

homework 3

problem 1

K-means convergence

[10 points]

In k -means, we minimize the loss function:

$$L(\boldsymbol{\mu}, \boldsymbol{\alpha}) = \sum_{k=1}^K \sum_{i=1}^n \|x_i - \mu_k\|_2^2 \mathbb{1}\{\alpha_i = k\},$$

where,

- x_i is a data point with p dimensions,
- n is the total number of data points,
- μ_k is the center of cluster k and is of dimension p ,
- $\mathbb{1}\{\cdot\}$ is an indicator function. That is, it is equal to one if the test is true, zero otherwise,
- α_i is the label of the i -th data point.

1a

(a) Let $\alpha_i^{(t)}$ be the assignment in iteration (t) . Show that

$$L(\boldsymbol{\mu}, \boldsymbol{\alpha}^{(t+1)}) \leq L(\boldsymbol{\mu}, \boldsymbol{\alpha}^{(t)}).$$

answer 1a

given: $\mathcal{J}^{(t)}$ = assignment in iteration (t)

$$L(\mu, \mathcal{J}) = \sum_{k=1}^K \sum_{i=1}^n \|x_i - \mu_k\|_2^2 \mathbb{1}_{\{\mathcal{J}_i = k\}}$$

prove: $L(\mu, \mathcal{J}^{(t)}) \geq L(\mu, \mathcal{J}^{(t+1)})$

Let: $\mathcal{J}^{(t)} = \arg \min_{a \in [1, k]} \|x_i - \mu_a\|_2^2$

$$L(\mu, \mathcal{J}^{(t+1)}) - L(\mu, \mathcal{J}^{(t)}) \leq 0$$

$$\sum_{k=1}^K \sum_{i=1}^n \left[\|x_i - \mu_k\|_2^2 \mathbb{1}_{\{\mathcal{J}_i^{(t+1)} = k\}} - \|x_i - \mu_k\|_2^2 \mathbb{1}_{\{\mathcal{J}_i^{(t)} = k\}} \right] \leq 0$$

We can see that the loss function during subsequent iterations gets assigned to a closer cluster. This leads to the total sum of distances from clusters decreasing. Therefore after each time step our loss function decreases in value or remains the same.

1b

- (b) After the assignment, k -means will do a refitting of μ conditional on the latest assignments. Show that the update

$$\mu_k = \frac{1}{\sum_i^n \mathbb{1}_{\{\alpha_i = k\}}} \sum_{i=1}^n x_i \mathbb{1}_{\{\alpha_i = k\}}$$

is the best you can do given this loss function. Hint: one way is to use those weird derivative things.

answer 1b

$$\mu_k = \frac{1}{\sum_{i=1}^n \mathbb{1}_{\{d_i=k\}}} \sum_{i=1}^n x_i \mathbb{1}_{\{d_i=k\}}$$

$$L(\mu_k, d) = \sum_{k=1}^K \sum_{i=1}^n \|x_i - \mu_k\|_2^2 \mathbb{1}_{\{d_i=k\}}$$

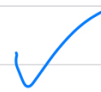
set
derivative
= 0

$$\frac{\partial L(\mu_k, d)}{\partial \mu_k} = \frac{\partial}{\partial \mu_k} \left[\sum_{k=1}^K \sum_{i=1}^n \|x_i - \mu_k\|_2^2 \mathbb{1}_{\{d_i=k\}} \right] = 0$$

$$0 = \cancel{\sum_{i=1}^n} (x_i - \mu_k) \mathbb{1}_{\{d_i=k\}}$$

$$0 = \sum_{i=1}^n x_i \mathbb{1}_{\{d_i=k\}} - \mu_k \sum_{i=1}^n \mathbb{1}_{\{d_i=k\}}$$

$$\mu_k = \frac{\sum_{i=1}^n x_i \mathbb{1}_{\{d_i=k\}}}{\sum_{i=1}^n \mathbb{1}_{\{d_i=k\}}}$$



problem 2

Soft k -means updates

Note: you may use write code for this problem if you please, but **you must know how to do this by hand**. Please refer to the notation in Chapter 8, section *Soft k -means Clustering*. Consider the following data in two dimensions:

data ID	x_{i1}	x_{i2}
1	0.1	0.2
2	0.2	0.1
3	0.3	0
4	1	1.2
5	0.8	1
6	9	0.1

and the following centers:

cluster ID	μ_{i1}	μ_{i2}
1	0.1	0.9
2	0.5	0
3	0.9	0.5

2a

- (a) Using the **partition function** and $\beta = 0.5$ compute the E-step and show the Hidden-Matrix.

answer 2a

$$HiddenMatrix_{i,j} = \frac{e^{-\beta \cdot d(Data_j, x_i)}}{\sum_{\text{all centers } x_i} e^{-\beta \cdot d(Data_j, x_i)}}$$

$$Force_{i,j} = e^{-\beta \cdot \text{distance}(\text{Data}_j, \text{Center}_i)}$$

iffness parameter.

$$HiddenMatrix_{ij} := \frac{Force_{i,j}}{\sum_{\text{all centers } j} Force_{i,j}}$$

```
48 #! /usr/bin/env python3
47
46 import math
45
44 beta = 0.5
43
42 def eucDistance(p1,p2):
41     x1, y1 = p1
40     x2, y2 = p2
39     return math.sqrt((x1-x2)**2+(y1-y2)**2)
38
37 def hiddenMatrix(data, clusters):
36     global beta
35     hidden = []
34     for d in data:
33         sigma = 0
32         forces = []
31         for cluster in clusters:
30             force = math.exp(-beta * eucDistance((d[0],d[1]),(cluster[0], cluster[1])))
29             sigma += force
28             forces.append(force)
27         norm = []
26         for force in forces:
25             norm.append(force/sigma)
24         hidden.append(norm)
23     return hidden
22
21 d = [
20     [0.1,0.2],
19     [0.2,0.1],
18     [0.3,0],
17     [1.1,2],
16     [0.8,1],
15     [9,0.1]
14 ]
13
12 c = [
11     [0.1,0.9],
10     [0.5,0],
9     [0.9,0.5]
8 ]
7
6 h = hiddenMatrix(d,c)
5
4 print('cluster id / data ID')
3 print(1,2,3,sep='\t')
2 for i, line in enumerate(h):
1     print(i+1,line)
49
~
~
1 2a.py [python] 100% 49:0
) ./2a.py hw3 via v3.9.7 11:10:03
cluster id / data ID
1 2 3
1 [0.3267511728523752, 0.3707737892616577, 0.3024750378859669]
2 [0.3050994280684201, 0.3898011438631598, 0.3050994280684201]
3 [0.2850840451215805, 0.4090188548359777, 0.30589710004244164]
4 [0.33700778468570974, 0.28271736237522366, 0.3802748529390666]
5 [0.33914505262395894, 0.28656516573360846, 0.3742897816424327]
6 [0.26634685780600265, 0.331108689646737, 0.4025444525472604]
) hw3 via v3.9.7 11:10:30

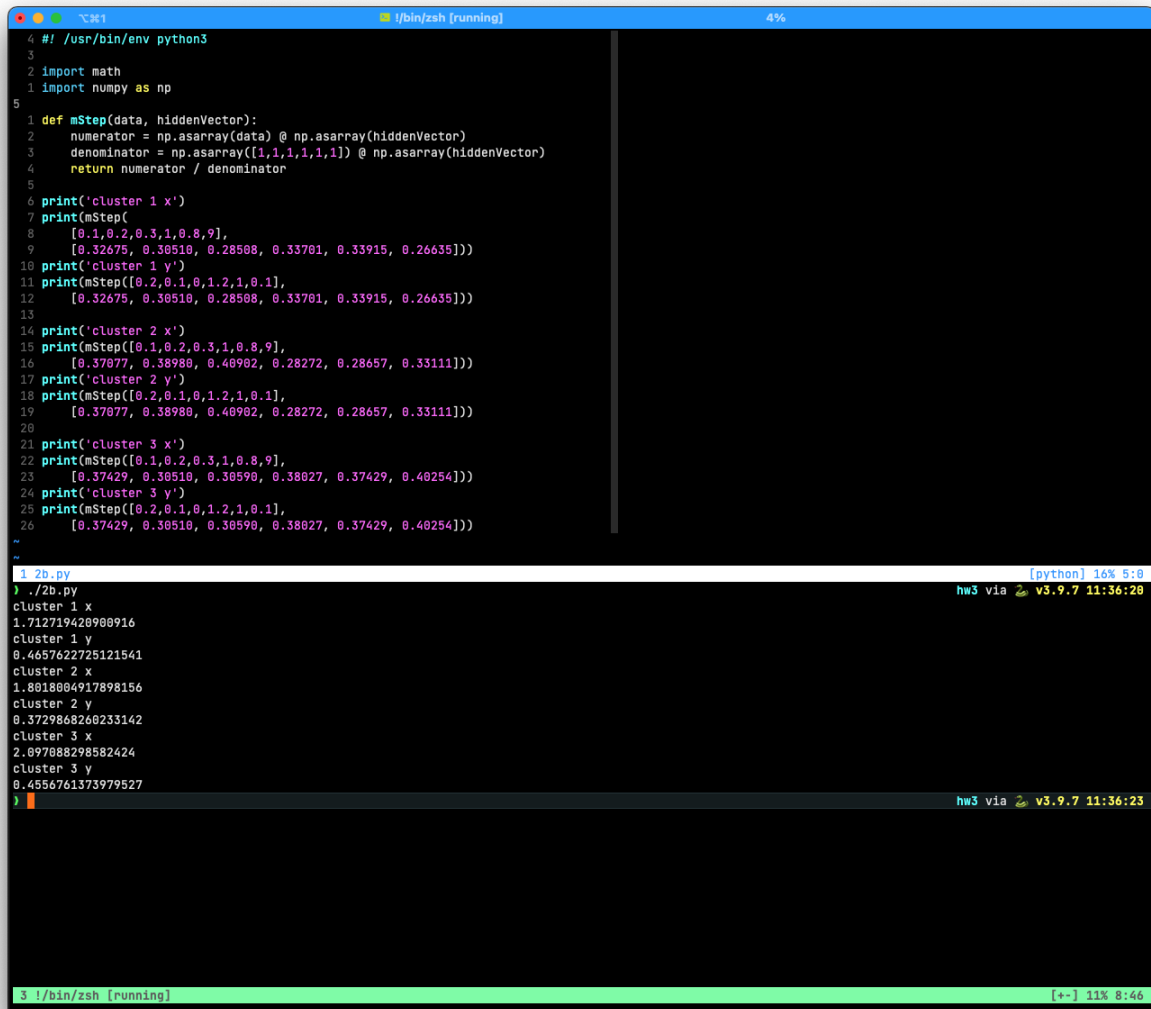
3 !/bin/zsh [running] [++] 10% 1:1
```

2b

(b) Using the assignments you made in (a), compute the M-step.

answer 2b

```
def mStep(data, hiddenVector):
    numerator = np.asarray(data) @ np.asarray(hiddenVector)
    denominator = np.asarray([1,1,1,1,1]) @ np.asarray(hiddenVector)
    return numerator / denominator
```



```
4 #! /usr/bin/env python3
3
2 import math
1 import numpy as np
5
1 def mStep(data, hiddenVector):
2     numerator = np.asarray(data) @ np.asarray(hiddenVector)
3     denominator = np.asarray([1,1,1,1,1]) @ np.asarray(hiddenVector)
4     return numerator / denominator
5
6 print('cluster 1 x')
7 print(mStep(
8     [0.1,0.2,0.3,1,0.8,9],
9     [0.32675, 0.30510, 0.28508, 0.33701, 0.33915, 0.26635]))
10 print('cluster 1 y')
11 print(mStep([0.2,0.1,0.1,2,1,0.1],
12     [0.32675, 0.30510, 0.28508, 0.33701, 0.33915, 0.26635]))
13
14 print('cluster 2 x')
15 print(mStep([0.1,0.2,0.3,1,0.8,9],
16     [0.37077, 0.38980, 0.40902, 0.28272, 0.28657, 0.33111]))
17 print('cluster 2 y')
18 print(mStep([0.2,0.1,0.1,2,1,0.1],
19     [0.37077, 0.38980, 0.40902, 0.28272, 0.28657, 0.33111]))
20
21 print('cluster 3 x')
22 print(mStep([0.1,0.2,0.3,1,0.8,9],
23     [0.37429, 0.30510, 0.30590, 0.38027, 0.37429, 0.40254]))
24 print('cluster 3 y')
25 print(mStep([0.2,0.1,0.1,2,1,0.1],
26     [0.37429, 0.30510, 0.30590, 0.38027, 0.37429, 0.40254]))
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

1 2b.py [python] 16% 5:0

cluster 1 x hw3 via v3.9.7 11:36:20

1.71271942090916

cluster 1 y

0.4657622725121541

cluster 2 x

1.8018084917898156

cluster 2 y

0.3729868260233142

cluster 3 x

2.097888298582424

cluster 3 y

0.4556761373979527

hw3 via v3.9.7 11:36:23

3 /bin/zsh [running] [+-] 11% 8:46

2c

(c) Using the Newtonian inverse-square law of gravitation, compute the E-step.

answer 2c

```
15 #! /usr/bin/env python3
16 import math
17
18 def distance(p1,p2):
19     x1, y1 = p1
20     x2, y2 = p2
21     return 1/ ( (x1-x2)**2 + (y1-y2)**2 )
22
23 def hiddenMatrix(data, clusters):
24     global beta
25     hidden = []
26     for d in data:
27         sigma = 0
28         forces = []
29         for cluster in clusters:
30             force = distance((d[0],d[1]),(cluster[0], cluster[1]))
31             sigma += force
32             forces.append(force)
33         norm = []
34         for force in forces:
35             norm.append(force/sigma)
36         hidden.append(norm)
37     return hidden
38
39 d = [
40     [0.1,0.2],
41     [0.2,0.1],
42     [0.3,0],
43     [1,1.2],
44     [0.8,1],
45     [9,0.1]
46 ]
47
48 c = [
49     [0.1,0.9],
50     [0.5,0],
51     [0.9,0.5]
52 ]
53
54 h = hiddenMatrix(d,c)
55
56 for line in h:
57     print(line)
58
59 ~
60 ~
61 1 2c.py [python] 37% 16:66
62 [0.24264583679574545, 0.5944823001495761, 0.1628718630546784] hw3 via v3.9.7 11:43:23
63 [0.11764705882352938, 0.7647058823529412, 0.11764705882352944]
64 [0.04229502513433871, 0.8987692841046975, 0.05893569076096377]
65 [0.3000710227272727, 0.15980113636363635, 0.5401278409090909]
66 [0.29570116861435725, 0.13564273789649417, 0.5686560934891486]
67 [0.301284974459967, 0.3329311543125985, 0.36578387122743455]
68 ]
69
70 2 !/bin/zsh [running] [v3.9.7] 18% 1:56
```

```
cluster id / data ID
1 2 3
1 [0.24264583679574545, 0.5944823001495761, 0.1628718630546784]
2 [0.11764705882352938, 0.7647058823529412, 0.11764705882352944]
3 [0.04229502513433871, 0.8987692841046975, 0.05893569076096377]
4 [0.3000710227272727, 0.15980113636363635, 0.5401278409090909]
5 [0.29570116861435725, 0.13564273789649417, 0.5686560934891486]
6 [0.301284974459967, 0.3329311543125985, 0.36578387122743455]
```

2d

(d) Using the assignments you made in (c), compute the M-step.

answer 2d

```
1 #! /usr/bin/env python3
2 import math
3 import numpy as np
4
5 def mStep(data, hiddenVector):
6     numerator = np.asarray(data) @ np.asarray(hiddenVector)
7     denominator = np.asarray([1,1,1,1,1]) @ np.asarray(hiddenVector)
8     return numerator / denominator
9
10 print('cluster 1 x')
11 print(mStep([0.1,0.2,0.3,1,0.8,9],
12     [0.24265, 0.11765, 0.04229, 0.30807, 0.29570, 0.30128]))
13 print('cluster 1 y')
14 print(mStep([0.2,0.1,0.1,2,1,0.1],
15     [0.24265, 0.11765, 0.04229, 0.30807, 0.29570, 0.30128]))
16
17 print('cluster 2 x')
18 print(mStep([0.1,0.2,0.3,1,0.8,9],
19     [0.59448, 0.76470, 0.89877, 0.15980, 0.13564, 0.33293]))
20 print('cluster 2 y')
21 print(mStep([0.2,0.1,0.1,2,1,0.1],
22     [0.59448, 0.76470, 0.89877, 0.15980, 0.13564, 0.33293]))
23
24 print('cluster 3 x')
25 print(mStep([0.1,0.2,0.3,1,0.8,9],
26     [0.16287, 0.11765, 0.05894, 0.54013, 0.56866, 0.36578]))
27 print('cluster 3 y')
28 print(mStep([0.2,0.1,0.1,2,1,0.1],
29     [0.16287, 0.11765, 0.05894, 0.54013, 0.56866, 0.36578]))
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

```
1 2d.py
2 ./2d.py
3 cluster 1 x
4 2.5458065310393643
5 cluster 1 y
6 0.574144383983257
7 cluster 2 x
8 1.2988892624518416
9 cluster 2 y
10 0.19265327475816957
11 cluster 3 x
12 2.3949863012188333
13 cluster 3 y
14 0.7153867356107672
15
```

```
2 /bin/zsh [running]
:set mouse=a
```

2e

(e) Any observations comparing the two different distance functions?

