SMAI Assignment 4
Rehas Mehar Kaur Sachdeva
201401102

## Kernel K-Means Clustering Formulation:

Let $X = \{a_1, a_2, a_3, \ldots, a_n\}$ be the set of data points and $k$ be the number of clusters.

1. Randomly initialize $k$ cluster centers.
2. Compute the distance of each data point from each cluster center in the transformed space using:

$$D_{ic} = \|\Phi(a_i) - m_c\|^2 \; where \; m_c = \frac{\sum_{a_i \in \Pi_c} \Phi(a_i)}{|\Pi_c|}$$

$$D_{ic} = \Phi(a_i)\Phi(a_i) - \frac{2 \sum_{a_j \in \Pi_c} \Phi(a_i)\Phi(a_j)}{|\Pi_c|} + \frac{\sum_{a_j, a_l \in \Pi_c} \Phi(a_j)\Phi(a_l)}{|\Pi_c|^2}$$

   where
   $i$ denotes the $i^{th}$ data point,
   $c$ is the index of the $c^{th}$ cluster,
   $D_{ic}$ denotes the distance between $i^{th}$ data point and $c^{th}$ cluster,
   $\Phi(a_i)$ denotes the projection of $i^{th}$ data point in transformed space,
   $\Pi_c$ denotes the set of $c^{th}$ cluster,
   $m_c$ denotes the mean of the $c^{th}$ cluster.
   $\Phi(a_i)\Phi(a_j)$ Is nothing but the $ij^{th}$ entry of the Kernel matrix.
3. Assign data point to that cluster center whose distance is minimum.
4. Until data points are re-assigned repeat from step 2.

## Agglomerative Clustering:

```
import pandas as pd
import numpy as np

# Disable warnings from being printed
from warnings import filterwarnings
filterwarnings('ignore')

from sys import maxsize

from scipy.spatial.distance import pdist, squareform, euclidean
```

```
# For seeds dataset
# Read the data
data = pd.read_csv("seeds_dataset.txt", sep=r"\s*", header=None)

# Given labels (natural clusters in data)
cluster_numbers = data.loc[:, 7].copy()
cluster_numbers_predicted = data.loc[:, 7].copy()
```

```
cluster_numbers_predicted.iloc[:] = 0
natural_clusters = cluster_numbers.unique().shape[0]

# Get attributes
data = data.loc[:, 0:6]

# Total number of points
n = data.shape[0]

# Total clusters formed in the end
total_clusters = 2*n - 1

# A hash of current unmerged clusters
current_clusters = np.array([1]*n + [0]*(n-1))

# A grid of distances between each pair of clusters
dist_grid = np.zeros((total_clusters, total_clusters))
initial_grid = squareform(pdist(data, 'euclidean'))
for i in range(n):
  for j in range(n):
    dist_grid[i][j] = initial_grid[i][j]

# To track all points in a cluster
cluster_points = []
for i in range(n):
  cluster_points.append((i, ))
```

```
# For Data_User_Modeling_Dataset_Hamdi Tolga KAHRAMAN dataset
# Read the data
data = pd.read_csv("Data_User_Modeling_Dataset_Hamdi Tolga KAHRAMAN.csv",
sep=r"\s*", header=None)

# Given labels (natural clusters in data)
cluster_numbers = data.loc[:, 5].copy()
cluster_numbers[cluster_numbers == "very_low"] = 1
cluster_numbers[cluster_numbers == "High"] = 2
cluster_numbers[cluster_numbers == "Low"] = 3
cluster_numbers[cluster_numbers == "Middle"] = 4
cluster_numbers_predicted = cluster_numbers.copy()
cluster_numbers_predicted.iloc[:] = 0
natural_clusters = cluster_numbers.unique().shape[0]

# Get attributes
data = data.loc[:, 0:4]

# Total number of points
n = data.shape[0]

# Total clusters formed in the end
total_clusters = 2*n - 1

# A hash of current unmerged clusters
current_clusters = np.array([1]*n + [0]*(n-1))

# A grid of distances between each pair of clusters
dist_grid = np.zeros((total_clusters, total_clusters))
```

```
initial_grid = squareform(pdist(data, 'euclidean'))
for i in range(n):
  for j in range(n):
    dist_grid[i][j] = initial_grid[i][j]

# To track all points in a cluster
cluster_points = []
for i in range(n):
  cluster_points.append((i, ))
```

```
# For average distance criterion
# Total n-1 iterations
for i in range(n - 1):

  # Find minimum distance clusters
  mindist = maxsize
  minj = 0
  mink = 0
  for j in range(n + i):
    if current_clusters[j] == 0:
      continue
    for k in range(j + 1, n + i):
      if current_clusters[k] == 0:
        continue
      if mindist > dist_grid[j][k]:
        mindist = dist_grid[j][k]
        minj = j
        mink = k

  # Merge clusters
  cluster_points.append(cluster_points[minj] + cluster_points[mink])
  current_clusters[minj] = current_clusters[mink] = 0
  current_clusters[n + i] = 1

  # Update average distances from other clusters
  centre = data.iloc[list(cluster_points[n + i])].mean().values
  for j in range(n + i):
    temp_centre = data.iloc[list(cluster_points[j])].mean().values
    new_dist = np.linalg.norm(centre - temp_centre)
    dist_grid[n + i][j] = dist_grid[j][n + i] = new_dist

  # When current number of clusters equal natural clusters in data,
  # save the cluster for each point. This is used for calculating accuracy.
  if n - i + 1 == natural_clusters:
    for j in range(natural_clusters):
      for point in cluster_points[n + i - j]:
        cluster_numbers_predicted[point] = natural_clusters - j
```

```
# For minimum distance criterion
# Total n-1 iterations
for i in range(n - 1):

  # Find minimum distance clusters
  mindist = maxsize
```

```
  minj = 0
  mink = 0
  for j in range(n + i):
    if current_clusters[j] == 0:
      continue
    for k in range(j + 1, n + i):
      if current_clusters[k] == 0:
        continue
      if mindist > dist_grid[j][k]:
        mindist = dist_grid[j][k]
        minj = j
        mink = k

  # Merge clusters
  cluster_points.append(cluster_points[minj] + cluster_points[mink])
  current_clusters[minj] = current_clusters[mink] = 0
  current_clusters[n + i] = 1

  # Update minimum distances from other clusters
  for j in range(n + i):
    if current_clusters[j] == 0:
      continue
    distances = []
    for k in range(len(cluster_points[n + i])):
      for l in range(len(cluster_points[j])):
        cur_dist = np.linalg.norm(data.iloc[cluster_points[n + i][k]] -
data.iloc[cluster_points[j][l]])
        distances.append(cur_dist)
    dist_grid[n + i][j] = dist_grid[j][n + i] = np.min(distances)

  # When current number of clusters equal natural clusters in data,
  # save the cluster for each point. This is used for calculating accuracy.
  if n - i + 1 == natural_clusters:
    for j in range(natural_clusters):
      for point in cluster_points[n + i - j]:
        cluster_numbers_predicted[point] = natural_clusters - j
```

```
# Map the original cluster labels to new cluster labels
mappings = {}
mappings_unavailable = []
for i in range(1, natural_clusters + 1):
  maxcnt = -1
  maxj = 0
  for j in range(1, natural_clusters + 1):
    if j in mappings_unavailable:
      continue
    # Count the number of points matching if i maps to j
    cnt = 0
    for k in range(n):
      if cluster_numbers[k] == i and cluster_numbers_predicted[k] == j:
        cnt = cnt + 1
    if maxcnt < cnt:
      maxcnt = cnt
```

```
      maxj = j
  mappings[i] = maxj
  mappings_unavailable.append(maxj)

for mapping in mappings.keys():
  cluster_numbers[cluster_numbers == mapping] = mappings[mapping]
```

```
# Finally compute accuracy
cnt = 0.0
for i in range(n):
  if cluster_numbers[i] == cluster_numbers_predicted[i]:
    cnt = cnt + 1.0
print("Accuracy: ", cnt/n)
```

| **Dataset** | **Merging Criterion** | **Accuracy** |
|---|---|---|
| Seeds | Distance between means | 0.3952380952380952 |
| Seeds | Minimum distance | 0.37142857142857144 |
| Hamdi Tolga KAHRAMAN | Distance between means | 0.30077519379844961 |
| Hamdi Tolga KAHRAMAN | Minimum distance | 0.27751937984496124 |

Spectral Clustering:

```
import pandas as pd
import numpy as np

# Disable warnings from being printed
from warnings import filterwarnings
filterwarnings('ignore')

from scipy.linalg import eigh, eig
from sklearn.cluster import Kmeans
from scipy.spatial.distance import pdist, squareform
```

```
# For seeds dataset
# Read the data
data = pd.read_csv("seeds_dataset.txt", sep=r"\s*", header=None)

# Given labels (natural clusters in data)
cluster_numbers = data.loc[:, 7].copy()
cluster_numbers_predicted = data.loc[:, 7].copy()
cluster_numbers_predicted.iloc[:] = 0
natural_clusters = cluster_numbers.unique().shape[0]

# Get attributes
data = data.loc[:, 0:6]

# Total number of points
n = data.shape[0]
```

```
# For Data_User_Modeling_Dataset_Hamdi Tolga KAHRAMAN dataset
# Read the data
data = pd.read_csv("Data_User_Modeling_Dataset_Hamdi Tolga KAHRAMAN.csv",
```

```python
sep=r"\s*", header=None)

# Given labels (natural clusters in data)
cluster_numbers = data.loc[:, 5].copy()
cluster_numbers[cluster_numbers == "very_low"] = 1
cluster_numbers[cluster_numbers == "High"] = 2
cluster_numbers[cluster_numbers == "Low"] = 3
cluster_numbers[cluster_numbers == "Middle"] = 4
cluster_numbers_predicted = cluster_numbers.copy()
cluster_numbers_predicted.iloc[:] = 0
natural_clusters = cluster_numbers.unique().shape[0]

# Get attributes
data = data.loc[:, 0:4]

# Total number of points
n = data.shape[0]
```

```python
# Construct Affinity matrix

# For Data_User_Modeling_Dataset_Hamdi Tolga KAHRAMAN dataset
# sigma_sq = 1

# For seeds dataset
sigma_sq = 1e5

affinity_matrix = squareform(pdist(data, 'sqeuclidean'))
for i in range(n):
  for j in range(n):
    if i == j:
      continue
    affinity_matrix[i][j] = np.exp(-affinity_matrix[i][j] / (2 * sigma_sq))
```

```python
# Construct diagonal matrix
diagonal_matrix = np.zeros((n, n))
for i in range(n):
  diagonal_matrix[i][i] = np.sum(affinity_matrix[i])
```

```python
# Construct Laplacian matrix
L = diagonal_matrix - affinity_matrix
```

```python
# Find eigenvectors corresponding to k smallest eigenvalues,
# of L and stack them columnwise
eigvals, eigvecs = np.linalg.eigh(L)
X = np.column_stack((eigvecs[i] for i in range(natural_clusters)))
```

```python
clusters = KMeans(n_clusters=natural_clusters).fit(X)
cluster_numbers_predicted = clusters.labels_
```

```python
# Map the original cluster labels to new cluster labels
mappings = {}
mappings_unavailable = []
for i in range(1, natural_clusters + 1):
  maxcnt = -1
  maxj = 0
  for j in range(0, natural_clusters):
    if j in mappings_unavailable:
      continue
```

```
    # Count the number of points matching if i maps to j
    cnt = 0
    for k in range(n):
      if cluster_numbers[k] == i and cluster_numbers_predicted[k] == j:
        cnt = cnt + 1
    if maxcnt < cnt:
      maxcnt = cnt
      maxj = j
  mappings[i] = maxj
  mappings_unavailable.append(maxj)

for mapping in mappings.keys():
  cluster_numbers[cluster_numbers == mapping] = mappings[mapping]
```

```
# Finally compute accuracy
cnt = 0.0
for i in range(n):
  if cluster_numbers[i] == cluster_numbers_predicted[i]:
    cnt = cnt + 1.0
print("Accuracy: ", cnt/n)
```

| **Dataset** | **Accuracy** |
|---|---|
| Seeds | 0.34285714285714286 |
| Hamdi Tolga KAHRAMAN | 0.25689922480620156 |

