In [2]:
```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

```
Mounted at /content/drive
```

In [3]:
```
import pandas as pd

data = {'1': [], '2': [], '3': []}

with open('/content/drive/MyDrive/twoSpirals.txt', 'r') as f:
    for line in f:
        row = line.strip().split('\t')
        data['1'].append(float(row[0]))
        data['2'].append(float(row[1]))
        data['3'].append(float(row[2]))

df_spiral = pd.DataFrame(data)
```

In [4]:
```
df_spiral.head()
```

Out[4]:

	1	2	3
0	10.5192	-0.7170	-1.0
1	0.9987	-9.9681	-1.0
2	3.5763	8.3756	-1.0
3	1.9236	-10.6448	-1.0
4	8.1583	-5.9066	-1.0

In [5]:
```
data = {'1': [], '2': [], '3': []}

with open('/content/drive/MyDrive/threecircles.txt', 'r') as f:
    for line in f:
        row = line.strip().split(',')
        data['1'].append(float(row[0]))
        data['2'].append(float(row[1]))
        data['3'].append(float(row[2]))

df_circle = pd.DataFrame(data)
```

In [6]:
```
df_circle.head()
```

Out[6]:

	1	2	3
0	-0.208626	-0.264189	-1.0
1	0.499955	-0.073624	-1.0
2	-0.241661	-0.221071	-1.0
3	-0.356841	0.204201	-1.0
4	0.529480	0.170605	-1.0

In [8]:
```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error


# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df_spiral.iloc[:, :-1], df

# Fit a linear regression model on the training set
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the testing set and calculate RMSE
y_pred = model.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print("RMSE:", rmse)
```

```
RMSE: 0.9127353943716923
```

In [9]:
```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error


# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df_circle.iloc[:, :-1], df

# Fit a linear regression model on the training set
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the testing set and calculate RMSE
y_pred = model.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print("RMSE:", rmse)
```

```
RMSE: 0.8398584529459426
```

Two Spirals

In [17]:
```
import numpy as np
import pandas as pd

# Define the parameters of the Gaussian kernel
sigma = 1.0
gamma = 1.0 / (2.0 * sigma ** 2)

# Step 1: Calculate the kernel matrix K
n_samples = len(df_spiral)
K = np.zeros((n_samples, n_samples))
for i in range(n_samples):
    for j in range(n_samples):
        x_i = df_spiral.iloc[i].values
        x_j = df_spiral.iloc[j].values
        K[i, j] = np.exp(-gamma * np.linalg.norm(x_i - x_j) ** 2)

# Step 2: Center the kernel matrix K
one_n = np.ones((n_samples, n_samples)) / n_samples
K_centered = K - one_n.dot(K) - K.dot(one_n) + one_n.dot(K).dot(one_n)

# Step 3: Compute the eigenvectors and eigenvalues of the centered kernel matr
eigenvalues, eigenvectors = np.linalg.eig(K_centered)

# Step 4: Select the top T eigenvectors and compute the new representation of
T = 3 # number of principal components
idx = eigenvalues.argsort()[::-1][:T]
eigenvectors = eigenvectors[:, idx]
new_data = np.dot(K_centered, eigenvectors) / np.sqrt(eigenvalues[idx])

# Print the shape of the new data representation
print("Shape of the new data representation:", new_data.shape)
```

```
Shape of the new data representation: (1000, 3)
```

In [18]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import mean_squared_error

# Define the parameters of the Gaussian kernel
sigma = 1.0
gamma = 1.0 / (2.0 * sigma ** 2)

# Step 1: Calculate the kernel matrix K
n_samples = len(df_spiral)
K = np.zeros((n_samples, n_samples))
for i in range(n_samples):
    for j in range(n_samples):
        x_i = df_spiral.iloc[i].values
        x_j = df_spiral.iloc[j].values
        K[i, j] = np.exp(-gamma * np.linalg.norm(x_i - x_j) ** 2)

# Step 2: Center the kernel matrix K
one_n = np.ones((n_samples, n_samples)) / n_samples
K_centered = K - one_n.dot(K) - K.dot(one_n) + one_n.dot(K).dot(one_n)

# Step 3: Compute the eigenvectors and eigenvalues of the centered kernel matr
eigenvalues, eigenvectors = np.linalg.eig(K_centered)

# Try different values of D and evaluate the performance of linear regression
for D in [3, 20, 100]:
    # Select the top D eigenvectors and compute the new representation of the
    idx = eigenvalues.argsort()[::-1][:D]
    eigenvectors_D = eigenvectors[:, idx]
    new_data = np.dot(K_centered, eigenvectors_D) / np.sqrt(eigenvalues[idx])

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(new_data, df_spiral.il

    # Fit a linear regression model on the training set using 10-fold cross-va
    model = LinearRegression()
    scores = cross_val_score(model, X_train, y_train, cv = 10, scoring='neg_me
    rmse = np.sqrt(-scores.mean())

    print(f"D = {D}, RMSE = {rmse}")
```

```
D = 3, RMSE = 0.8751436955790198
D = 20, RMSE = 0.37583346410762714
D = 100, RMSE = 0.09169852784284811
```

Three Circles

In [19]:
```
import numpy as np
import pandas as pd

# Define the parameters of the Gaussian kernel
sigma = 1.0
gamma = 1.0 / (2.0 * sigma ** 2)

# Step 1: Calculate the kernel matrix K
n_samples = len(df_circle)
K = np.zeros((n_samples, n_samples))
for i in range(n_samples):
    for j in range(n_samples):
        x_i = df_circle.iloc[i].values
        x_j = df_circle.iloc[j].values
        K[i, j] = np.exp(-gamma * np.linalg.norm(x_i - x_j) ** 2)

# Step 2: Center the kernel matrix K
one_n = np.ones((n_samples, n_samples)) / n_samples
K_centered = K - one_n.dot(K) - K.dot(one_n) + one_n.dot(K).dot(one_n)

# Step 3: Compute the eigenvectors and eigenvalues of the centered kernel matr
eigenvalues, eigenvectors = np.linalg.eig(K_centered)

# Step 4: Select the top T eigenvectors and compute the new representation of
T = 3 # number of principal components
idx = eigenvalues.argsort()[::-1][:T]
eigenvectors = eigenvectors[:, idx]
new_data = np.dot(K_centered, eigenvectors) / np.sqrt(eigenvalues[idx])

# Print the shape of the new data representation
print("Shape of the new data representation:", new_data.shape)
```

```
Shape of the new data representation: (1000, 3)
```

In [23]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import mean_squared_error

# Define the parameters of the Gaussian kernel
sigma = 1.0
gamma = 1.0 / (2.0 * sigma ** 2)

# Step 1: Calculate the kernel matrix K
n_samples = len(df_spiral)
K = np.zeros((n_samples, n_samples))
for i in range(n_samples):
    for j in range(n_samples):
        x_i = df_spiral.iloc[i].values
        x_j = df_spiral.iloc[j].values
        K[i, j] = np.exp(-gamma * np.linalg.norm(x_i - x_j) ** 2)

# Step 2: Center the kernel matrix K
one_n = np.ones((n_samples, n_samples)) / n_samples
K_centered = K - one_n.dot(K) - K.dot(one_n) + one_n.dot(K).dot(one_n)

# Step 3: Compute the eigenvectors and eigenvalues of the centered kernel matr
eigenvalues, eigenvectors = np.linalg.eig(K_centered)

# Try different values of D and evaluate the performance of linear regression
for D in [3, 20, 100]:
    # Select the top D eigenvectors and compute the new representation of the
    idx = eigenvalues.argsort()[::-1][:D]
    eigenvectors_D = eigenvectors[:, idx]
    new_data = np.dot(K_centered, eigenvectors_D) / np.sqrt(eigenvalues[idx])

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(new_data, df_spiral.il

    # Fit a linear regression model on the training set using 100-fold cross-v
    model = LinearRegression()
    scores = cross_val_score(model, X_train, y_train, cv = 100, scoring='neg_m
    rmse = np.sqrt(-scores.mean())

    print(f"D = {D}, RMSE = {rmse}")
```

```
D = 3, RMSE = 0.8734927600018749
D = 20, RMSE = 0.37610237552700265
D = 100, RMSE = 0.0919786451129687
```

In [ ]: