In [1]:
```python
#PDE_Graphs task: spectral clustering
```

In [2]:
```python
#import packages

import numpy as np
import matplotlib.pyplot as plt
import scipy
import sklearn
from sklearn.datasets import make_moons
from scipy.spatial import KDTree
from scipy.sparse import coo_matrix
from scipy.sparse import csr_matrix
from sklearn.datasets import fetch_openml

plt.close('all')
```

In [3]:
```python
#Bungert starting code
#%% create data
d = 2
N = 2 * 500

mean1 = np.array([-1.5, 0])
cov1 = 0.8 * np.array([[1, 0], [0, 1]])

mean2 = np.array([1.5, 0])
cov2 = np.array([[1, 0], [0, 1]])

x1 = np.random.multivariate_normal(mean1, cov1, int(N/2))
x2 = np.random.multivariate_normal(mean2, cov2, int(N/2))

#remove outliers
for i in range(int(N/2)):
    dist1 = np.linalg.norm(x1[i,:] - mean1)
    if dist1 > 3 * np.linalg.det(cov1):
        x1[i,:] = (x1[i,:] - mean1)/dist1 + mean1

    dist2 = np.linalg.norm(x2[i,:] - mean2)
    if dist2 > 3 * np.linalg.det(cov2):
        x2[i,:] = (x2[i,:] - mean2)/dist2 + mean2
```

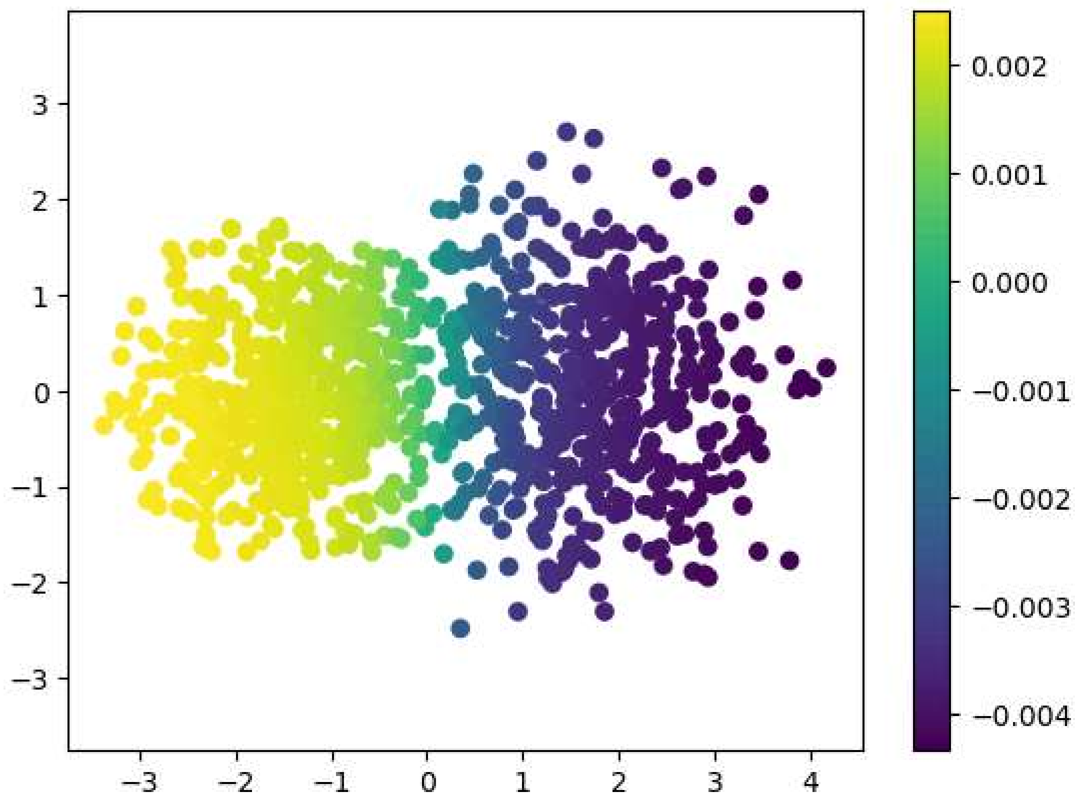
```
x = np.concatenate((x1, x2))
```

In [4]:

```
#%%create eps ball graph and its operators
eps = 1
eta = lambda t: 1 if t < 1 else 0

#compute weight matrix
W = np.zeros((N,N))
for i in range(N):
    for j in range(i+1,N):
        dist = np.linalg.norm(x[i,:]-x[j,:])
        W[i,j] = eta(dist/eps)

W += W.T

#compute degree matrix
D = np.zeros((N,N))
for i in range(N):
    D[i,i] = np.sum(W[i,:])

#compute Laplace matrix
L = W - D
```

In [5]:

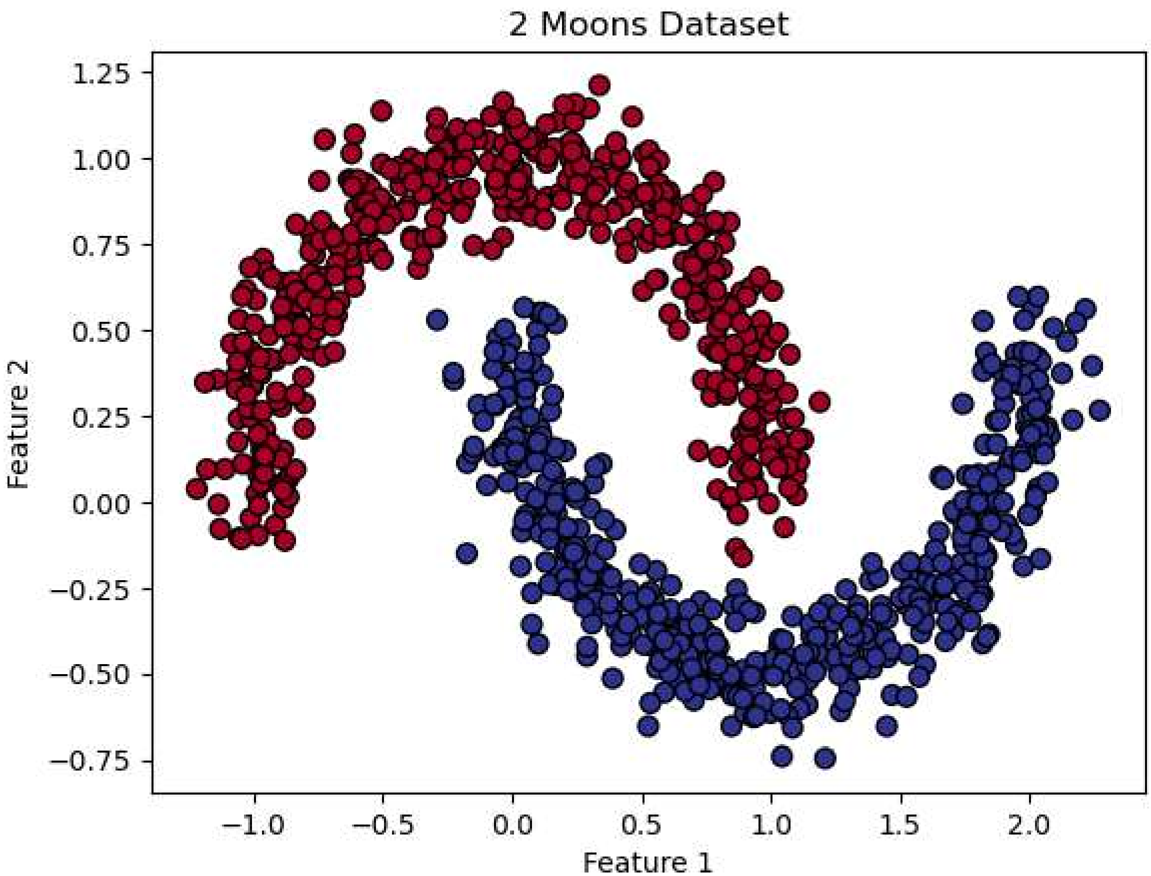
```
#%% compute eigenvectors
vals, vecs = scipy.linalg.eigh(-L, D, subset_by_index=[0, 1])

#%% plot
color = vecs[:,1]
plt.scatter(x[:,0], x[:,1], c=color)
plt.axis('equal')
plt.colorbar()
plt.show()
```



In [6]:
```
##############################################
#Now my code:
##############################################
```

In [7]:
```
#task 1: 2moons and knn
```

In [8]:
```
#data
N= 2* 500
X, y = make_moons(n_samples=N, noise=0.1, random_state=42)

# Plot the dataset
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.RdYlBu, edgecolors='black', s=50)  #s=marker size,  cmap=map scalar to
plt.title("2 Moons Dataset")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```



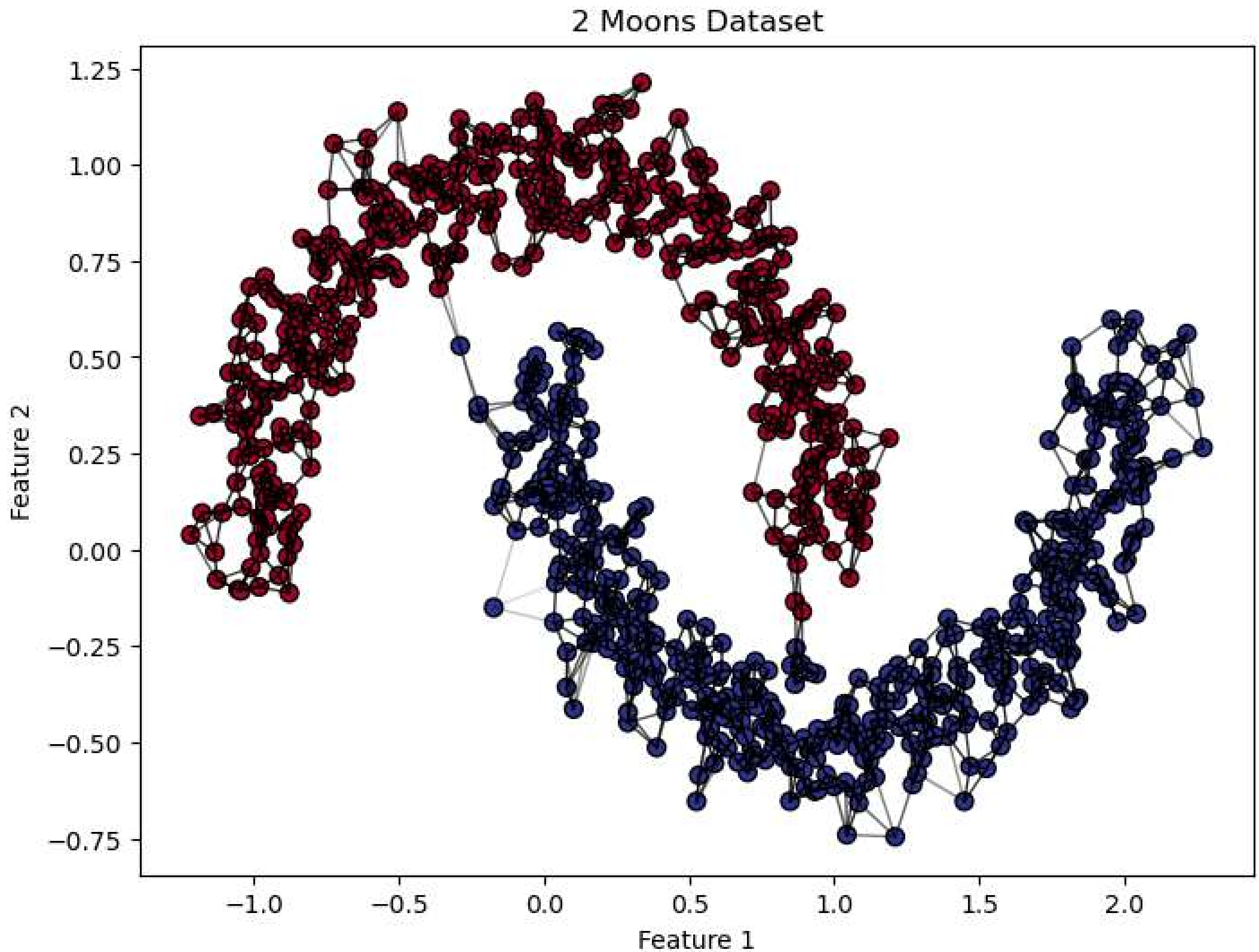
In [9]:
```
#build a knn graph

k = 4
```

```
# Build the KDTree
tree = KDTree(X)

# Query the KDTree for the k nearest neighbors for each point
distances, indices = tree.query(X, k=k+1)  # k+1 because the point itself will be the first neighbor

#some extra info
#type(distances)  numpy.ndarray
#distances.shape (1000, 4)  #distance to the neighbors

#indices.shape  (1000, 4)  #index in X of the neighbors
```

In [10]:

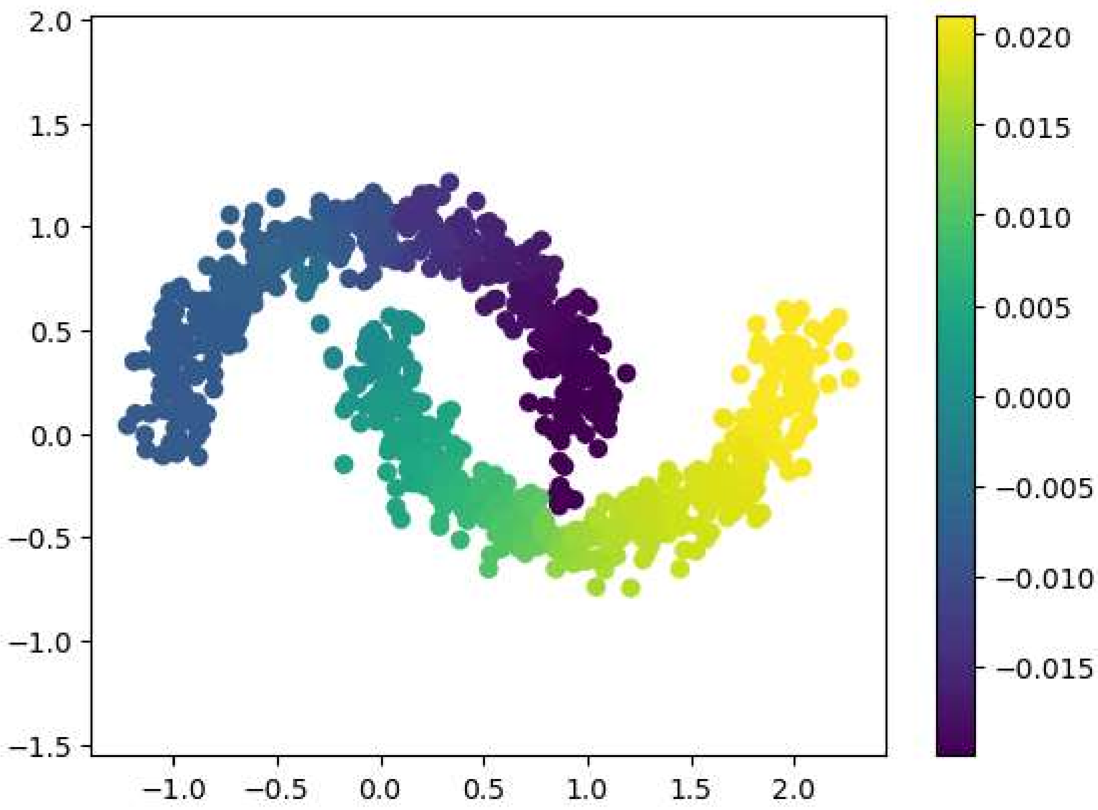
```
#%%create knn graph operators
#2 points connected if one of them is in knn

#compute weight matrix

eps = 1
eta = lambda t: np.exp(-t**2) if t < 1 else 0

W = np.zeros((N,N))
for i in range(N):
    for j in range(i+1,N):
        if j in indices[i] :
            index_j = np.argmax(indices[i]==j)
            W[i,j] = eta(distances[i, index_j]/eps)
        elif i in indices[j]:
            index_i = np.argmax(indices[j]==i)
            W[i,j] = eta(distances[j,index_i]/eps)

W += W.T

#compute degree matrix
D = np.zeros((N,N))
for i in range(N):
    D[i,i] = np.sum(W[i,:])

#compute Laplace matrix
L = W - D
```

```
In [11]: #plot

edges=[]

for i in range(N):
    for j in range(i+1,N):
        if W[i,j] > 0:
            edges.append([i,j,W[i,j]])

edges_con = np.array(edges)
#edges_con.shape (1917, 3)

#Normalize the weights to scale alpha (transparency) and linewidth (thickness)
weights = edges_con[:,2]
norm = plt.Normalize(vmin=min(weights), vmax=max(weights))
alphas = [norm(weight) for weight in weights]  # Normalize weights to alpha (0 to 1)
linewidths = [1  for weight in weights]  # Scale line width



# Create the plot
plt.figure(figsize=(8, 6))



for k in range(edges_con.shape[0]):
    ind_i=int(edges_con[k,0])
    ind_j=int(edges_con[k,1])
    plt.plot([X[ind_i,0],X[ind_j,0]], [X[ind_i,1],X[ind_j,1]], color='black', linestyle='-', alpha=alphas[k], linewid
    #alpha=transparency, linewidth="Liniuenbreite", zorder=1 means background and z = 2 forground)


plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.RdYlBu, edgecolors='black', s=50)  #s=marker size,  cmap=map scalar to

plt.title("2 Moons Dataset")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```



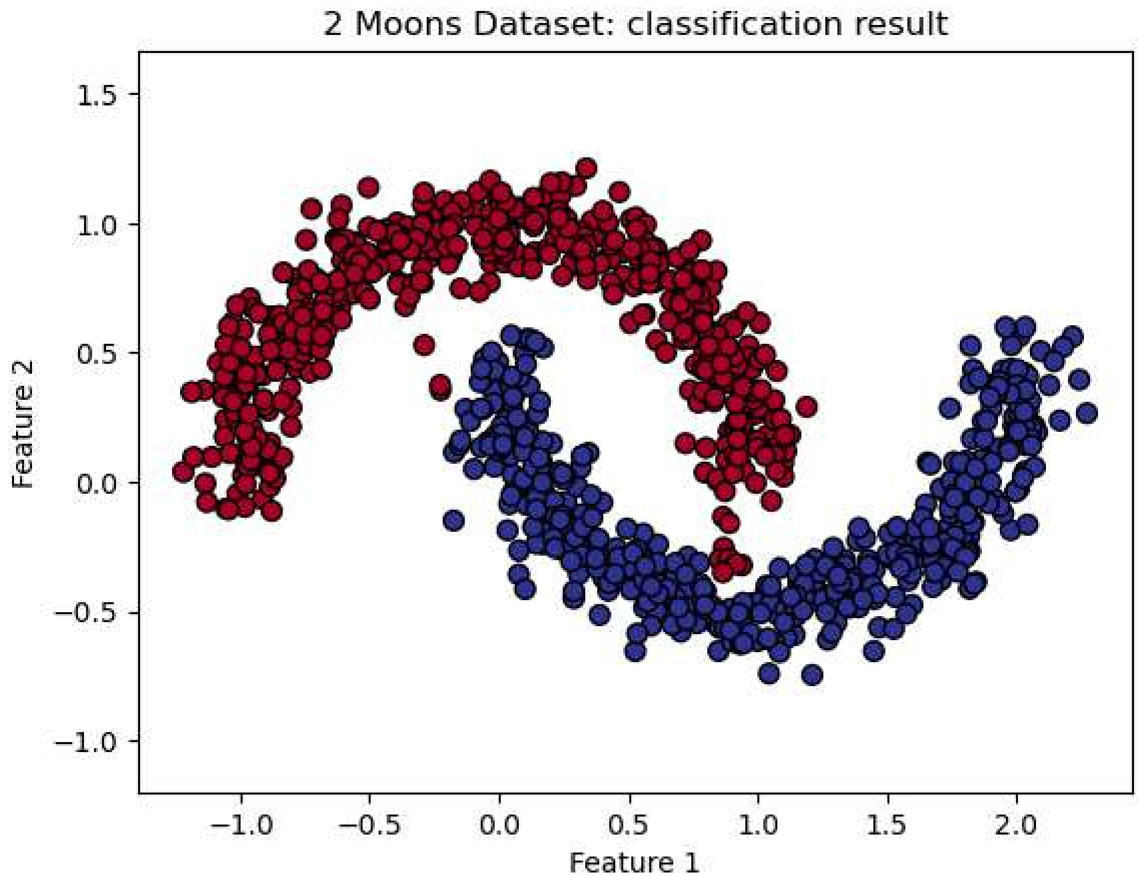
In [12]:
```
#%% compute eigenvectors
vals, vecs = scipy.linalg.eigh(-L, D, subset_by_index=[0, 1])

#%% plot
color = vecs[:,1]
plt.scatter(X[:,0], X[:,1], c=color)
plt.axis('equal')
```

```
plt.colorbar()
plt.show()
```



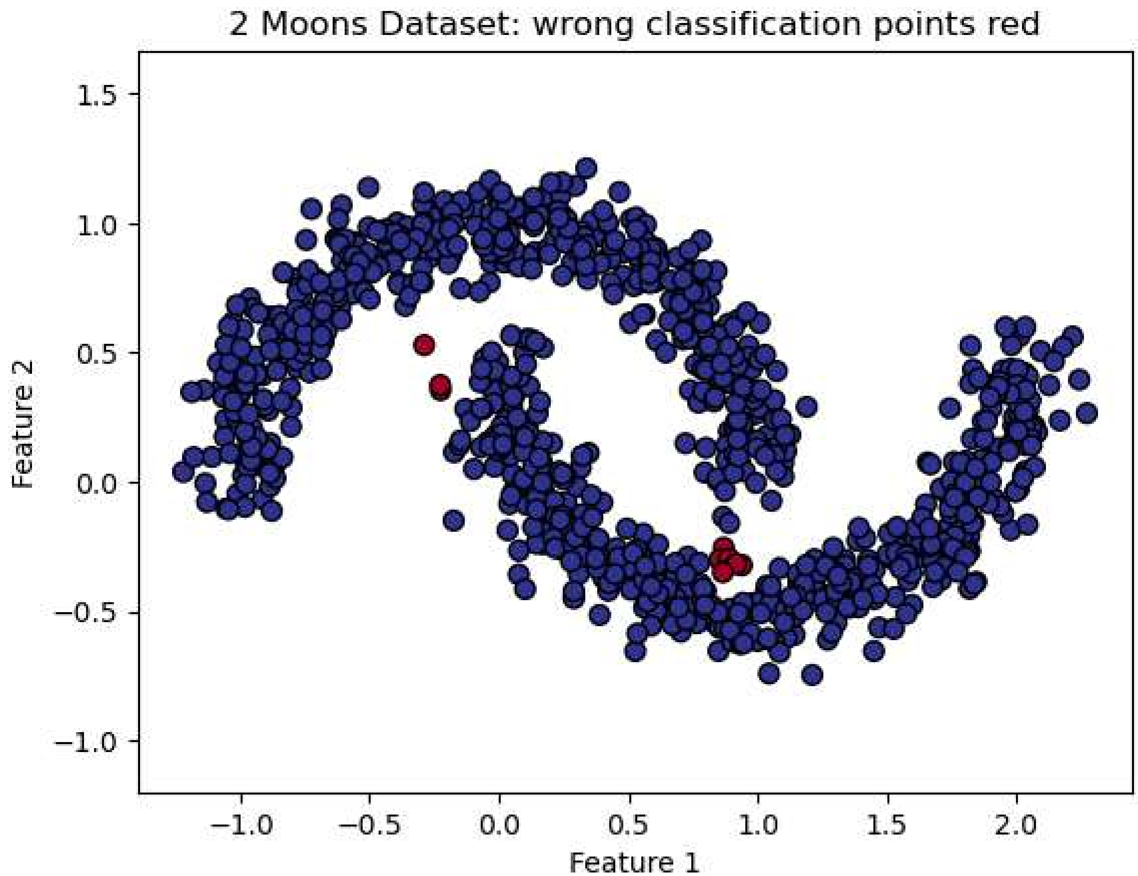
In [13]:

```
classif=np.zeros(X.shape[0])
boundary = 0
for j in range(X.shape[0]):
    if vecs[j,1]>=0:
        classif[j]=1
#%% plot
color = classif
plt.scatter(X[:,0], X[:,1], c=color, cmap=plt.cm.RdYlBu, edgecolors='black', s=50)
plt.axis('equal')
plt.title("2 Moons Dataset: classification result")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```



In [14]:
```
#wrong classified points
```

In [15]:
```
#%% plot
color = 1- np.abs(classif-y)
plt.scatter(X[:,0], X[:,1], c=color, cmap=plt.cm.RdYlBu, edgecolors='black', s=50)
plt.axis('equal')
plt.title("2 Moons Dataset: wrong classification points red")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```



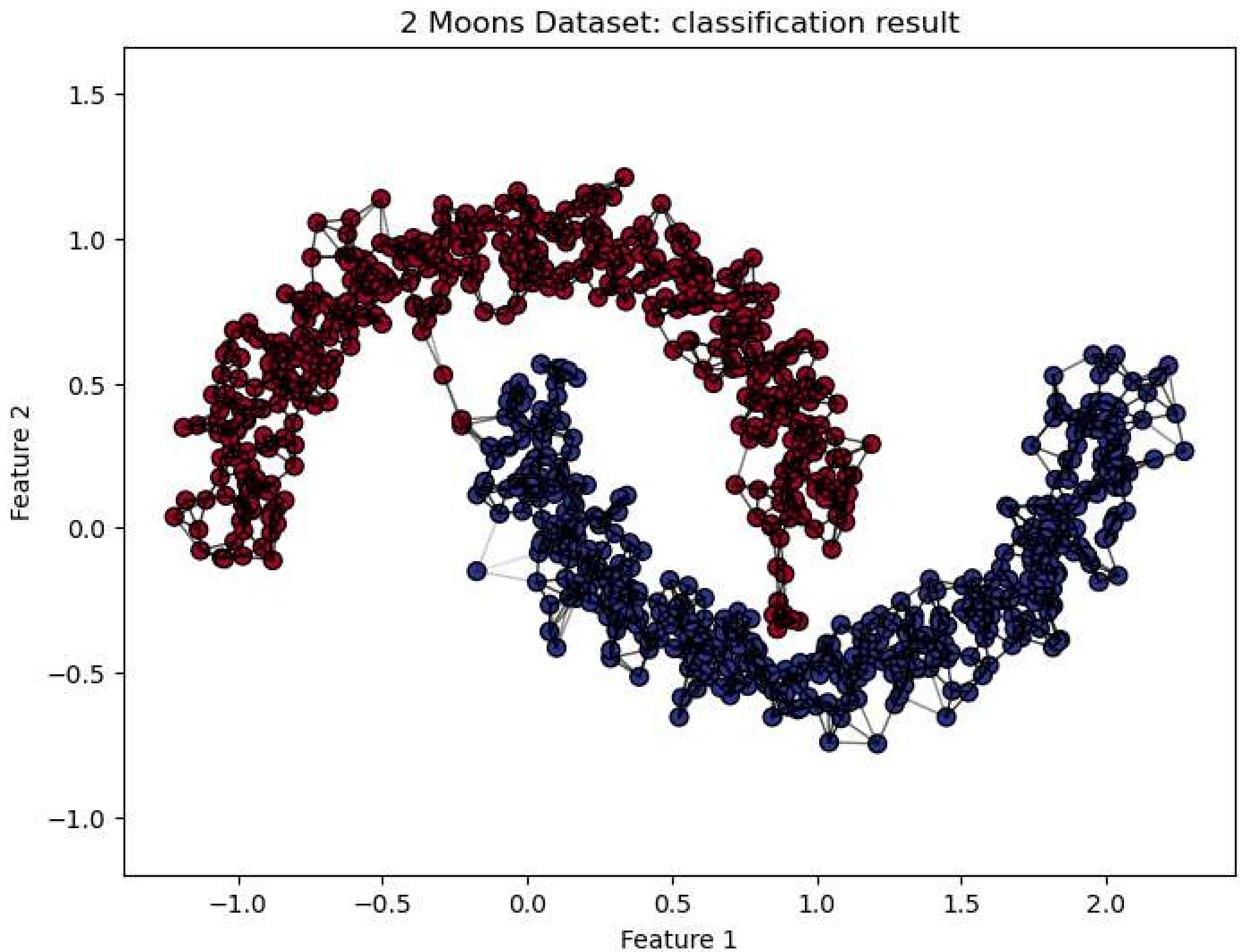
In [16]:
```
##############################################################################
#task 2: sparse matrices
##############################################################################
```

In [17]:
```
#data
N= 2* 500
X_dens, y_dens = make_moons(n_samples=N, noise=0.1, random_state=42)
X=csr_matrix(X_dens)
y=csr_matrix(y_dens)
#build a knn graph
k = 4
tree = KDTree(X_dens) #! did not work with dense matrix
distances_dens, indices_dens = tree.query(X_dens, k=k+1)  # k+1 because the point itself will be the first neighbor
#distances=csr_matrix(distances_dens) #! made problems by W calculation...
```

```
#indices=csr_matrix(indices_dens)
distances=distances_dens
indices=indices_dens

#compute weight matrix
eps = 1
eta = lambda t: np.exp(-t**2) if t < 1 else 0

W = csr_matrix((N, N))
for i in range(N):
    for j in range(i+1,N):
        if j in indices[i] :
            index_j = np.argmax(indices[i]==j)
            W[i,j] = eta(distances[i, index_j]/eps)
        elif i in indices[j]:
            index_i = np.argmax(indices[j]==i)
            W[i,j] = eta(distances[j,index_i]/eps)

W += W.T

#compute degree matrix
D = csr_matrix((N, N))
for i in range(N):
    D[i,i] = np.sum(W[i,:])

#compute Laplace matrix
L = W - D
#L


#%% compute eigenvectors
vals, vecs = scipy.sparse.linalg.eigsh(A=-L, M=D,k=2, which='SM') #k says how many eigenvectors are computed, all is
#'SM' means smallest eigenvalues


#######
#plotten
#######

#classif=csr_matrix((X_dens.shape[0],1))
classif=np.zeros(X_dens.shape[0])
boundary = 0
```

```
for j in range(X_dens.shape[0]):
    if vecs[j,1]>=0:
        classif[j]=1

edges=[]

for i in range(N):
    for j in range(i+1,N):
        if W[i,j] > 0:
            edges.append([i,j,W[i,j]])

edges_con = np.array(edges)
#edges_con = csr_matrix(edges)
#Normalize the weights to scale alpha (transparency) and linewidth (thickness)
weights = edges_con[:,2]
norm = plt.Normalize(vmin=min(weights), vmax=max(weights))
alphas = [norm(weight) for weight in weights]  # Normalize weights to alpha (0 to 1)
linewidths = [1  for weight in weights]  # Scale line width

vals



#%% plot

# Create the plot
plt.figure(figsize=(8, 6))

plt.scatter(X_dens[:,0], X_dens[:,1], c=classif, cmap=plt.cm.RdYlBu, edgecolors='black', s=50)  #s=marker size,  cmap

for k in range(edges_con.shape[0]):
    ind_i=int(edges_con[k,0])
    ind_j=int(edges_con[k,1])
    plt.plot([X_dens[ind_i,0],X_dens[ind_j,0]], [X_dens[ind_i,1],X_dens[ind_j,1]], color='black', linestyle='-', alph
    #alpha=transparency, linewidth="Liniuenbreite", zorder=1 means background and z = 2 forground)

plt.title("2 Moons Dataset: classification result")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.axis('equal')
plt.show()
```

```
C:\Users\Daniel\anaconda3\envs\ML_Graphs_Seminar\lib\site-packages\scipy\sparse\_index.py:108: SparseEfficiencyWarnin
g: Changing the sparsity structure of a csr_matrix is expensive. lil_matrix is more efficient.
  self._set_intXint(row, col, x.flat[0])
```



In [18]:

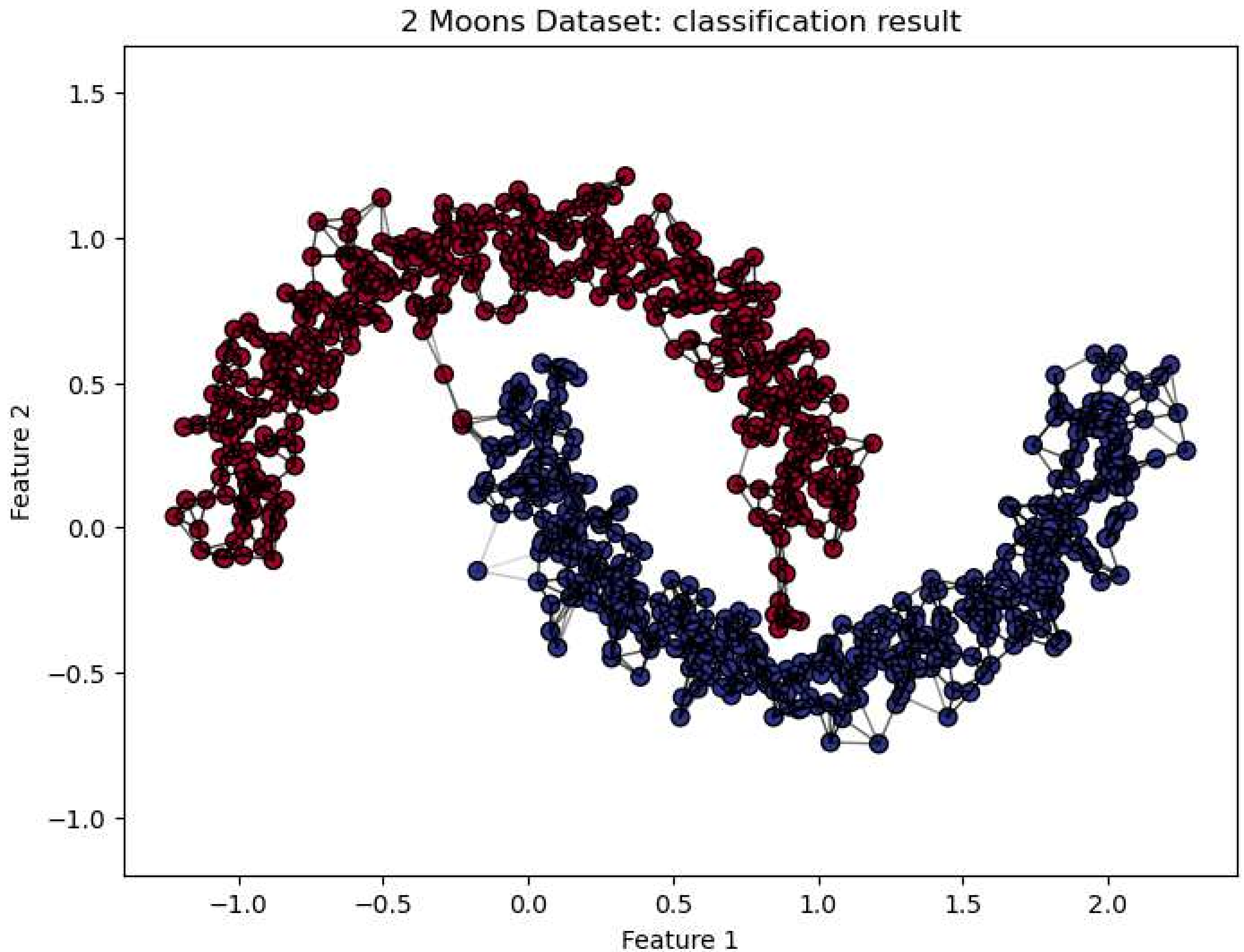
```
#Code without sparse
```

In [19]:

```
#data
N= 2* 500
X, y = make_moons(n_samples=N, noise=0.1, random_state=42)
```

```
#build a knn graph
k = 4
tree = KDTree(X)
distances, indices = tree.query(X, k=k+1)  # k+1 because the point itself will be the first neighbor

#compute weight matrix
eps = 1
eta = lambda t: np.exp(-t**2) if t < 1 else 0

W = np.zeros((N,N))
for i in range(N):
    for j in range(i+1,N):
        if j in indices[i] :
            index_j = np.argmax(indices[i]==j)
            W[i,j] = eta(distances[i, index_j]/eps)
        elif i in indices[j]:
            index_i = np.argmax(indices[j]==i)
            W[i,j] = eta(distances[j,index_i]/eps)

W += W.T

#compute degree matrix
D = np.zeros((N,N))
for i in range(N):
    D[i,i] = np.sum(W[i,:])

#compute Laplace matrix
L = W - D

#%% compute eigenvectors
vals, vecs = scipy.linalg.eigh(-L, D, subset_by_index=[0, 1])


#######
#plotten
#######

classif=np.zeros(X.shape[0])
boundary = 0
for j in range(X.shape[0]):
```

```
    if vecs[j,1]>=0:
        classif[j]=1

edges=[]

for i in range(N):
    for j in range(i+1,N):
        if W[i,j] > 0:
            edges.append([i,j,W[i,j]])

edges_con = np.array(edges)
#Normalize the weights to scale alpha (transparency) and linewidth (thickness)
weights = edges_con[:,2]
norm = plt.Normalize(vmin=min(weights), vmax=max(weights))
alphas = [norm(weight) for weight in weights]  # Normalize weights to alpha (0 to 1)
linewidths = [1  for weight in weights]  # Scale line width

#%% plot

# Create the plot
plt.figure(figsize=(8, 6))

plt.scatter(X[:,0], X[:,1], c=classif, cmap=plt.cm.RdYlBu, edgecolors='black', s=50)  #s=marker size,  cmap=map scalc

for k in range(edges_con.shape[0]):
    ind_i=int(edges_con[k,0])
    ind_j=int(edges_con[k,1])
    plt.plot([X[ind_i,0],X[ind_j,0]], [X[ind_i,1],X[ind_j,1]], color='black', linestyle='-', alpha=alphas[k], linewic
    #alpha=transparency, linewidth="Liniuenbreite", zorder=1 means background and z = 2 forground)

plt.title("2 Moons Dataset: classification result")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.axis('equal')
plt.show()
```



In [20]:
```
##################################################################################
#task 3: mnist 1 and 7 classification
##################################################################################
```

In [44]:
```
#data
# Load MNIST dataset (this may take a moment to download)
mnist = fetch_openml('mnist_784', version=1, as_frame=False,parser='liac-arff')
```

```
# `mnist.data` contains the image data (784 features, each representing one pixel of a 28x28 image)
# `mnist.target` contains the labels (the digits 0-9)
X_mn = mnist.data
y_mn = mnist.target.astype(int)  # Convert labels to integers

# Filter data for digits 1 and 7
filter_1_and_7 = np.isin(y_mn, [1, 7])

# Apply filter to get only 1s and 7s
X_filtered = X_mn[filter_1_and_7]
y_filtered = y_mn[filter_1_and_7]

#y_filtered
#y_filtered.size = 15170
```

In [22]:
```
X_mn.shape
```

Out[22]:
```
(70000, 784)
```

In [45]:
```
X=csr_matrix(X_filtered[0:1000,:])
y=csr_matrix(y_filtered[0:1000])
X_dens=X_filtered[0:1000,:]
y_dens=y_filtered[0:1000]

N=y_dens.size

#build a knn graph
k = 4
tree = KDTree(X_dens) #! did not work with dense matrix
distances_dens, indices_dens = tree.query(X_dens, k=k+1)  # k+1 because the point itself will be the first neighbor
#distances=csr_matrix(distances_dens) #! made problems by W calculation...
#indices=csr_matrix(indices_dens)
distances=distances_dens
indices=indices_dens

# #handle sparse data
# # Extract the non-zero indices (row, column) and values from the sparse matrix
# rows = X_dens.row  # Row indices of non-zero elements
# cols = X_dens.col  # Column indices of non-zero elements
# values = X_dens.data  # Non-zero values
```

```
# #  Prepare the data for KDTree
# points = np.vstack([rows, cols]).T  # Combine row and column indices into (x, y) points

# # Optionally: Use values as the third dimension (3D points)
# points_with_values = np.hstack([points, values.reshape(-1, 1)])

# #build a knn graph
# k = 4
# tree = KDTree(points_with_values) #! did not work with dense matrix
# distances_dens, indices_dens = tree.query(points_with_values, k=k+1)  # k+1 because the point itself will be the fi
# #distances=csr_matrix(distances_dens) #! made problems by W calculation...
# #indices=csr_matrix(indices_dens)
# distances=distances_dens
# indices=indices_dens
```

