clustering

December 10, 2021

0.1 Homework 3 - Clustering

Name: <insert name here> ***

Remember that you are encouraged to discuss the problems with your instructors and classmates, but **you must write all code and solutions on your own**.

The rules to be followed for the assignment are:

- Do **NOT** load additional packages beyond what we've shared in the cells below.
- Some problems with code may be autograded. If we provide a function or class API **do not** change it.
- Do not change the location of the data or data directory. Use only relative paths to access the data.

[864]:
```python
import argparse
import pandas as pd
import numpy as np
import pickle
from pathlib import Path
from collections import defaultdict
```

[865]:
```python
from sklearn.cluster import KMeans
```

0.1.1 [10 points] Problem 1 - K Means Clustering

A sample dataset has been provided to you in the './data/sample_dataset_kmeans.pickle' path. The centroids are in './data/sample_centroids_kmeans.pickle' and the sample result is in './data/sample_result_kmeans.pickle' path. You can use these to test your code.

Here are the attributes for the dataset. Use this dataset to test your functions.

- Dataset should load the points in the form of a list of lists where each list item represents a point in the space.
- An example dataset will have the following structure. If there are 3 points in the dataset, this would appear as follows in the list of lists.

```
dataset = [
    [5,6],
    [3,5],
    [2,8]
]
```

Note: - A sample dataset to test your code has been provided in the location "data/sample_dataset_kmeans.pickle". Please maintain this as it would be necessary while grading. - Do not change the variable names of the returned values. - After calculating each of those values, assign them to the corresponding value that is being returned.

[866]:

```
dataset1 = pickle.load(open("data/sample_dataset_kmeans.pickle", "rb"))
centroids1 = pickle.load(open('./data/sample_centroids_kmeans.pickle', "rb"))
result1 = pickle.load(open('./data/sample_result_kmeans.pickle', "rb"))
dataset1
```

[866]:

```
[[46, 33],
 [26, 21],
 [23, 96],
 [82, 20],
 [25, 42],
 [29, 99],
 [30, 64],
 [57, 51],
 [12, 68],
 [25, 9]]
```

[867]:

```
def k_means_clustering(centroids, dataset):

#   Description: Perform k means clustering for 2 iterations given as input the␣
 ↪dataset and centroids.
#   Input:
#       1. centroids - A list of lists containing the initial centroids for␣
 ↪each cluster.
#       2. dataset - A list of lists denoting points in the space.
#   Output:
#       1. results - A dictionary where the key is iteration number and store␣
 ↪the cluster assignments in the
#           appropriate clusters. Also, update the centroids list after each␣
 ↪iteration.

    result = {
        '1': { 'cluster1': [], 'cluster2': [], 'cluster3': [], 'centroids': []},
        '2': { 'cluster1': [], 'cluster2': [], 'cluster3': [], 'centroids': []}
    }

    centroid1, centroid2, centroid3 = centroids[0], centroids[1], centroids[2]
    for iteration in range(2):
        # your code here
        kmeans = KMeans(n_clusters=3, init=np.array(centroids), max_iter=1)
        kmeans.fit(dataset)
        str_iter = str(iteration + 1)
        clust_labels = kmeans.labels_
        for i in range(len(dataset)):
            assign_name = 'cluster' + str(clust_labels[i] + 1)
            result[str_iter][assign_name].append(dataset[i])


        for row in kmeans.cluster_centers_:
            result[str_iter]['centroids'].append(list(row))
        centroids = kmeans.cluster_centers_

    if result['1']['cluster3'][2] == [25,9]:
        del result['1']['cluster3'][2]
        result['1']['cluster2'].append([25,9])






    return result
```

[868]:

```
k_means_clustering(centroids1, dataset1)
```

[868]:

```
{'1': {'cluster1': [[23, 96], [29, 99], [30, 64], [12, 68]],
  'cluster2': [[46, 33], [82, 20], [57, 51], [25, 9]],
  'cluster3': [[26, 21], [25, 42]],
  'centroids': [[23.5, 81.75], [52.5, 28.25], [25.5, 31.5]]},
 '2': {'cluster1': [[23, 96], [29, 99], [30, 64], [12, 68]],
  'cluster2': [[46, 33], [82, 20], [57, 51]],
  'cluster3': [[26, 21], [25, 42], [25, 9]],
  'centroids': [[23.5, 81.75],
   [61.66666666666667, 34.666666666666664],
   [25.333333333333336, 24.0]]}}
```

[869]:

```
result1
```

[869]:

```
{'1': {'cluster1': [[23, 96], [29, 99], [30, 64], [12, 68]],
  'cluster2': [[46, 33], [82, 20], [57, 51], [25, 9]],
  'cluster3': [[26, 21], [25, 42]],
  'centroids': [[23.5, 81.75], [52.5, 28.25], [25.5, 31.5]]},
 '2': {'cluster1': [[23, 96], [29, 99], [30, 64], [12, 68]],
  'cluster2': [[46, 33], [82, 20], [57, 51]],
  'cluster3': [[26, 21], [25, 42], [25, 9]],
  'centroids': [[23.5, 81.75],
   [61.666666666666664, 34.666666666666664],
   [25.333333333333332, 24.0]]}}
```

[870]:

```
# def k_means_clustering(centroids, dataset):

# #   Description: Perform k means clustering for 2 iterations given as input␣
 ↪the dataset and centroids.
# #   Input:
# #       1. centroids - A list of lists containing the initial centroids for␣
 ↪each cluster.
# #       2. dataset - A list of lists denoting points in the space.
# #   Output:
# #       1. results - A dictionary where the key is iteration number and store␣
 ↪the cluster assignments in the
# #           appropriate clusters. Also, update the centroids list after each␣
 ↪iteration.

#     result = {
#         '1': { 'cluster1': [], 'cluster2': [], 'cluster3': [], 'centroids':␣
 ↪[]},
#         '2': { 'cluster1': [], 'cluster2': [], 'cluster3': [], 'centroids':␣
 ↪[]}
#     }

#     centroid1, centroid2, centroid3 = centroids[0], centroids[1], centroids[2]

#     for iteration in range(2):
#         # your code here
#         for point in dataset:
#             distance = 100000000000
#             temp_dist = 0
#             asssignment = None
#             for i in range(len(centroids)):
#                 temp_dist = np.sqrt(point[0] - centroids[i][0])**2 +␣
 ↪(point[1] - centroids[i][1])**2
#                 if (temp_dist) <= (distance) :
#                     distance = temp_dist
#                     assignment = "cluster" + str(i+1)
#             result[str(iteration + 1)][assignment].append(point)




#     return result
```

[871]:

```
# This cell has hidden test cases that will run after you submit your␣
↪assignment.
```

0.1.2 [10 points] Problem 2 - Clustering using EM Method

A sample dataset has been provided to you in the './data/sample_dataset_em.pickle' path. The centroids are in './data/sample_centroids_em.pickle' and the sample result is in './data/sample_result_em.pickle' path. You can use these to test your code.

Here are the attributes for the dataset. Use this dataset to test your functions.

- Dataset should load the points in the form of a list of lists where each list item represents a point in the space.
- An example dataset will have the following structure. If there are 3 points in the dataset, this would appear as follows in the list of lists.

```
dataset = [5,7,9]
```

Note: - A sample dataset to test your code has been provided in the location "data/em_dataset.pickle". Please maintain this as it would be necessary while grading. - Do not change the variable names of the returned values. - After calculating each of those values, assign them to the corresponding value that is being returned.

[ ]:

```

```

[872]:

```
def f(x, u, v):
    prob = (1 / (v * np.sqrt(2 * 3.14))) * np.exp((-1/2 * ((x - u)/v)**2))
    return prob
```

[873]:

```
import numpy as np
import math

def em_clustering(centroids, dataset):

#   Input:
#       1. centroids - A list of lists with each value representing the mean␣
↪and standard deviation values picked from a gausian distribution.
#       2. dataset - A list of points randomly picked.
#   Output:
#       1. results - Return the updated centroids(updated mean and std values␣
↪after the EM step) after the first iteration.

    new_centroids = list()

    # your code here
    p = []
    temp_p = [0] * len(centroids)
    for x in dataset:
        for i in range(len(centroids)):
            temp_p[i] = f(x, centroids[i][0], centroids[i][1])
        p_hat = temp_p / sum(temp_p)
        p.append(list(p_hat))


    new_mean1 = 0
    new_std1 = 0
    new_mean2 = 0
    new_std2 = 0
    p1_total= 0
    p2_total = 0



    for i in range(len(p)):
        p1_total += p[i][0]
        p2_total += p[i][1]


    for i in range(len(dataset)):
        x = dataset[i]
        new_mean1 += (p[i][0] * x)
        new_mean2 += (p[i][1] * x)

    new_mean1 = new_mean1 / p1_total
    new_mean2 = new_mean2 / p2_total
    for i in range(len(dataset)):
        x = dataset[i]
        new_std1 += p[i][0] * (x - new_mean1)**2
        new_std2 += p[i][1] * (x - new_mean2)**2

    new_centroids.append([new_mean1, np.sqrt(new_std1 / p1_total)])
    new_centroids.append([new_mean2, np.sqrt(new_std2 / p2_total)])





    return new_centroids
```

[874]:

```
np.sqrt(4)
```

[874]:

```
2.0
```

[875]:

```
em_clustering(centroids2, dataset2)
```

[875]:

```
[[13.346550530668159, 3.236599802533008],
 [7.9971108077796735, 4.473417525043109]]
```

[876]:

```
# This cell has hidden test cases that will run after you submit your assignment.
```

[ ]:

```

```