Kernel K-Means Clustering:

```
import pandas as pd
import numpy as np
import random

# Disable warnings from being printed
from warnings import filterwarnings
filterwarnings('ignore')

from scipy.spatial.distance import pdist, squareform
from sys import maxsize
```

```
# For seeds dataset
# Read the data
data = pd.read_csv("seeds_dataset.txt", sep=r"\s*", header=None)

# Given labels (natural clusters in data)
cluster_numbers = data.loc[:, 7].copy()
cluster_numbers_predicted = data.loc[:, 7].copy()
cluster_numbers_predicted.iloc[:] = 0
natural_clusters = cluster_numbers.unique().shape[0]

# Get attributes
data = data.loc[:, 0:6]
```

```python
# Total number of points
n = data.shape[0]
```

```python
# For Data_User_Modeling_Dataset_Hamdi Tolga KAHRAMAN dataset
# Read the data
data = pd.read_csv("Data_User_Modeling_Dataset_Hamdi Tolga KAHRAMAN.csv",
sep=r"\s*", header=None)

# Given labels (natural clusters in data)
cluster_numbers = data.loc[:, 5].copy()
cluster_numbers[cluster_numbers == "very_low"] = 1
cluster_numbers[cluster_numbers == "High"] = 2
cluster_numbers[cluster_numbers == "Low"] = 3
cluster_numbers[cluster_numbers == "Middle"] = 4
cluster_numbers_predicted = cluster_numbers.copy()
cluster_numbers_predicted.iloc[:] = 0
natural_clusters = cluster_numbers.unique().shape[0]

# Get attributes
data = data.loc[:, 0:4]

# Total number of points
n = data.shape[0]
```

```python
# Compute Kernel Matrix
gamma = 1e-3
c = random.randint(1, 10)
d = 2

sq_dists = pdist(data, 'sqeuclidean')

# Converting the pairwise distances into a symmetric NxN matrix.
mat_sq_dists = squareform(sq_dists)

# Computing the NxN RBF kernel matrix.
# K = np.exp(-gamma * mat_sq_dists)

# Compute Polynomial kernel
K = (mat_sq_dists + c)**d
```

```python
# Randomly initialise k centroids as first k data points
centroid_matrix = data.iloc[0:natural_clusters]
cluster_points = []
for i in range(natural_clusters):
  cluster_points.append((i, ))
```

```python
# While data points are re-assigned clusters, loop
reassigned_flag = True
iter = 0
while reassigned_flag:
  iter = iter + 1
  if iter == 1e2:
    break
  reassigned_flag = False
  cluster_points_new = [()]*natural_clusters
  # For each data point compute nearest cluster
  for i in range(n):
```

```
    minj = 0
    mindist = maxsize
    for j in range(natural_clusters):
      cluster_cardinality = len(cluster_points[j])
      dist = K[i][i]
      sum = 0

      for point in cluster_points[j]:
        sum = sum + K[i][point]

      if cluster_cardinality != 0:
        dist = dist - (2*sum) / cluster_cardinality

      sum = 0
      for p1 in cluster_points[j]:
        for p2 in cluster_points[j]:
          sum = sum + K[p1][p2]

      if cluster_cardinality != 0:
        dist = dist - (sum / (cluster_cardinality**2))

      if mindist > dist:
        mindist = dist
        minj = j

    if cluster_numbers_predicted[i] != minj:
      reassigned_flag = True
      cluster_numbers_predicted[i] = minj

    cluster_points_new[minj] = cluster_points_new[minj] + (i, )

  cluster_points = cluster_points_new.copy()
```

```
# Map the original cluster labels to new cluster labels
mappings = {}
mappings_unavailable = []
for i in range(1, natural_clusters + 1):
  maxcnt = -1
  maxj = 0
  for j in range(0, natural_clusters):
    if j in mappings_unavailable:
      continue
    # Count the number of points matching if i maps to j
    cnt = 0
    for k in range(n):
      if cluster_numbers[k] == i and cluster_numbers_predicted[k] == j:
        cnt = cnt + 1
    if maxcnt < cnt:
      maxcnt = cnt
      maxj = j
  mappings[i] = maxj
  mappings_unavailable.append(maxj)
```

```python
for mapping in mappings.keys():
  cluster_numbers[cluster_numbers == mapping] = mappings[mapping]
```

```python
# Finally compute accuracy
cnt = 0.0
for i in range(n):
  if cluster_numbers[i] == cluster_numbers_predicted[i]:
    cnt = cnt + 1.0
print("Accuracy: ", cnt/n)
```

| **Dataset** | **Kernel** | **Accuracy** |
|---|---|---|
| Seeds | Polynomial | 0.5952380952380952 |
| Seeds | RBF | 0.8952380952380953 |
| Hamdi Tolga KAHRAMAN | Polynomial | 0.53069767441860467 |
| Hamdi Tolga KAHRAMAN | RBF | 0.70573643410852715 |