In [ ]:

In [46]:

```
#compute weight matrix
eps = 10000
eta = lambda t: np.exp(-t**2) if t > 0 else 0

W = csr_matrix((N, N))
for i in range(N):
    for j in range(i+1,N):
        if j in indices[i] :
            index_j = np.argmax(indices[i]==j)
            W[i,j] = eta(distances[i, index_j]/eps)
        elif i in indices[j]:
            index_i = np.argmax(indices[j]==i)
            W[i,j] = eta(distances[j,index_i]/eps)

W += W.T
```

In [47]:

```
W[1,:]
```

Out[47]:

```
<1x1000 sparse matrix of type '<class 'numpy.float64'>'
	with 6 stored elements in Compressed Sparse Row format>
```

In [48]:

```
#compute degree matrix
D = csr_matrix((N, N))
for i in range(N):
```

```
    D[i,i] = np.sum(W[i,:])

#compute Laplace matrix
L = W - D
#L


#%% compute eigenvectors
vals, vecs = scipy.sparse.linalg.eigsh(A=-L, M=D,k=2, which='SM') #k says how many eigenvectors are computed, all is
#'SM' means smallest eigenvalues
```

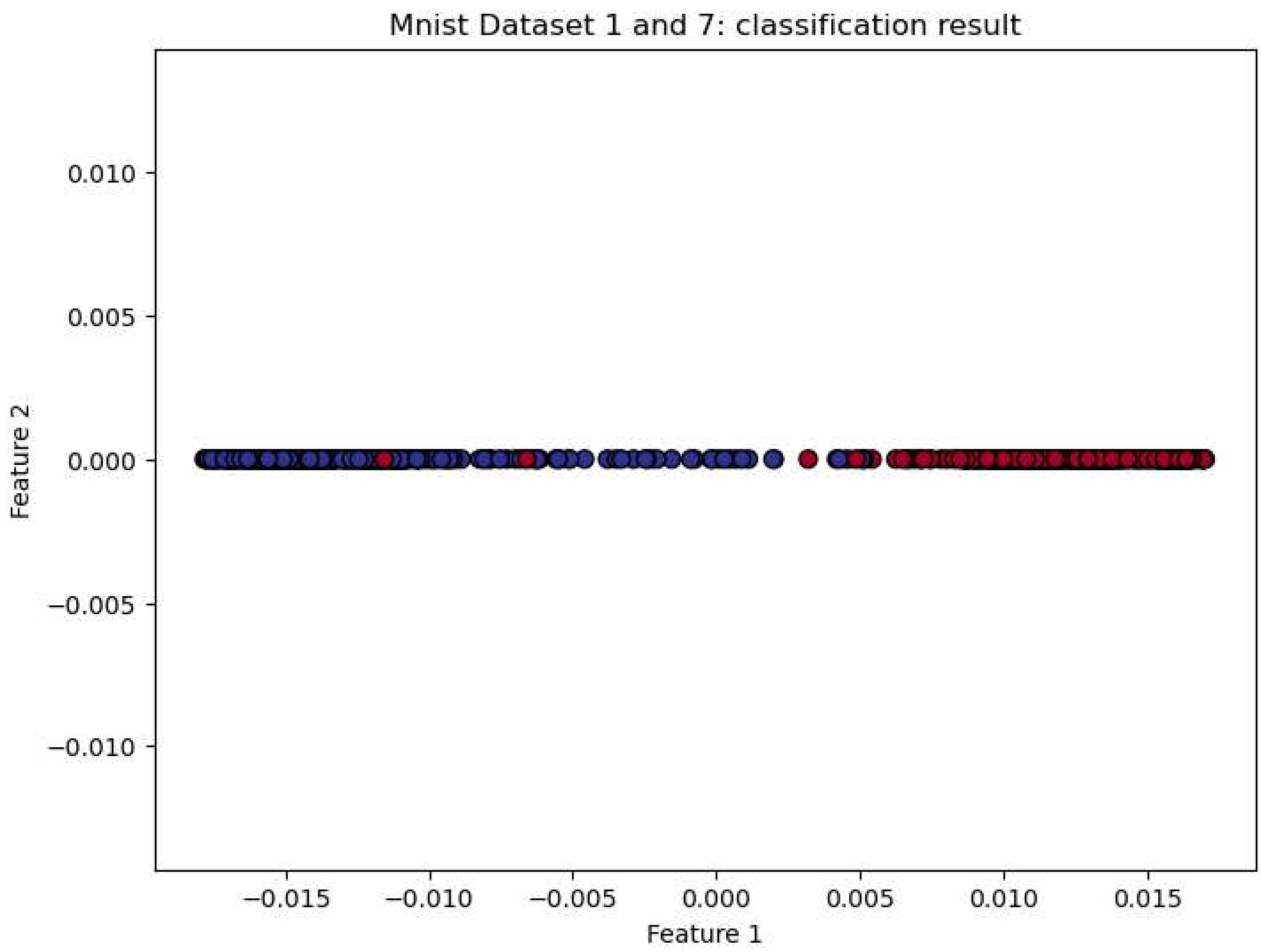
In [49]:
```
vals
```

Out[49]:
```
array([9.20958273e-17, 5.60303711e-03])
```

In [50]:
```
#######
#plotten
#######

#classif=csr_matrix((X_dens.shape[0],1))
classif=np.zeros(X_dens.shape[0])
boundary = 0.0025
for j in range(X_dens.shape[0]):
    if vecs[j,1]>=boundary:
        classif[j]=1

# edges=[]

# for i in range(N):
#     for j in range(i+1,N):
#         if W[i,j] > 0:
#             edges.append([i,j,W[i,j]])

# edges_con = np.array(edges)
# #edges_con = csr_matrix(edges)
# #Normalize the weights to scale alpha (transparency) and linewidth (thickness)
# weights = edges_con[:,2]
# norm = plt.Normalize(vmin=min(weights), vmax=max(weights))
# alphas = [norm(weight) for weight in weights]  # Normalize weights to alpha (0 to 1)
# linewidths = [1  for weight in weights]  # Scale line width
```

```
# vals
```

In [51]:
```
#%% plot

# Create the plot
plt.figure(figsize=(8, 6))

plt.scatter(vecs[:,1], np.zeros(X_dens[:,1].size), c=y_dens, cmap=plt.cm.RdYlBu, edgecolors='black', s=50)  #s=marke

# for k in range(edges_con.shape[0]):
#     ind_i=int(edges_con[k,0])
#     ind_j=int(edges_con[k,1])
#     plt.plot([X_dens[ind_i,0],X_dens[ind_j,0]], [X_dens[ind_i,1],X_dens[ind_j,1]], color='black', linestyle='-', al
#     #alpha=transparency, linewidth="Liniuenbreite", zorder=1 means background and z = 2 forground)

plt.title("Mnist Dataset 1 and 7: classification result")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.axis('equal')
plt.show()
```



In [53]:
```
#they are elements in R ^784...
#instead calculate how many correctly classified:
y_dens
classif

#1 means 1 and 7 means 0

y_cl=y_dens
y_cl[y_dens==7]=0
```

```
sum(abs(y_cl-classif))/y_dens.size  #part of wrong classified

# #or other way round
# #1 means 0 and 7 means 1

# y_cl=y_dens
# y_cl[y_dens==1]=0
# y_cl[y_dens==7]=1
# sum(abs(y_cl-classif))/y_dens.size  #part of wrong classified
```

Out[53]: 0.015

In [31]:
```
#######################################
#now all 10 numbers
#######################################
```

In [32]:
```
X_mn = mnist.data
y_mn = mnist.target.astype(int)  # Convert labels to integers
# Apply filter to get only 1s and 7s
X_filtered = X_mn[0:1000,:]
y_filtered = y_mn[0:1000]
```

