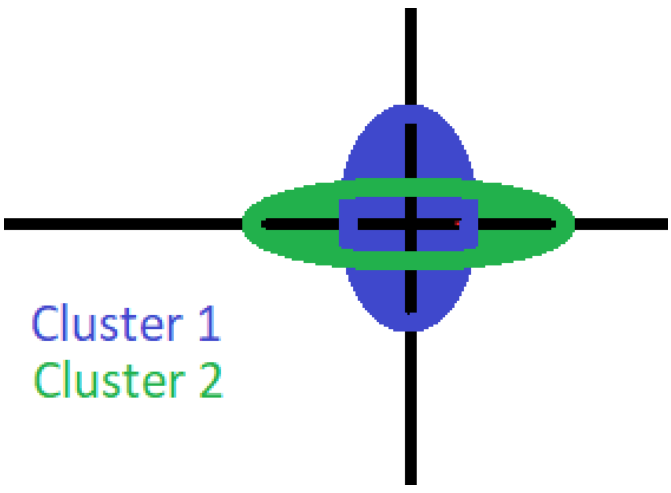
Basta ver pelas novas matrizes de covariância:

Relembrar:

>>> Novas matrizes de covariância:

sigma_1 = [[0.43605335, 0.07757255],[0.07757255 0.77845521]]

sigma_2 = [[0.9988177 ,-0.21530512],[-0.21530512 0.46747582]]



Para simplificar desconsiderámos as inclinações dos clusters, ainda assim gráficamente é dificil entender qual é o que tem uma área maior. Teremos por isso que usar a nossa intuição matemática para descobrir que o cluster dois é maior.

Justificação:

Os que contabilizam para a área:

0.43 < 0.47 e o.77 < 0.99

Garantimos assim que o cluster 2 é maior que o 1.

Agora basta aplicar a formula de S(x) no cluster 2

Para o cluster c2:

S(x2) = 1 – d(x2,x3) / d(x2,x1) = 0

S(x3) = 1 . d(x3,x2) / d(x3,x1) = -0.11805

S(c2) = 'média ponderada para cada ponto do cluster' = -0.11805 / 2 = 0.059

II. Programming and critical analysis

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.

Pergunta 1:

1) [4.5v] 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.

Silhouette score for seed 0: 0.11362027575179426

Silhouette score for seed 1: 0.11403554201377068

Silhouette score for seed 2: 0.11362027575179426

Purity score for seed 0: 0.7671957671957672

Purity score for seed 1: 0.7632275132275133

Purity score for seed 2: 0.7671957671957672

Pergunta 2:

2) [1.5v] What is causing the non-determinism?

O não determinismo é causado pelo simples facto de que a origem dos clusters varia de seed para seed.

Exemplo:

Na seed 0 o cluster 1 tem coordenadas (x1,y1), já na seed 1 terá coordenadas (x2,y2), com inicializações diferentes, a convergência de resultados será também diferente. Justificando assim o não determinismo.

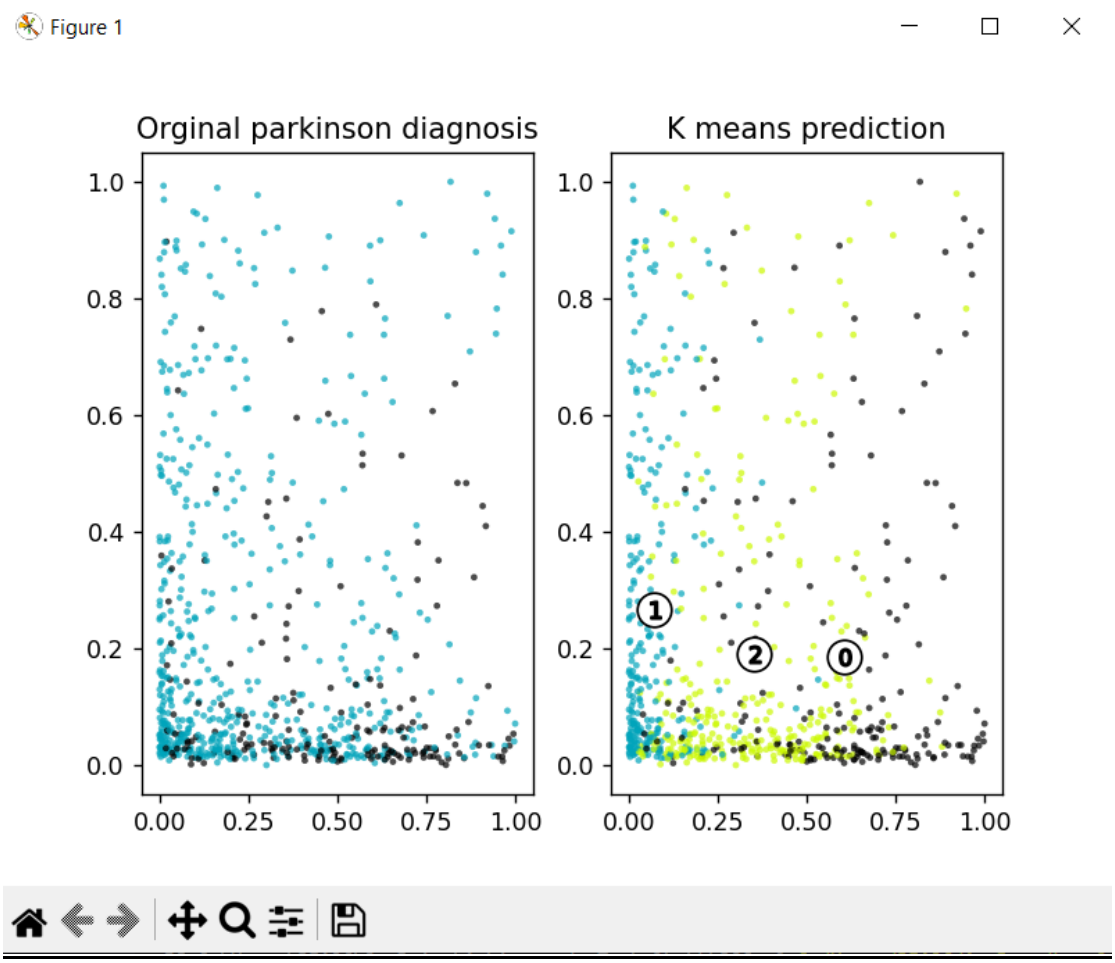
TÉCNICO LISBOA

Aprendizagem 2021/22

Homework I – Group 081

Pergunta 3:

3) [1.5v] 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.



Pergunta 4:

4) [1.5v] 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? Hint: explore the `DimReduction` notebook to be familiar with PCA in `sklearn`.

Para explicar 80% precisamos de pelo menos 31 componentes

III. APPENDIX

```python
from scipy.io.arff import loadarff
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
from sklearn.cluster import KMeans
from sklearn.linear_model import ridge_regression
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import MinMaxScaler
from sklearn import metrics
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score
import matplotlib.cm as cm
from sklearn.decomposition import PCA

def purity_score(y_true, y_pred):
    # compute contingency matrix (also called confusion matrix)
    contingency_matrix = metrics.cluster.contingency_matrix(y_true, y_pred)
    # return purity
    return np.sum(np.amax(contingency_matrix, axis=0)) / np.sum(contingency_matrix)

if __name__ == "__main__":

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

    # drop class
    X = df.drop('class', axis=1)
    y = df['class']

    # Normalize data using sklearn MinMaxScaler
    X =  MinMaxScaler().fit_transform(X)
    normalized_data = X


    seeds = [0, 1, 2]

    k_means_0 = KMeans(n_clusters=3, random_state=0)
    k_means_1 = KMeans(n_clusters=3, random_state=1)
    k_means_2 = KMeans(n_clusters=3, random_state=2)
```

```
    aplied_k_means_0 = k_means_0.fit(normalized_data)
    aplied_k_means_1 = k_means_1.fit(normalized_data)
    aplied_k_means_2 = k_means_2.fit(normalized_data)


    # Predict the cluster for each data point

    y_pred0 = aplied_k_means_0.predict(normalized_data)
    y_pred1 = aplied_k_means_1.predict(normalized_data)
    y_pred2 = aplied_k_means_2.predict(normalized_data)

    # Get the silhoeutte score for each seed

    s0 = silhouette_score(normalized_data, aplied_k_means_0.labels_)
    s1 = silhouette_score(normalized_data, aplied_k_means_1.labels_)
    s2 = silhouette_score(normalized_data, aplied_k_means_2.labels_)

    print("Silhouette score for seed 0: ", s0)
    print("Silhouette score for seed 1: ", s1)
    print("Silhouette score for seed 2: ", s2)

    # Get the purity score for each seed

    purity_0 = purity_score(y, y_pred0)
    purity_1 = purity_score(y, y_pred1)
    purity_2 = purity_score(y, y_pred2)

    print("Purity score for seed 0: ", purity_0)
    print("Purity score for seed 1: ", purity_1)
    print("Purity score for seed 2: ", purity_2)

    # Select the two input variables with highest variance on the MinMax normalized
data

    variancias = np.var(normalized_data, axis=0)

    sorted_var = sorted(variancias)

    top_1 = sorted_var[-1]
    top_2 = sorted_var[-2]

    top_1_index = np.where(variancias == top_1)
    top_2_index = np.where(variancias == top_2)


    fig, (axog,ax) = plt.subplots(1,2)
```

```
    colors = cm.nipy_spectral(y_pred0.astype(float) / 3)
    ax.scatter(
        X[:, top_1_index], X[:, top_2_index], marker=".", s=30, lw=0, alpha=0.7,
c=colors, edgecolor="k"
    )

    colors = cm.nipy_spectral(y.astype(float) / 3)
    axog.scatter(
        X[:, top_1_index], X[:, top_2_index], marker=".", s=30, lw=0, alpha=0.7,
c=colors, edgecolor="k"
    )
    axog.set_title("Orginal parkinson diagnosis")


    # Labeling the clusters
    centers = aplied_k_means_0.cluster_centers_
    # Draw white circles at cluster centers
    ax.scatter(
        centers[:, top_1_index],
        centers[:, top_2_index],
        marker="o",
        c="white",
        alpha=1,
        s=200,
        edgecolor="k",
    )

    for i, c in enumerate(centers):
        ax.scatter(c[top_1_index], c[top_2_index], marker="$%d$" % i, alpha=1, s=50,
edgecolor="k")

    ax.set_title("K means prediction")
    plt.show()


    # PCA Components

    pca = PCA(n_components=0.8, svd_solver='full')
    pca.fit(X)

    # Get the number of components (length of the array)
    print("Components:\n",len(pca.components_))
```

END