answer 2e

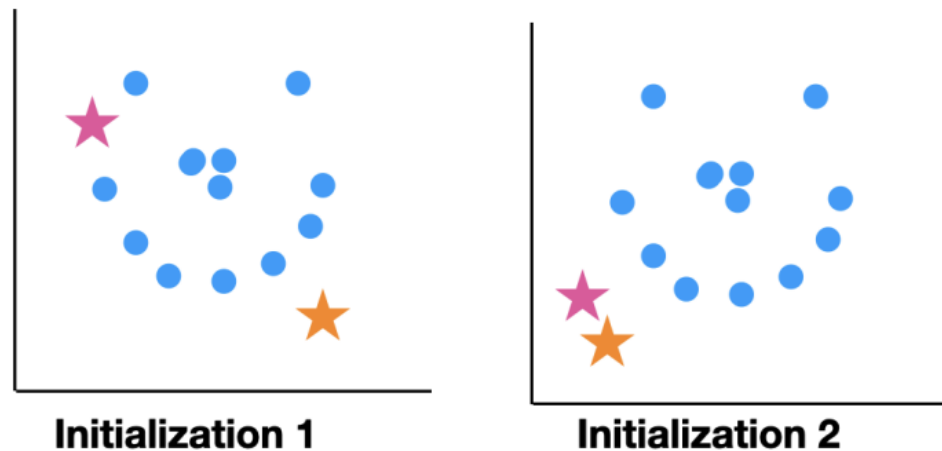
The clusters are farther apart in the M step of d than they are in the M step of b.

This is because the hidden matrix in part b weighs the points more equally whereas the hidden matrix in part d the hidden matrix has a more biased weighting.

problem 3

Decision boundaries in standard k -means

Consider the data in the figure below:

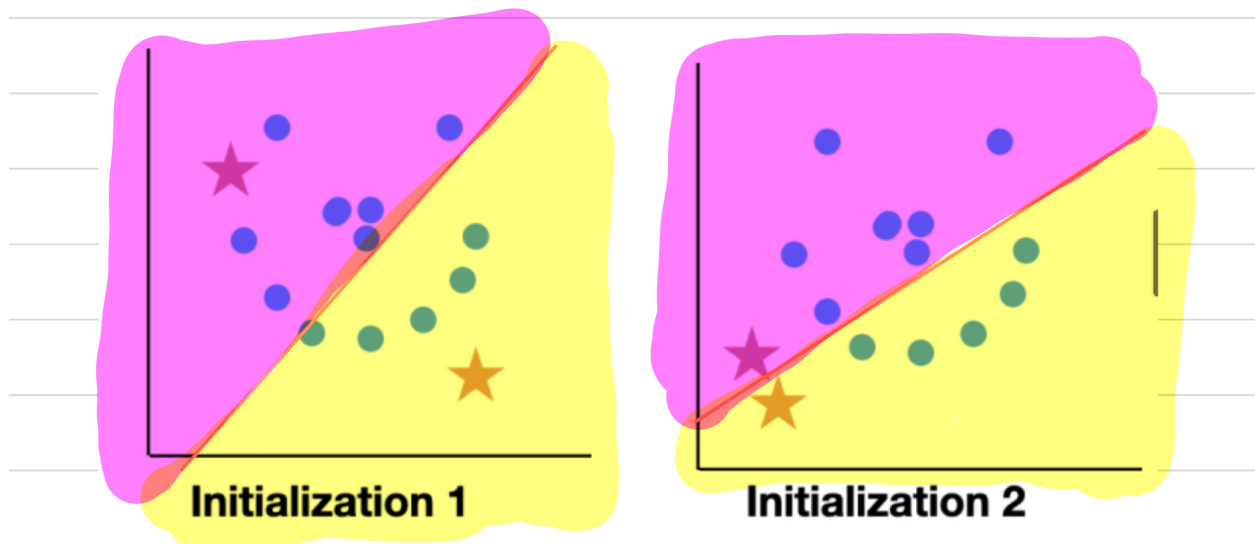


The blue dots are the data points, and the stars are the cluster centers.

3a

- (a) Given the current initializations, how are the points assigned using Initialization 1 and Initialization 2? In other words, how does the decision boundary look for each of these initializations in the current assignment step.

answer 3a



3b

- (b) Is there a qualitative difference between the assignments in the current step given the different initializations?

answer 3b

There is not a qualitative difference between initialization 1 and 2 for the current step.

3c

- (c) If you were to run the Lloyd algorithm with both initializations, how would it behave in the long term? Do they converge to the same answer? Does one take longer to converge than the other?

answer 3c

If we used the Lloyd algorithm with these two initializations then the next step would be to move the centers to the “center of gravity” of each of the two clusters. This would result in both initializations leading to the same next step.

Therefore both will lead to the same answer.

They will take the same amount of time because the next step for both is the same.

3d

(d) Do you see a picture in the data?

answer 3d

yes 😊

3e

(e) Draw some clustered data in two dimensions that is trivially easy to cluster by eye, but impossible to cluster correctly using k-means.

answer 3e