In [33]:
```
X=csr_matrix(X_filtered[0:1000,:])
y=csr_matrix(y_filtered[0:1000])
X_dens=X_filtered[0:1000,:]
y_dens=y_filtered[0:1000]

N=y_dens.size

#build a knn graph
k = 4
tree = KDTree(X_dens) #! did not work with dense matrix
distances_dens, indices_dens = tree.query(X_dens, k=k+1)  # k+1 because the point itself will be the first neighbor
#distances=csr_matrix(distances_dens) #! made problems by W calculation...
#indices=csr_matrix(indices_dens)
distances=distances_dens
indices=indices_dens
```

In [34]:
```
#compute weight matrix
eps = 10000
eta = lambda t: np.exp(-t**2) if t > 0 else 0
```

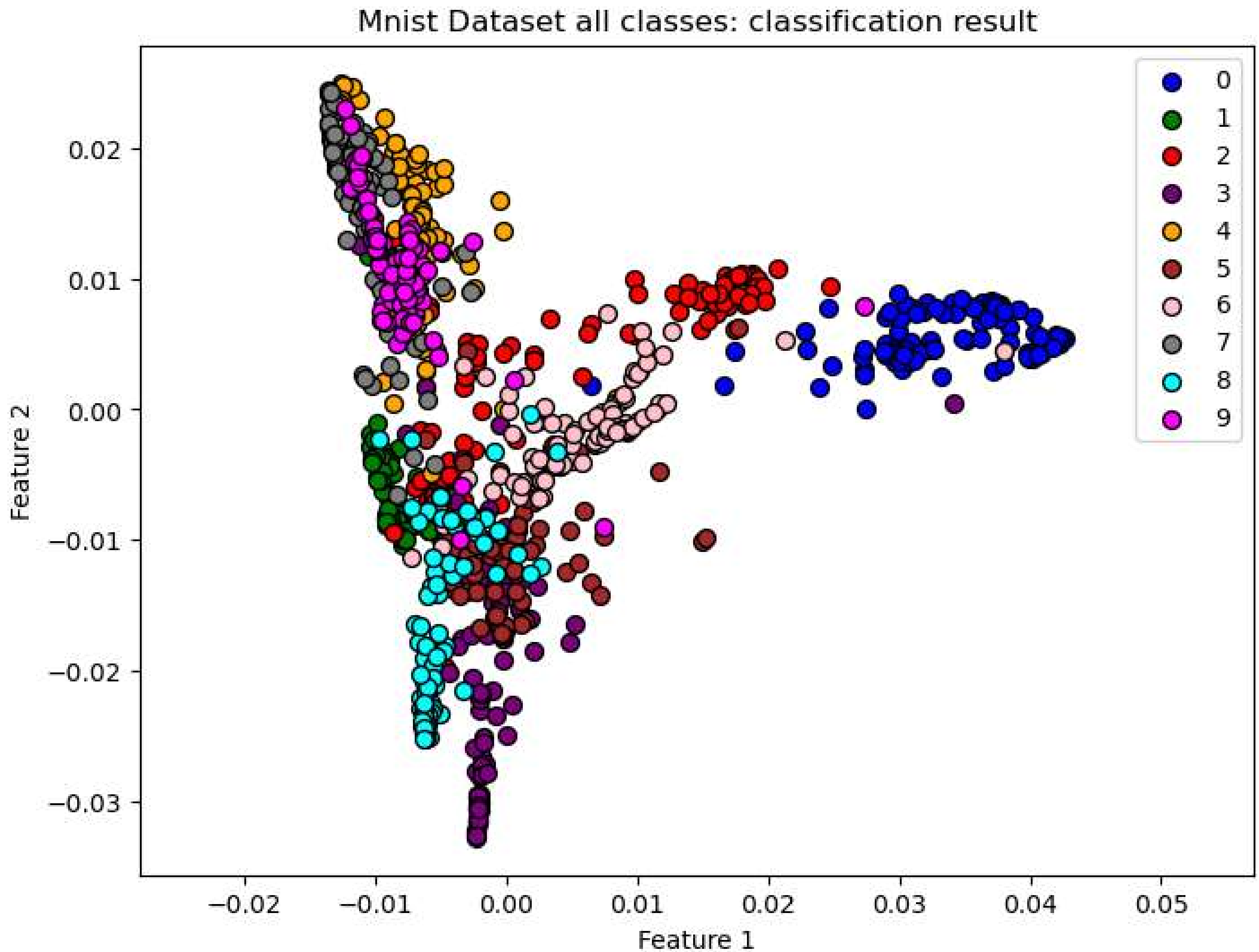
```
W = csr_matrix((N, N))
for i in range(N):
    for j in range(i+1,N):
        if j in indices[i] :
            index_j = np.argmax(indices[i]==j)
            W[i,j] = eta(distances[i, index_j]/eps)
        elif i in indices[j]:
            index_i = np.argmax(indices[j]==i)
            W[i,j] = eta(distances[j,index_i]/eps)

W += W.T
```

In [35]:
```
#compute degree matrix
D = csr_matrix((N, N))
for i in range(N):
    D[i,i] = np.sum(W[i,:])

#compute Laplace matrix
L = W - D
#L



#%% compute eigenvectors
vals, vecs = scipy.sparse.linalg.eigsh(A=-L, M=D,k=4, which='SM') #k says how many eigenvectors are computed, all is
#'SM' means smallest eigenvalues
vals
```

Out[35]:
```
array([-1.39572605e-17,  1.46830230e-02,  2.24133045e-02,  2.63795230e-02])
```

In [36]:
```
#%% plot

# Create the plot
plt.figure(figsize=(8, 6))
colors = ['blue', 'green', 'red', 'purple', 'orange',
          'brown', 'pink', 'gray', 'cyan', 'magenta']

for i in range(10):
    indexe=np.isin(y_filtered, [i])
    x_val=vecs[indexe,1]
    y_val=vecs[indexe,2]
```

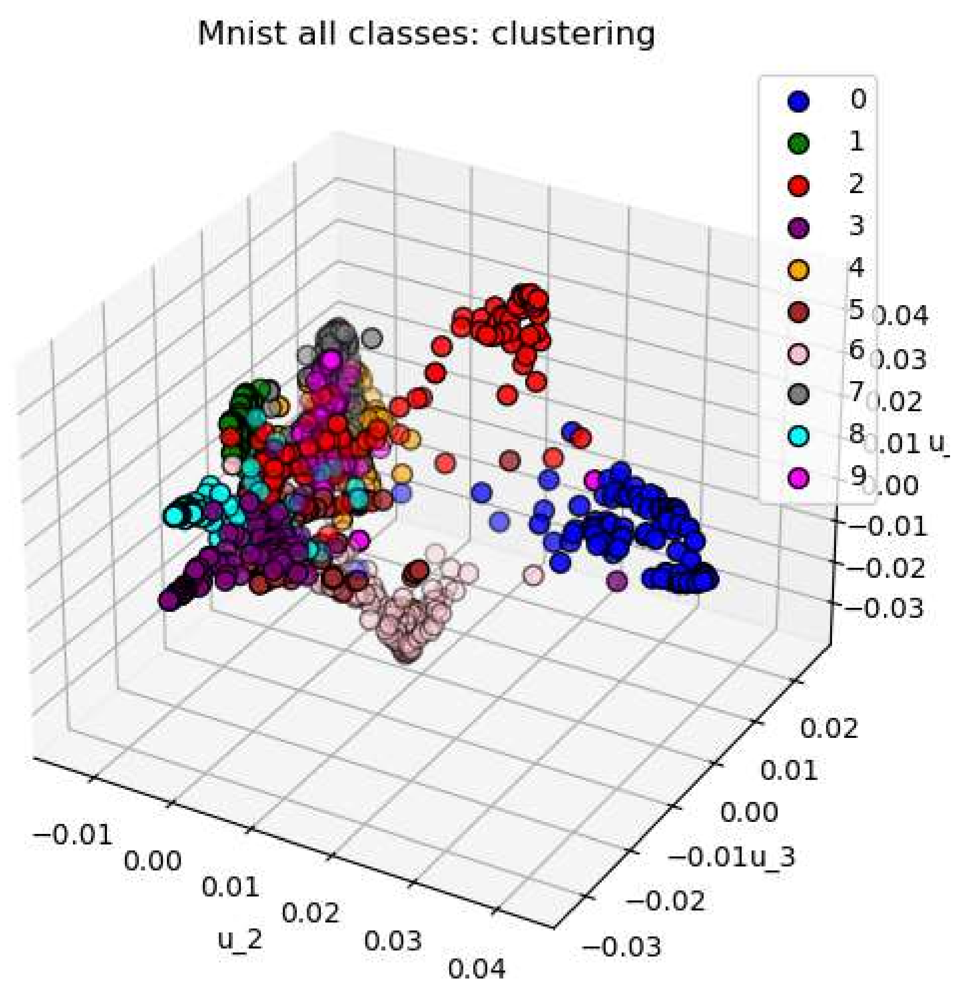
```
    plt.scatter(x_val, y_val, c=colors[i], edgecolors='black', s=50, label=i)  #s=marker size,  cmap=map scalar to co

# for k in range(edges_con.shape[0]):
#     ind_i=int(edges_con[k,0])
#     ind_j=int(edges_con[k,1])
#     plt.plot([X_dens[ind_i,0],X_dens[ind_j,0]], [X_dens[ind_i,1],X_dens[ind_j,1]], color='black', linestyle='-', al
#     #alpha=transparency, linewidth="Liniuenbreite", zorder=1 means background and z = 2 forground)

plt.title("Mnist Dataset all classes: classification result")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.axis('equal')
plt.legend()
plt.show()
```



