

CSE 4020 Machine Learning

Lab Assessment - 4

Sujay Kumar M 20BDS0294

Computer Science Engineering with Specialization with DataScience

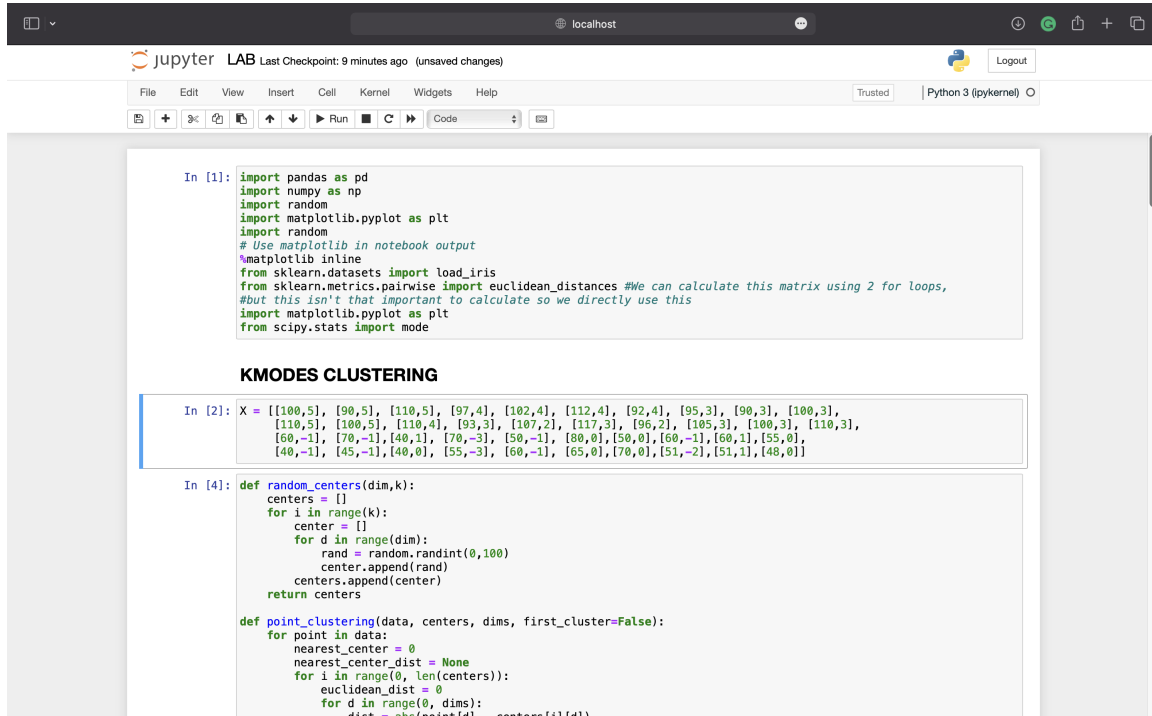
sujaykumarreddy.m2020@vitstudent.ac.in

<https://github.com/sujaykumarmag/CSE4020>

March 27, 2023

1 Imports

```
1 import pandas as pd
2 import numpy as np
3 import random
4 import matplotlib.pyplot as plt
5 import random
6 # Use matplotlib in notebook output
7 %matplotlib inline
8 from sklearn.datasets import load_iris
9 from sklearn.metrics.pairwise import euclidean_distances #We can calculate this matrix
    using 2 for loops,
10 #but this isn't that important to calculate so we directly use this
11 import matplotlib.pyplot as plt
12 from scipy.stats import mode
```



The screenshot shows a Jupyter Notebook interface with a dark theme. The top bar indicates 'localhost' and 'jupyter LAB'. Below the top bar is a menu bar with 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. The main area contains two code cells. The first cell, labeled 'In [1]:', contains the same import statements as shown in the previous block. The second cell, labeled 'In [2]:', contains a 2D array X of 100 data points. Below this is a third cell, labeled 'In [4]:', which defines two functions: 'random_centers' and 'point_clustering'. The 'random_centers' function takes dimensions and number of clusters as input and returns a list of random centers. The 'point_clustering' function takes data, centers, dimensions, and a flag for the first cluster as input and returns the cluster assignment for each point.

```
In [1]: import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt
import random
# Use matplotlib in notebook output
%matplotlib inline
from sklearn.datasets import load_iris
from sklearn.metrics.pairwise import euclidean_distances #We can calculate this matrix using 2 for loops,
#but this isn't that important to calculate so we directly use this
import matplotlib.pyplot as plt
from scipy.stats import mode
```

KMODES CLUSTERING

```
In [2]: X = [[100,5], [90,5], [110,5], [97,4], [102,4], [112,4], [92,4], [95,3], [90,3], [100,3],
[110,5], [100,5], [110,4], [93,3], [107,2], [117,3], [96,2], [105,3], [100,3], [110,3],
[60,-1], [70,-1], [40,1], [70,-3], [50,-1], [80,0], [50,0], [60,-1], [60,1], [55,0],
[40,-1], [45,-1], [40,0], [55,-3], [60,-1], [65,0], [70,0], [51,-2], [51,1], [48,0]]
```

```
In [4]: def random_centers(dim,k):
centers = []
for i in range(k):
center = []
for d in range(dim):
rand = random.randint(0,100)
center.append(rand)
centers.append(center)
return centers

def point_clustering(data, centers, dims, first_cluster=False):
for point in data:
nearest_center = 0
nearest_center_dist = None
for i in range(0, len(centers)):
euclidean_dist = 0
for d in range(0, dims):
dist = abs(point[d] - centers[i][d])
```

2 KModes Clustering

2.1 Input - a 2D array

```

1  def random_centers(dim,k):
2  centers = []
3  for i in range(k):
4  center = []
5  for d in range(dim):
6  rand = random.randint(0,100)
7  center.append(rand)
8  centers.append(center)
9  return centers
10
11 def point_clustering(data, centers, dims, first_cluster=False):
12 for point in data:
13 nearest_center = 0
14 nearest_center_dist = None
15 for i in range(0, len(centers)):
16 euclidean_dist = 0
17 for d in range(0, dims):
18 dist = abs(point[d] - centers[i][d])
19 euclidean_dist += dist
20 euclidean_dist = np.sqrt(euclidean_dist)
21 if nearest_center_dist == None:
22 nearest_center_dist = euclidean_dist
23 nearest_center = i
24 elif nearest_center_dist > euclidean_dist:
25 nearest_center_dist = euclidean_dist
26 nearest_center = i
27 if first_cluster:
28 point.append(nearest_center)
29 else:
30 point[-1] = nearest_center
31 return data
32
33 def mean_center(data, centers, dims):
34 print('centers:', centers, 'dims:', dims)
35 new_centers = []
36 for i in range(len(centers)):
37 new_center = []
38 n_of_points = 0
39 total_of_points = []
40 for point in data:
41 if point[-1] == i:
42 n_of_points += 1
43 for dim in range(0,dims):
44 if dim < len(total_of_points):
45 total_of_points[dim] += point[dim]
46 else:
47 total_of_points.append(point[dim])
48 if len(total_of_points) != 0:
49 for dim in range(0,dims):
50 print(total_of_points, dim)
51 new_center.append(total_of_points[dim]/n_of_points)
52 new_centers.append(new_center)
53 else:
54 new_centers.append(centers[i])
55
56
57 return new_centers
58
59 # Gets data and k, returns a list of center points.
60 def train_k_means_clustering(data, k=2, epochs=5):
61 dims = len(data[0])
62 print('data[0]:',data[0])
63 centers = random_centers(dims,k)
64
65 clustered_data = point_clustering(data, centers, dims, first_cluster=True)
66
67 for i in range(epochs):
68 centers = mean_center(clustered_data, centers, dims)
69 clustered_data = point_clustering(data, centers, dims, first_cluster=False)
70
71 return centers
72
73 def predict_k_means_clustering(point, centers):

```

```

74     dims = len(point)
75     center_dims = len(centers[0])
76
77     if dims != center_dims:
78         raise ValueError('Point given for prediction have', dims, 'dimensions but centers have'
79                             , center_dims, 'dimensions')
79
80     nearest_center = None
81     nearest_dist = None
82
83     for i in range(len(centers)):
84         euclidean_dist = 0
85         for dim in range(1, dims):
86             dist = point[dim] - centers[i][dim]
87             euclidean_dist += dist**2
88         euclidean_dist = np.sqrt(euclidean_dist)
89         if nearest_dist == None:
90             nearest_dist = euclidean_dist
91             nearest_center = i
92         elif nearest_dist > euclidean_dist:
93             nearest_dist = euclidean_dist
94             nearest_center = i
95         print('center:',i, 'dist:',euclidean_dist)
96
97     return nearest_center

```

The screenshot shows a Jupyter Lab window with a browser address bar at localhost. The interface includes a top bar with the Jupyter logo, 'LAB' title, and 'Last Checkpoint: 11 minutes ago (autosaved)'. Below this is a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and code execution. The main area contains a code cell with the following Python code:

```

nearest_center = 1
elif nearest_dist > euclidean_dist:
    nearest_dist = euclidean_dist
    nearest_center = i
print('center:',i, 'dist:',euclidean_dist)

return nearest_center

```

Below the code cell is the output for 'In [6]:', which shows the result of a K-means clustering operation:

```

centers = train_k_means_clustering(X, k=2, epochs=5)

data[0]: [100, 5]
centers: [[89, 98], [28, 41]] dims: 2
[2036, 73] 0
[2036, 73] 1
[1120, -12] 0
[1120, -12] 1
centers: [[101.8, 3.65], [56.0, -0.6]] dims: 2
[2036, 73] 0
[2036, 73] 1
[1120, -12] 0
[1120, -12] 1
centers: [[101.8, 3.65], [56.0, -0.6]] dims: 2
[2036, 73] 0
[2036, 73] 1
[1120, -12] 0
[1120, -12] 1
centers: [[101.8, 3.65], [56.0, -0.6]] dims: 2
[2036, 73] 0
[2036, 73] 1
[1120, -12] 0
[1120, -12] 1

```

3 KModes Clustering

```

1     clusters = np.zeros(data.shape[0], dtype=int)
2     clusters_prev = np.zeros(data.shape[0], dtype=int)
3
4
5     for i in range(10):
6         for j, object in enumerate(data):
7             distances = np.array([sum(object != mode) for mode in modes])
8             clusters[j] = np.argmin(distances)
9
10        for j in range(k):
11            modes[j] = mode(data[clusters == j]).mode[0]
12
13
14    if (clusters == clusters_prev).all():

```

```

15 break
16 clusters_prev = clusters
17
18 print("The cluster assignments for each data object: ", clusters)
19 print("Modes for each cluster: ", modes)
20

```

The screenshot shows a JupyterLab window with a code cell titled "KMODES CLUSTERING". The code defines a function for K-Modes clustering. The output shows the cluster assignments for each data object and the modes for each cluster. There are also some warnings about future and deprecation changes in SciPy and pandas.

```

In [7]: data = np.array([[ 'A', 'B', 'C'],[ 'B', 'C', 'A'],[ 'C', 'A', 'B'],[ 'A', 'C', 'B'],[ 'A', 'A', 'B']])
k = 2
modes = [[ 'A', 'B', 'C'],[ 'C', 'B', 'A']]

In [8]: clusters = np.zeros(data.shape[0], dtype=int)
clusters_prev = np.zeros(data.shape[0], dtype=int)

for i in range(10):
    for j, object in enumerate(data):
        distances = np.array([sum(object != mode) for mode in modes])
        clusters[j] = np.argmin(distances)

    for j in range(k):
        modes[j] = mode(data[clusters == j]).mode[0]

    if (clusters == clusters_prev).all():
        break
    clusters_prev = clusters

print("The cluster assignments for each data object: ", clusters)
print("Modes for each cluster: ", modes)

The cluster assignments for each data object: [0 1 0 0 0]
Modes for each cluster: [array([ 'A', 'A', 'B'], dtype='<U1'), array([ 'B', 'C', 'A'], dtype='<U1')]

/var/folders/l1/rp1rrpyx24d84x0k_6p7pwgww0000gn/T/ipykernel_18814/2679619983.py:11: FutureWarning: Unlike other reduction functions (e.g. 'skew', 'kurtosis'), the default behavior of 'mode' typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of 'keepdims' will become False, the 'axis' over which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set 'keepdims' to True or False to avoid this warning.
  modes[j] = mode(data[clusters == j]).mode[0]

/var/folders/l1/rp1rrpyx24d84x0k_6p7pwgww0000gn/T/ipykernel_18814/2679619983.py:11: DeprecationWarning: Support for non-numeric arrays has been deprecated as of SciPy 1.9.0 and will be removed in 1.11.0. 'pandas.DataFrame.mode' can be used instead, see https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.mode.html.
  modes[j] = mode(data[clusters == j]).mode[0]

```

4 Heirarchial Clustering

```

1 def OwnHeirarchical(data, cutoff, linkage):
2     #This is done using dynamic programming approach
3     # if 1, it is single linkage else 2 is complete linkage, 3 is average linkage
4     distance_matrix = euclidean_distances(data, data)
5     distance_matrix = np.tril(distance_matrix)
6     distance_matrix[distance_matrix == 0] = np.inf #Step 3 - Replace 0 by inf, it makes it
7     easy for us to extract minimum using min function
8     df = pd.DataFrame(data=np.ones(data.shape[0])*np.inf) #Initialized a dataframe which
9     will store which point is in which cluster
10    if cutoff > distance_matrix.shape[0]: #If user provides impractical cut-off, cluster
11    everthing into one cluster and not listen to user
12    cutoff = distance_matrix.shape[0]
13    if linkage == 1: #This 1 means formula of single linkage will be used, it is explained
14    ahead
15    d = {} #This dictionary keeps record of which data points or cluster are merging, hence
16    can be used to make a dendrogram
17    for i in range(0, cutoff):
18        ij_min = np.unravel_index(distance_matrix.argmin(), distance_matrix.shape) #from the
19        distance_matrix, get the minimum distance
20        #np.unravel_index gives us the position of minimum distance. e.g. (0,4) is where
21        minimum value is present in matrix.
22        #This is what we need as in Hierarchical clustering, we merge the two pairs with
23        minimum distance
24        if i == 0:
25            df.iloc[ij_min[0]] = 0
26            df.iloc[ij_min[1]] = 0
27        else:
28            try:
29                a = int(df.iloc[ij_min[0]])
30            except:
31                df.iloc[ij_min[0]] = i
32            a = i
33        try:

```

```

26 b = int(df.iloc[ij_min[1]])
27 except:
28 df.iloc[ij_min[1]] = i
29 b = i
30 df[(df[0]==a) | (df[0]==b)] = i
31 d[i] = ij_min
32
33 for j in range(0, ij_min[0]):
34
35 if np.isfinite(distance_matrix[ij_min[0]][j]) and np.isfinite(distance_matrix[ij_min
36 [1]][j]):
37 distance_matrix[ij_min[1]][j] = min(distance_matrix[ij_min[0]][j], distance_matrix[
38 ij_min[1]][j])
39 distance_matrix[ij_min[0]] = np.inf
40 return d, df[0]
41 elif linkage == 2:
42 d_complete = {}
43 for i in range(0,cutoff):
44 ij_min = np.unravel_index(distance_matrix.argmin(), distance_matrix.shape)
45 if i == 0:
46 df.iloc[ij_min[0]] = 0
47 df.iloc[ij_min[1]] = 0
48 else:
49 try:
50 a = int(df.iloc[ij_min[0]])
51 except:
52 df.iloc[ij_min[0]] = i
53 a = i
54 try:
55 b = int(df.iloc[ij_min[1]])
56 except:
57 df.iloc[ij_min[1]] = i
58 b = i
59 df[(df[0]==a) | (df[0]==b)] = i
60 d_complete[i] = ij_min
61 for j in range(0, ij_min[0]):
62 if np.isfinite(distance_matrix[ij_min[0]][j]) and np.isfinite(distance_matrix[ij_min
63 [1]][j]):
64 distance_matrix[ij_min[1]][j] = max(distance_matrix[ij_min[0]][j], distance_matrix[
65 ij_min[1]][j])
66 distance_matrix[ij_min[0]] = np.inf
67 return d_complete, df[0]
68 elif linkage == 3:
69 d_average = {}
70 for i in range(0,cutoff):
71 ij_min = np.unravel_index(distance_matrix.argmin(), distance_matrix.shape)
72 if i == 0:
73 df.iloc[ij_min[0]] = 0
74 df.iloc[ij_min[1]] = 0
75 else:
76 try:
77 a = int(df.iloc[ij_min[0]])
78 except:
79 df.iloc[ij_min[0]] = i
80 a = i
81 try:
82 b = int(df.iloc[ij_min[1]])
83 except:
84 df.iloc[ij_min[1]] = i
85 b = i
86 df[(df[0]==a) | (df[0]==b)] = i
87 d_average[i] = ij_min
88 for j in range(0, ij_min[0]):
89 if np.isfinite(distance_matrix[ij_min[0]][j]) and np.isfinite(distance_matrix[ij_min
90 [1]][j]):
91 distance_matrix[ij_min[1]][j] = (distance_matrix[ij_min[0]][j] + distance_matrix[ij_min
92 [1]][j])/2.0
93 distance_matrix[ij_min[0]] = np.inf
94 return d_average, df[0]

```

Jupyter LAB Last Checkpoint: 15 minutes ago (unsaved changes) Python 3 (pykernel)

```
In [18]: d, target = OwnHeirarchical(np.array(X), 14, 1)

In [19]: d
Out[19]: {0: (12, 2),
1: (19, 12),
2: (26, 24),
3: (32, 22),
4: (36, 21),
5: (13, 6),
6: (16, 7),
7: (37, 24),
8: (38, 26),
9: (8, 1),
10: (9, 0),
11: (11, 9),
12: (18, 0),
13: (23, 21)}
```

```
In [20]: target
Out[20]: 0 12.0
1 9.0
2 1.0
3 inf
4 inf
5 inf
6 5.0
7 6.0
8 9.0
9 12.0
10 inf
11 12.0
12 1.0
13 5.0
14 inf
15 inf
16 6.0
17 inf
18 12.0
19 1.0
20 inf
```

Jupyter LAB Last Checkpoint: 15 minutes ago (unsaved changes) Python 3 (pykernel)

```
In [20]: target
Out[20]: 0 12.0
1 9.0
2 1.0
3 inf
4 inf
5 inf
6 5.0
7 6.0
8 9.0
9 12.0
10 inf
11 12.0
12 1.0
13 5.0
14 inf
15 inf
16 6.0
17 inf
18 12.0
19 1.0
20 inf
21 13.0
22 3.0
23 13.0
24 8.0
25 inf
26 8.0
27 inf
28 inf
29 inf
30 inf
31 inf
32 3.0
33 inf
34 inf
35 inf
36 13.0
37 8.0
38 8.0
39 inf
Name: 0, dtype: float64
```