Unsupervised Learning – a3rl

André Filipe Frade Guerra

Novembro/2022

Conteúdo

Introdução

Neste relatório irá ser explicado resumidamente as classes criadas, funcionalidade dos métodos criados e por fim a execução e discussão dos resultados dos exercícios.

Este trabalho tem como objetivo implementar vários algoritmos que tratam informação num conjunto de pontos por clusters.

Classes e métodos

Para desenvolver os exercícios da terceira prática da disciplina de Introdução à Aprendizagem Automática foi desenvolvido um código em python usando o vsCode para realizar a mesma.

Presente em todas as classes é criada a população de pontos que vai ser trabalhada:

```
runs = 10
alfa = 0.0005  #0.00001
howManyPoints = 500

mean = [3, 3]
cov = [[1 , 0], [0 , 1]]
setA = np.random.multivariate_normal(mean, cov, howManyPoints).T
mean = [-3, -3]
cov = [[2, 0], [0 , 5]]
setB = np.random.multivariate_normal(mean, cov, howManyPoints).T

setC = np.concatenate((setA, setB), axis = 1)
setC = setC.T
np.random.shuffle(setC)
setC = setC.T

setAX = setA[0]
setAY = setA[1]

setBX = setB[0]
setBY = setB[1]

setCX = setC[0]
setCY = setC[1]
```

Exercício 1

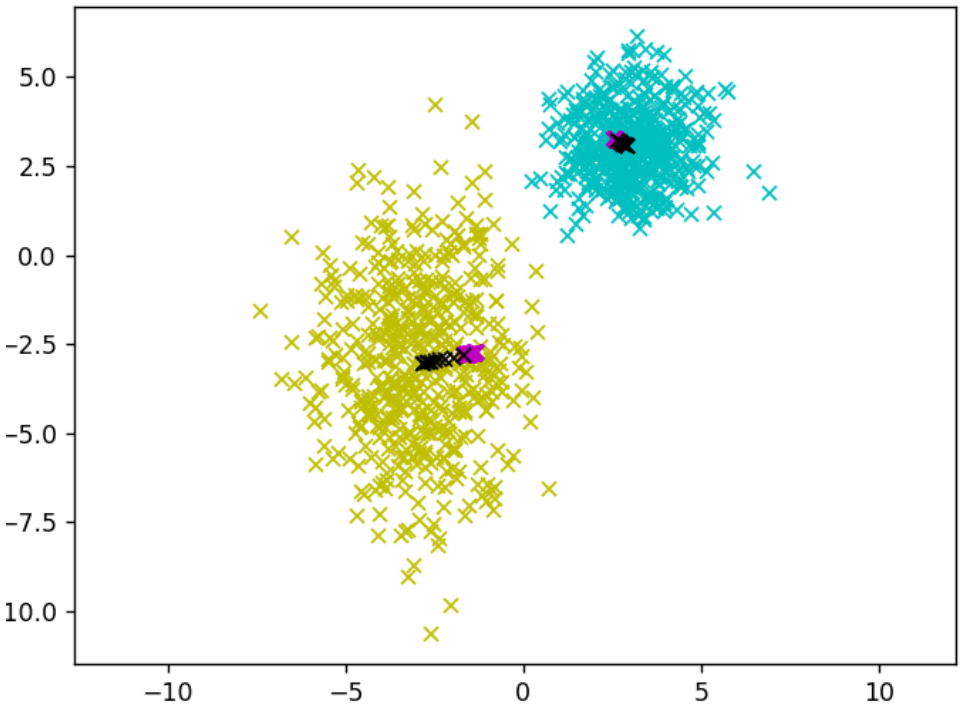
Presente no ficheiro exer1.py, este ficheiro começa por criar os pontos anteriormente descrito, e escolhe 2 pontos aleatórios:

```
  setRX = []
  setRX.append(setC[0][random.randrange(0,howManyPoints-1)])
  setRX.append(setC[0][random.randrange(0,howManyPoints-1)])
  setRY = []
  setRY.append(setC[0][random.randrange(0,howManyPoints-1)])
  setRY.append(setC[0][random.randrange(0,howManyPoints-1)])
  setR = [setRX, setRY]
```

De seguida, encontra-se o algoritmo implementado, que vai fazer andar os dois pontos escolhidos para dentro do kluster mais próximo:

```
setRXEndEachPassageP1 = []
setRYEndEachPassageP1 = []
setRXEndEachPassageP2 = []
setRYEndEachPassageP2 = []

for run in range(runs):
    for i in range(howManyPoints*2):
        if math.dist([setCX[i], setCY[i]], [setRX[0], setRY[0]]) < math.dist([setCX[i], setCY[i]], [setRX[1], setRY[1]]):
            setRX[0] = (1 - alfa) * setRX[0] + alfa * setCX[i]
            setRY[0] = (1 - alfa) * setRY[0] + alfa * setCY[i]
            if run == 0:
                setRXFirstPassage.append(setRX[0])
                setRYFirstPassage.append(setRY[0])
            #print("closer to p1")
        else:
            setRX[1] = (1 - alfa) * setRX[1] + alfa * setCX[i]
            setRY[1] = (1 - alfa) * setRY[1] + alfa * setCY[i]
            if run == 0:
                setRXFirstPassage.append(setRX[1])
                setRYFirstPassage.append(setRY[1])
            #print("closer to p2")


    setRXEndEachPassageP1.append(setRX[0])
    setRYEndEachPassageP1.append(setRY[0])
    setRXEndEachPassageP2.append(setRX[1])
    setRYEndEachPassageP2.append(setRY[1])

    #plt.plot(setAX, setAY, 'x', color='c')
    #plt.plot(setBX, setBY, 'x', color='y')
    #plt.plot(setRX, setRY, 'x', color='r')
    #plt.axis("equal")
    #plt.show()

plt.plot(setAX, setAY, 'x', color='c')
plt.plot(setBX, setBY, 'x', color='y')
plt.plot(setRXFirstPassage, setRYFirstPassage, 'x', color='m')
plt.plot(setRXEndEachPassageP1, setRYEndEachPassageP1, 'x', color='k')
plt.plot(setRXEndEachPassageP2, setRYEndEachPassageP2, 'x', color='k')
plt.axis("equal")
plt.show()

```

No fim mostra o resultado da localização dos pontos e o percurso que fizeram:



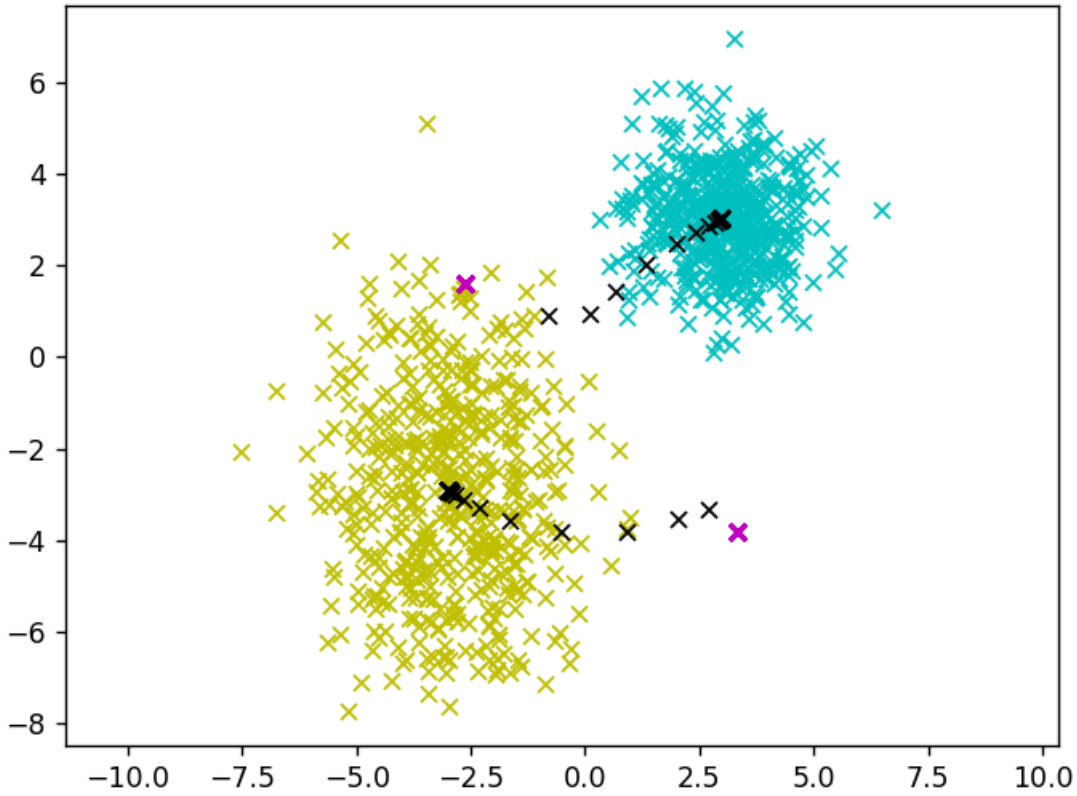
Neste gráfico a roxo, podemos ver a deslocação na primeira interação do ciclo, e a preto as restantes deslocações. Por fim podemos ver que os pontos se encontram mais perto do “meio” dos clusters.

Exercício 2

Presente no ficheiro exer2.py, este ficheiro começa por declarar vários valores iniciais que definem o algoritmo. Com a diferença do primeiro algoritmo, este apenas atualiza a posição do ponto após percorrer todos os pontos:

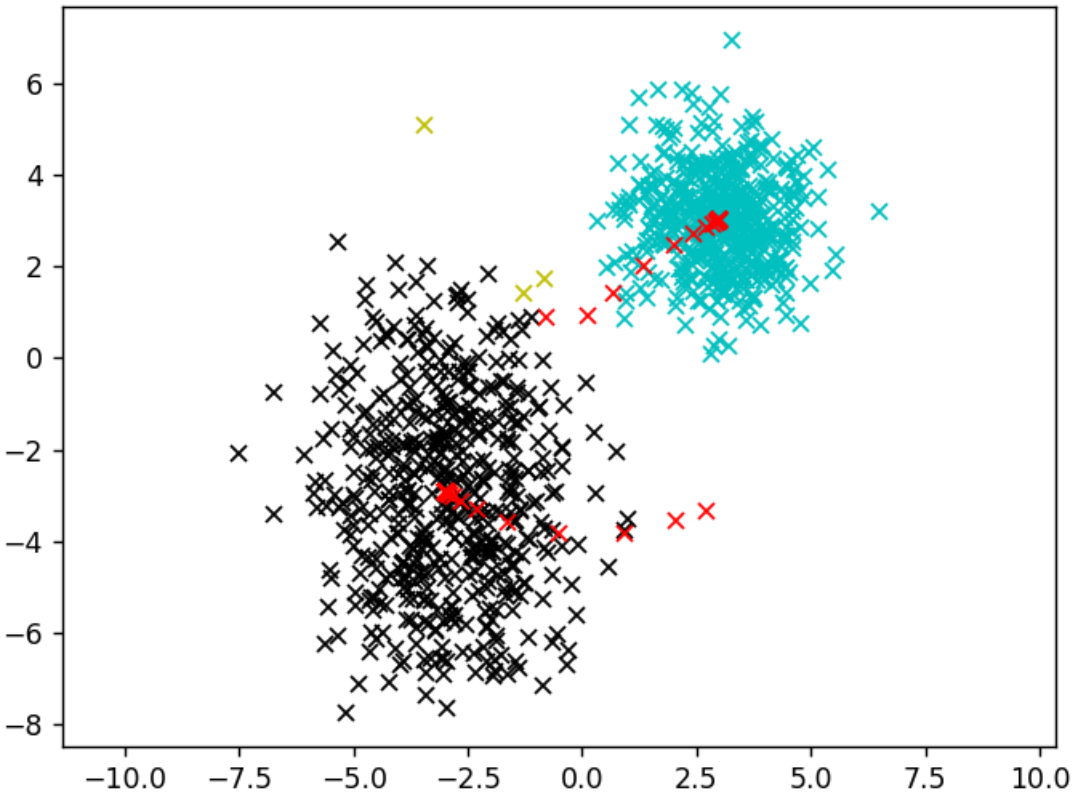
```
for run in range(runs):
    dXP1 = 0
    dYP1 = 0
    dXP2 = 0
    dYP2 = 0
    for i in range(howManyPoints*2):
        if math.dist([setCX[i], setCY[i]], [setRX[0], setRY[0]]) < math.dist([setCX[i], setCY[i]], [setRX[1], setRY[1]]):
            dXP1 = dXP1 + (setCX[i] - setRX[0])
            dYP1 = dYP1 + (setCY[i] - setRY[0])
            if run == 0:
                setRXFirstPassage.append(setRX[0])
                setRYFirstPassage.append(setRY[0])
        else:
            dXP2 = dXP2 + (setCX[i] - setRX[1])
            dYP2 = dYP2 + (setCY[i] - setRY[1])
            if run == 0:
                setRXFirstPassage.append(setRX[1])
                setRYFirstPassage.append(setRY[1])

    setRX[0] = setRX[0] + (alfa / howManyPoints) * dXP1
    setRY[0] = setRY[0] + (alfa / howManyPoints) * dYP1
    setRX[1] = setRX[1] + (alfa / howManyPoints) * dXP2
    setRY[1] = setRY[1] + (alfa / howManyPoints) * dYP2

    setRXEndEachPassageP1.append(setRX[0])
    setRYEndEachPassageP1.append(setRY[0])
    setRXEndEachPassageP2.append(setRX[1])
    setRYEndEachPassageP2.append(setRY[1])

    #plt.plot(setAX, setAY, 'x', color='c')
    #plt.plot(setBX, setBY, 'x', color='y')
    #plt.plot(setRX, setRY, 'x', color='r')
    #plt.axis("equal")
    #plt.show()

plt.plot(setAX, setAY, 'x', color='c')
plt.plot(setBX, setBY, 'x', color='y')
plt.plot(setRXFirstPassage, setRYFirstPassage, 'x', color='m')
plt.plot(setRXEndEachPassageP1, setRYEndEachPassageP1, 'x', color='k')
plt.plot(setRXEndEachPassageP2, setRYEndEachPassageP2, 'x', color='k')
plt.axis("equal")
plt.show()
```

No fim mostra o resultado da localização dos pontos e o percurso que fez:



Após a conclusão desta parte do código, o restante, verifica se existem pontos do kluster que se encontram mais próximos do outro kluster (representados a amarelo):

```
97   setXCloserToP1SetA = []
98   setYCloserToP1SetA = []
99   setXCloserToP1SetB = []
100  setYCloserToP1SetB = []
101
102  setXCloserToP2SetA = []
103  setYCloserToP2SetA = []
104  setXCloserToP2SetB = []
105  setYCloserToP2SetB = []
106  for i in range(howManyPoints):
107      #setA
108      if math.dist([setRX[0], setRY[0]], [setAX[i], setAY[i]]) < math.dist([setRX[1], setRY[1]], [setAX[i], setAY[i]]):
109          setXCloserToP1SetA.append(setAX[i])
110          setYCloserToP1SetA.append(setAY[i])
111      else:
112          setXCloserToP2SetA.append(setAX[i])
113          setYCloserToP2SetA.append(setAY[i])
114      #setB
115      if math.dist([setRX[0], setRY[0]], [setBX[i], setBY[i]]) < math.dist([setRX[1], setRY[1]], [setBX[i], setBY[i]]):
116          setXCloserToP1SetB.append(setBX[i])
117          setYCloserToP1SetB.append(setBY[i])
118      else:
119          setXCloserToP2SetB.append(setBX[i])
120          setYCloserToP2SetB.append(setBY[i])
121
122  plt.plot(setXCloserToP1SetA, setYCloserToP1SetA, 'x', color='c')
123  plt.plot(setXCloserToP1SetB, setYCloserToP1SetB, 'x', color='y')
124  plt.plot(setXCloserToP2SetA, setYCloserToP2SetA, 'x', color='m')
125  plt.plot(setXCloserToP2SetB, setYCloserToP2SetB, 'x', color='k')
126  plt.plot(setRXEndEachPassageP1, setRYEndEachPassageP1, 'x', color='r')
127  plt.plot(setRXEndEachPassageP2, setRYEndEachPassageP2, 'x', color='r')
128  plt.axis("equal")
129  plt.show()
```

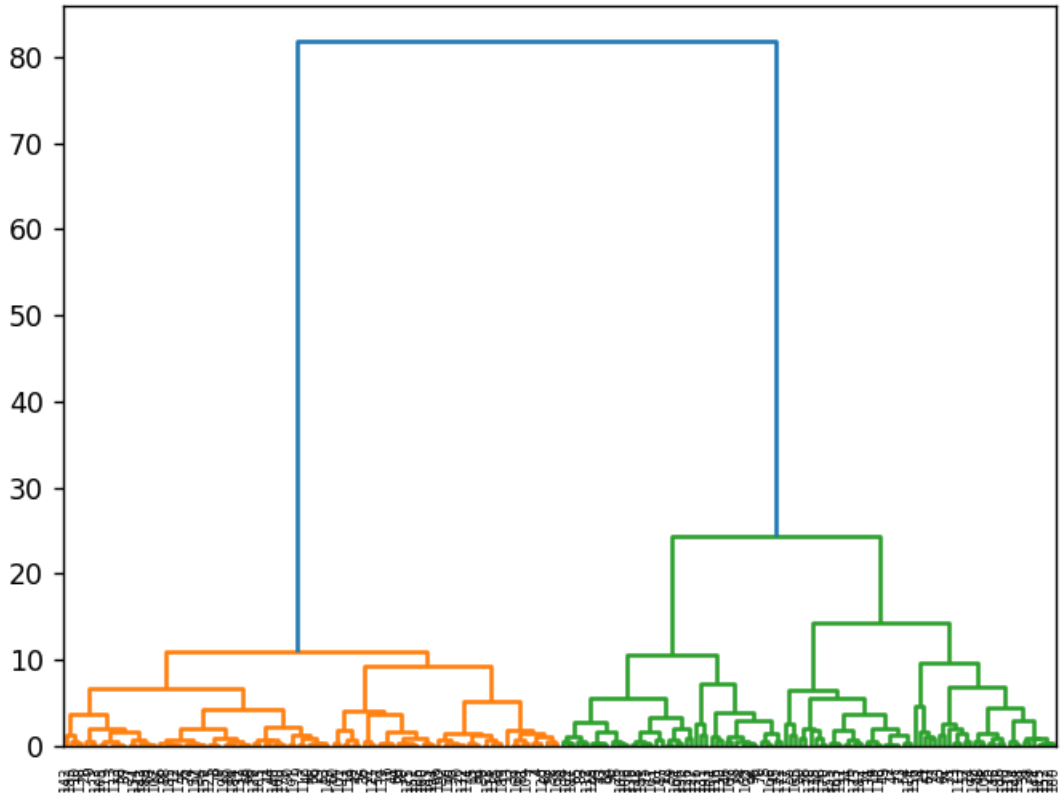

6
4
2
0
−2
−4
−6
−8
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

Exercício 3

Presente no ficheiro exer3.py, este ficheiro começa por declarar vários valores iniciais que definem o algoritmo.

Inicialmente foi implementado um algoritmo que mostra o gráfico em árvore:

```
32  data = list(zip(setCX, setCY))
33  linkage_data = linkage(data, method='ward', metric='euclidean')
34  dendrogram(linkage_data)
35  plt.show()
```



E de seguida a implementação do algoritmo, sem usar bibliotecas externas, sem mostrar visualmente a árvore:

```
47  while len(setCX) > 1:
48      distance = 10000
49      for i in range(len(setCX)-1):
50          for j in range(len(setCX)-1):
51              if math.dist([setCX[j], setCY[j]], [setCX[i], setCY[i]]) < distance and setCX[j] != setCX[i] and setCY[j] != setCY[i]:
52                  distance = math.dist([setCX[j], setCY[j]], [setCX[i], setCY[i]])
53                  positionClosestX1 = setCX[j]
54                  positionClosestY1 = setCY[j]
55                  positionClosestX2 = setCX[i]
56                  positionClosestY2 = setCY[i]
57                  auxI = i
58                  auxJ = j
59
60      setCX = np.delete(setCX, i)
61      setCY = np.delete(setCY, i)
62      setCX = np.delete(setCX, j)
63      setCY = np.delete(setCY, j)
64      setCX = np.append(setCX, [(positionClosestX1 + positionClosestX2) / 2])
65      setCY = np.append(setCY, [(positionClosestY1 + positionClosestY2) / 2])
66      px.append((positionClosestX1 + positionClosestX2) / 2)
67      py.append((positionClosestY1 + positionClosestY2) / 2)
```

Exercício 4

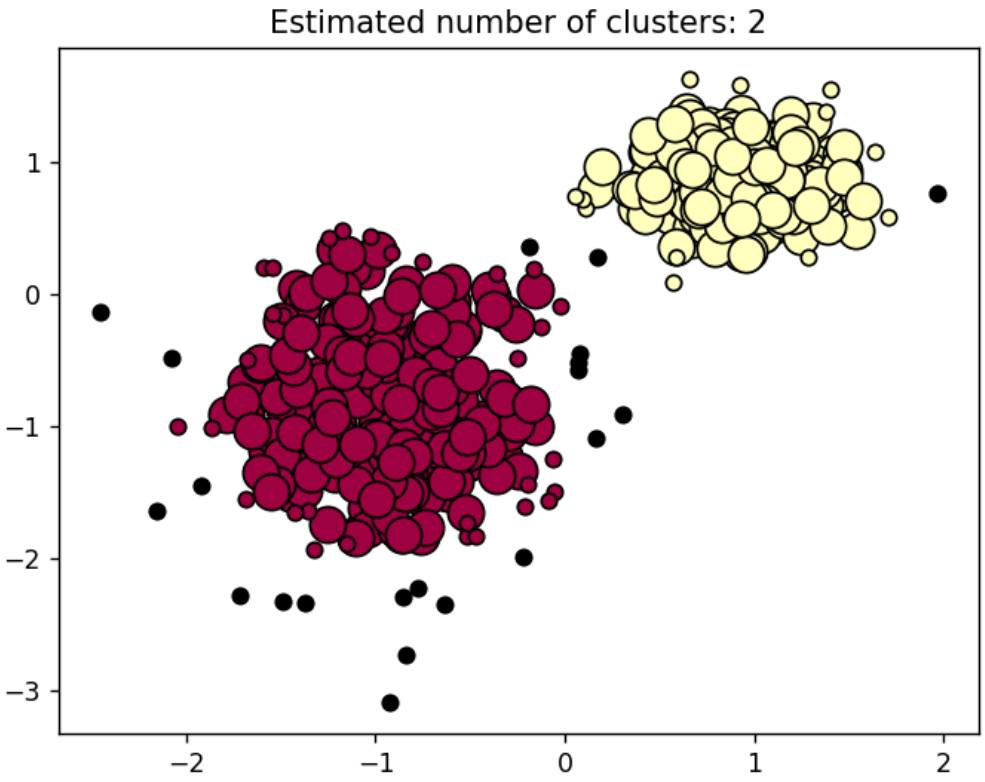
Presente no ficheiro exer4.py, este ficheiro começa por declarar vários valores iniciais que definem o algoritmo.

Inicialmente temos uma implementação do algoritmo que guarda nas variáveis “klusterFinalX” e “klusterFinalY”, os pontos de cada kluster encontrado:

```
39  klusterFinalX = []
40  klusterFinalY = []
41
42  while len(setCX) > 0:
43      indexToRemove = []
44      klusterAuxX = []
45      klusterAuxY = []
46      r = randrange(0, len(setCX))
47      pX = setCX[r]
48      pY = setCY[r]
49      for i in range(len(setCX)):
50          if math.dist([pX, pY], [setCX[i], setCY[i]]) <= alfaDistance:
51              howManyPointsTotal -= 1
52              klusterAuxX.append(setCX[i])
53              klusterAuxY.append(setCY[i])
54              pX = setCX[i]
55              pY = setCY[i]
56              indexToRemove.append(i)
57      setCXaux = setCX
58      setCYaux = setCY
59      for j in range(len(indexToRemove)):
60          setCXaux = np.delete(setCXaux, indexToRemove[j])
61          setCYaux = np.delete(setCYaux, indexToRemove[j])
62
63      setCX = setCXaux
64      setCY = setCYaux
65
66      klusterFinalX.append(klusterAuxX)
67      klusterFinalY.append(klusterAuxY)
```

E por fim uma implementação que usa bibliotecas externas para realizar o algoritmo, que mostra visualmente o resultado de quantos clusters foram encontrados:

```
################## SECOND implementation
labels_true = []
points = []
for i in range(howManyPoints):
    points.append([setCX[i], setCY[i]])
    labels_true.append(0)

X = StandardScaler().fit_transform(points)

db = DBSCAN(eps=0.3, min_samples=10).fit(X)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_

# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)

print("Estimated number of clusters: %d" % n_clusters_)
print("Estimated number of noise points: %d" % n_noise_)
print("Homogeneity: %0.3f" % metrics.homogeneity_score(labels_true, labels))
print("Completeness: %0.3f" % metrics.completeness_score(labels_true, labels))
print("V-measure: %0.3f" % metrics.v_measure_score(labels_true, labels))
print("Adjusted Rand Index: %0.3f" % metrics.adjusted_rand_score(labels_true, labels))
print(
    "Adjusted Mutual Information: %0.3f"
    % metrics.adjusted_mutual_info_score(labels_true, labels)
)
unique_labels = set(labels)
colors = [plt.cm.Spectral(each) for each in np.linspace(0, 1, len(unique_labels))]
for k, col in zip(unique_labels, colors):
    if k == -1:
        col = [0, 0, 0, 1]
    class_member_mask = labels == k
    xy = X[class_member_mask & core_samples_mask]
    plt.plot(
        xy[:, 0],
        xy[:, 1],
        "o",
        markerfacecolor=tuple(col),
        markeredgecolor="k",
        markersize=14,
    )
    xy = X[class_member_mask & ~core_samples_mask]
    plt.plot(
        xy[:, 0],
        xy[:, 1],
        "o",
        markerfacecolor=tuple(col),
        markeredgecolor="k",
        markersize=6,
    )
plt.title("Estimated number of clusters: %d" % n_clusters_)
plt.show()
```


Estimated number of clusters: 2
1
0
−1
−2
−3
−2
−1
0
1
2