In [37]:
```python
from mpl_toolkits.mplot3d import Axes3D  # 3D plotting tools
```

In [38]:
```python
# Create the plot
fig=plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')
colors = ['blue', 'green', 'red', 'purple', 'orange',
          'brown', 'pink', 'gray', 'cyan', 'magenta']
```

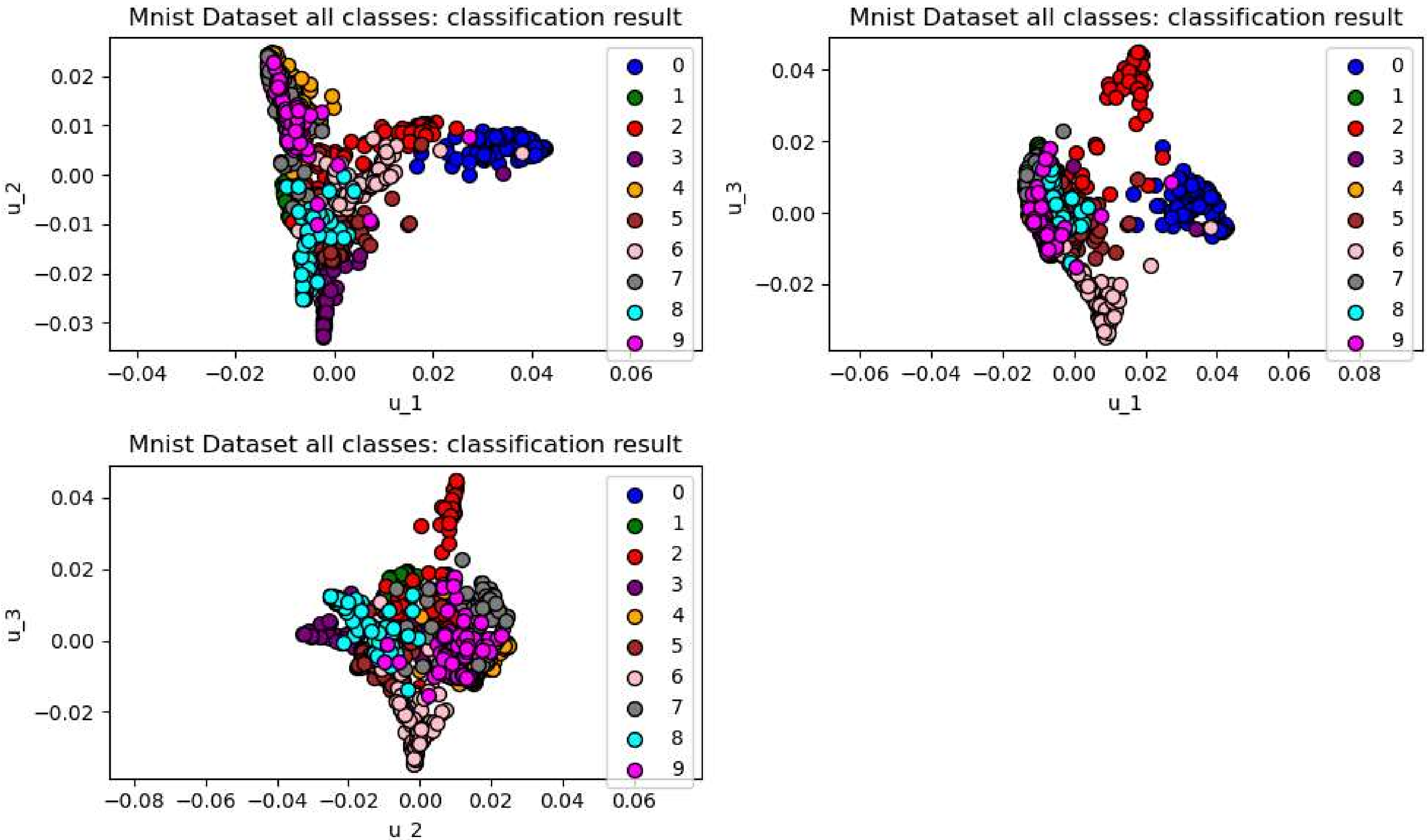
```
for i in range(10):
    indexe=np.isin(y_filtered, [i])
    x_val=vecs[indexe,1]
    y_val=vecs[indexe,2]
    z_val=vecs[indexe,3]
    ax.scatter(x_val, y_val, z_val, c=colors[i], edgecolors='black', s=50, label=i)  #s=marker size,  cmap=map scalar



# Set labels for axes
ax.set_xlabel('u_2')
ax.set_ylabel('u_3')
ax.set_zlabel('u_4')

# Set a title
ax.set_title('Mnist all classes: clustering')
ax.legend()

# Show the plot
plt.show()
```



In [39]:

```
#%% plot

# Create the plot
plt.figure(figsize=(10, 6))
colors = ['blue', 'green', 'red', 'purple', 'orange',
          'brown', 'pink', 'gray', 'cyan', 'magenta']

plt.subplot(2, 2, 1)  # (rows, cols, index)
for i in range(10):
    indexe=np.isin(y_filtered, [i])
    x_val=vecs[indexe,1]
```

```
    y_val=vecs[indexe,2]
    plt.scatter(x_val, y_val, c=colors[i], edgecolors='black', s=50, label=i)  #s=marker size,  cmap=map scalar to co


plt.title("Mnist Dataset all classes: classification result")
plt.xlabel("u_1")
plt.ylabel("u_2")
plt.axis('equal')
plt.legend()




plt.subplot(2, 2, 2)  # (rows, cols, index)
for i in range(10):
    indexe=np.isin(y_filtered, [i])
    x_val=vecs[indexe,1]
    y_val=vecs[indexe,3]
    plt.scatter(x_val, y_val, c=colors[i], edgecolors='black', s=50, label=i)  #s=marker size,  cmap=map scalar to co

plt.title("Mnist Dataset all classes: classification result")
plt.xlabel("u_1")
plt.ylabel("u_3")
plt.axis('equal')
plt.legend()




plt.subplot(2, 2, 3)  # (rows, cols, index)
for i in range(10):
    indexe=np.isin(y_filtered, [i])
    x_val=vecs[indexe,2]
    y_val=vecs[indexe,3]
    plt.scatter(x_val, y_val, c=colors[i], edgecolors='black', s=50, label=i)  #s=marker size,  cmap=map scalar to co

plt.title("Mnist Dataset all classes: classification result")
plt.xlabel("u_2")
plt.ylabel("u_3")
plt.axis('equal')
plt.legend()

plt.tight_layout()
plt.show()
```


Mnist Dataset all classes: classification result
Mnist Dataset all classes: classification result
Mnist Dataset all classes: classification result
u_1
u_2
u_1
u_3
u_2
u_3
0
1
2
3
4
5
6
7
8
9

In [ ]: