Department of Electronic & Telecommunication Engineering
University of Moratuwa

EN3150 - PATTERN RECOGNITION

KERNEL METHODS

MANIMOHAN T. *200377M*

This report is submitted as Assignment - 04 of module EN3150
6^{th} November 2023

ABSTRACT

This report describe about the Project EN3160 - Image Processing and Machine Vision module, which conducted by Dr. Sampath Perera

In this assignment on Kernel Methods, the concept of feature space mapping and kernel functions is explored. It begins by deriving the kernel function for both one-dimensional and two-dimensional input spaces, accompanied by their respective mapping functions. The practical application of kernel methods is demonstrated through the utilization of a Linear Support Vector Classifier (Linear SVC) on datasets generated under different conditions, with classification accuracies reported for each scenario. The assignment emphasizes the importance of choosing appropriate kernel functions and mapping techniques, showcasing their impact on classification accuracy and serving as a practical introduction to Scikit-learn for implementing kernel methods.

1 LOGISTIC REGRESSION WEIGHT UPDATE PROCESS

1.1 Kernel Methods

1. Suppose that the input space to feature space mapping (projection) is given by the following function:

$$\Phi(x) = \left(1, \sqrt{2}x, x^2\right) \quad (1)$$

For a one-dimensional input space, show that the corresponding kernel function is:

$$k(x, z) = (1 + xz)^2$$

To find the kernel function for the given feature space mapping, we can use the definition of a kernel function, which is the inner product of the mapped feature vectors in the feature space. The kernel function is defined as:

$$k(x, z) = \Phi(x) \cdot \Phi(z) \quad (1)$$

In this case, the feature space mapping $\Phi(x)$ is given as:

$$\Phi(x) = \left(1, \sqrt{2}x, x^2\right) \quad (2)$$

And $\Phi(z)$ would be:

$$\Phi(z) = \left(1, \sqrt{2}z, z^2\right) \quad (3)$$

Now, we can calculate the inner product of these two feature vectors:

$$\begin{aligned} \Phi(x) \cdot \Phi(z) &= \left(1, \sqrt{2}x, x^2\right) \cdot \left(1, \sqrt{2}z, z^2\right) \\ &= (1 \cdot 1) + (\sqrt{2}x \cdot \sqrt{2}z) + (x^2 \cdot z^2) \\ &= 1 + 2xz + x^2z^2 \end{aligned} \quad (4)$$

Now, we need to simplify this expression. Notice that the middle term $2xz$ is exactly what we have in the desired kernel function $k(x, z)$. So, our kernel function is:

$$k(x, z) = 1 + 2xz + x^2z^2 \quad (5)$$

Now, let's factor this expression:

$$k(x, z) = (1 + xz)^2 \quad (6)$$

So, the corresponding kernel function for the given feature space mapping is:

$$k(x, z) = (1 + xz)^2 \quad (7)$$

This is the kernel function for a one-dimensional input space corresponding to the given feature space mapping $\Phi(x)$.

2. Express the kernel function provided above for a two-dimensional input space, where $x = (x_1, x_2)$ and $z = (z_1, z_2)$?
   For a two-dimensional input space, where $x = (x_1, x_2)$ and $z = (z_1, z_2)$, you can generalize the kernel function obtained in the previous response. The kernel function is given by:

$$k(x, z) = (1 + xz)^2$$

   In this case, the input vectors x and z have two components, so we need to express the kernel function for this two-dimensional input space. We can expand the expression as follows:

$$k(x, z) = (1 + x_1 z_1 + x_2 z_2)^2$$

   This is the kernel function for a two-dimensional input space using the given feature space mapping $\Phi(x)$. We calculate the inner product of $\Phi(x)$ and $\Phi(z)$ in this high-dimensional feature space and obtain the squared result.

3. What is the mapping function $\Phi(x)$ for the kernel function provided above in the above question (Q2)?
   The kernel function provided in the previous question is:

$$k(x, z) = (1 + x_1 z_1 + x_2 z_2)^2$$

   To find the mapping function $\Phi(x)$ corresponding to this kernel function, we can consider the expansion of the kernel function. The mapping function $\Phi(x)$ should map the input vector x to a high-dimensional feature space in such a way that the inner product of $\Phi(x)$ and $\Phi(z)$ is equal to the kernel function $k(x, z)$.

The expansion of the kernel function suggests the following mapping function $\Phi(x)$ for the two-dimensional input space:

$$\Phi(x) = \left(1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2\right)$$

In this mapping, we have included the constant term (1), each component of x scaled by $\sqrt{2}$, the square of each component of x, and the cross-product term $\sqrt{2}x_1 x_2$. This mapping allows us to calculate the inner product of $\Phi(x)$ and $\Phi(z)$ in the high-dimensional feature space, and when squared, it matches the kernel function $k(x, z)$.

4. Consider the kernel $k = (1 + x^T z)^2$. For the following dataset, determine the kernel matrix (gram matrix). Is this a valid kernel for this dataset? Justify your answer.

$$G = \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & \dots & k(x_1, x_N) \\ k(x_2, x_1) & k(x_2, x_2) & \dots & k(x_2, x_N) \\ \vdots & \vdots & \ddots & \vdots \\ k(x_N, x_1) & k(x_N, x_2) & \dots & k(x_N, x_N) \end{bmatrix}$$

Sample index	Data sample ($x = (x_1, x_2)$)	Feature 1 (x_1)	Feature 2 (x_2)
1	x_1	1	5
2	x_2	3	4
3	x_3	4	2
4	x_4	10	12

Table 1 — Data Sample Table

To determine if the given kernel $k = (1 + x^T z)^2$ is valid for the dataset, we need to calculate the kernel matrix (gram matrix) G for the provided data samples. The kernel matrix is given by the formula:

$$G = \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & \dots & k(x_1, x_N) \\ k(x_2, x_1) & k(x_2, x_2) & \dots & k(x_2, x_N) \\ \vdots & \vdots & \ddots & \vdots \\ k(x_N, x_1) & k(x_N, x_2) & \dots & k(x_N, x_N) \end{bmatrix}$$

We have four data samples with features $x_1 = (1, 5)$, $x_2 = (3, 4)$, $x_3 = (4, 2)$, and $x_4 = (10, 12)$.

Now, let's calculate the entries of the kernel matrix using the given kernel function $k(x, z) = (1 + x^T z)^2$:

$$\begin{aligned}
1. G[1][1] &= k(x_1, x_1) = (1 + x_1^T x_1)^2 \\
G[1][1] &= (1 + (1^2 + 5^2))^2 = (1 + 26)^2 = 27^2 = 729 \\
2. G[1][2] &= k(x_1, x_2) = (1 + x_1^T x_2)^2 \\
G[1][2] &= (1 + (1 \cdot 3 + 5 \cdot 4))^2 = (1 + 23)^2 = 24^2 = 576 \\
3. G[1][3] &= k(x_1, x_3) = (1 + x_1^T x_3)^2 \\
G[1][3] &= (1 + (1 \cdot 4 + 5 \cdot 2))^2 = (1 + 14)^2 = 15^2 = 225 \\
4. G[1][4] &= k(x_1, x_4) = (1 + x_1^T x_4)^2 \\
G[1][4] &= (1 + (1 \cdot 10 + 5 \cdot 12))^2 = (1 + 70)^2 = 71^2 = 5041 \\
5. G[2][1] &= k(x_2, x_1) = k(x_1, x_2) = 576 \\
6. G[2][2] &= k(x_2, x_2) = (1 + x_2^T x_2)^2 \\
G[2][2] &= (1 + (3^2 + 4^2))^2 = (1 + 25)^2 = 26^2 = 676 \\
7. G[2][3] &= k(x_2, x_3) = (1 + x_2^T x_3)^2 \\
G[2][3] &= (1 + (3 \cdot 4 + 4 \cdot 2))^2 = (1 + 20)^2 = 21^2 = 441 \\
8. G[2][4] &= k(x_2, x_4) = (1 + x_2^T x_4)^2 \\
G[2][4] &= (1 + (3 \cdot 10 + 4 \cdot 12))^2 = (1 + 100)^2 = 101^2 = 10201 \\
9. G[3][1] &= k(x_3, x_1) = k(x_1, x_3) = 225 \\
10. G[3][2] &= k(x_3, x_2) = k(x_2, x_3) = 441
\end{aligned}$$

11. $G[3][3] = k(x_3, x_3) = (1 + x_3^T x_3)^2$

$G[3][3] = (1 + (4^2 + 2^2))^2 = (1 + 20)^2 = 21^2 = 441$

12. $G[3][4] = k(x_3, x_4) = (1 + x_3^T x_4)^2$

$G[3][4] = (1 + (4 \cdot 10 + 2 \cdot 12))^2 = (1 + 64)^2 = 65^2 = 4225$

13. $G[4][1] = k(x_4, x_1) = k(x_1, x_4) = 5041$

14. $G[4][2] = k(x_4, x_2) = k(x_2, x_4) = 10201$

15. $G[4][3] = k(x_4, x_3) = k(x_3, x_4) = 4225$

16. $G[4][4] = k(x_4, x_4) = (1 + x_4^T x_4)^2$

$G[4][4] = (1 + (10^2 + 12^2))^2 = (1 + 244)^2 = 245^2 = 60025$

And similarly for other entries in the matrix. Calculating all entries of the kernel matrix:

$$G = \begin{bmatrix} 729 & 576 & 225 & 5041 \\ 576 & 676 & 441 & 10201 \\ 225 & 441 & 441 & 4225 \\ 5041 & 10201 & 4225 & 60025 \end{bmatrix}$$

Now, to determine if this is a valid kernel, we need to check whether the resulting kernel matrix is positive semi-definite. A matrix is positive semi-definite if all of its eigenvalues are non-negative. We can compute the eigenvalues of G and check if they are all non-negative. Now, we'll calculate the eigenvalues by solving the characteristic equation:

$$\det(G - \lambda I) = 0$$

Where I is the 4x4 identity matrix.
First, subtract λI from G:

$$G - \lambda I = \begin{bmatrix} 729 - \lambda & 576 & 225 & 5041 \\ 576 & 676 - \lambda & 441 & 10201 \\ 225 & 441 & 441 - \lambda & 4225 \\ 5041 & 10201 & 4225 & 60025 - \lambda \end{bmatrix}$$

Now, calculate the determinant of this matrix:

$$\det(G - \lambda I) = (729 - \lambda)((676 - \lambda)((441 - \lambda)(60025 - \lambda) - 4225 \cdot 10201) - 441 \cdot (10201 \cdot 5041$$

$$-4225 \cdot 576)) - 576((576 - \lambda)((441 - \lambda)(60025 - \lambda) - 4225 \cdot 5041) - 225 \cdot (10201 \cdot 5041 - 4225 \cdot 5041)) + \ldots$$

Then calculate the eigen values using coding.

Listing 1.1 — calculate the eigen values

```
import numpy as np

# Define the matrix G
G = np.array([
    [729, 576, 225, 5041],
```

```
    [576, 676, 441, 10201],
    [225, 441, 441, 4225],
    [5041, 10201, 4225, 60025]
])

# Calculate eigenvalues
eigenvalues = np.linalg.eigvals(G)

# Print the eigenvalues
for eigenvalue in eigenvalues:
    print(eigenvalue)
```

```
Eigenvalues of G:
62437.99699452922
-1137.8088759182488
389.5936824902826
181.2181988987965
```

If not all eigenvalues are non-negative, then the matrix G cannot be positive semi-definite.

The kernel matrix G provided is not a valid kernel matrix because it's not positive semi-definite. This means that the kernel $k = (1 + x^T z)^2$ is not a valid kernel for this dataset.

5. Use the code given in Listing 1 to generate data.

Listing 1.2 — Data generation

```
import numpy as np
import matplotlib . pyplot as plt
from sklearn . datasets import make_circles
# Generate data with make_circles
np . random . seed (5)
X , y = make_circles ( n_samples =500 , factor =0.3 , noise =0.1)
```

a) Plot the scatter plot of the data:

$$\text{plt.scatter}(X[:,0], X[:,1], c = y, cmap = plt.cm.Paired)$$

Listing 1.3 — Scatter plot

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_circles

# Generate data with make_circles
```

```
np.random.seed(5)
X, y = make_circles(n_samples=500, factor=0.3, noise=0.1)

# Plot the scatter plot of the data
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)
plt.title('Scatter Plot of Data')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')

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

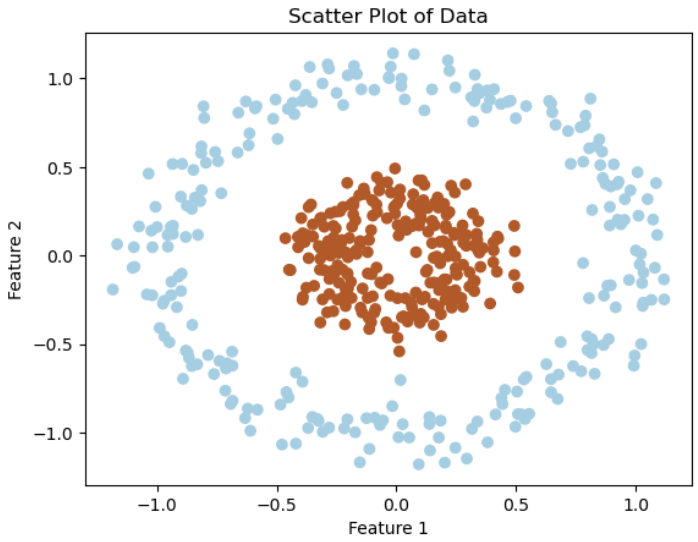

Figure 1 — scatter plot

b) Use the following mapping to map two-dimensional space to three-dimensional space (feature space). This feature space set is known as the projected set. Visualize the projected set:

$$\Phi : x = (x_1, x_2) \rightarrow \Phi(x) = \left(x_1, x_2, x_1^2 + x_2^2\right) \in R^3$$

Listing 1.4 — Map two-dimensional space to three-dimensional space

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D  # Import 3D plotting
functionality

# Generate the data in 2D space
np.random.seed(5)
X, y = make_circles(n_samples=500, factor=0.3, noise=0.1)

# Define the mapping function Φ
def phi_mapping(x):
    x1, x2 = x
```

```
    x3 = x1**2 + x2**2
    return np.array([x1, x2, x3])

# Apply the mapping to all data points
X_projected = np.apply_along_axis(phi_mapping, axis=1, arr=X)

# Create a 3D scatter plot to visualize the projected set
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Scatter plot in 3D
ax.scatter(X_projected[:, 0], X_projected[:, 1], X_projected[:, 2],
c=y, cmap=plt.cm.Paired)
ax.set_title('Projected Set in 3D Space')
ax.set_xlabel('Feature 1')
ax.set_ylabel('Feature 2')
ax.set_zlabel('Feature 3')

# Show the 3D plot
plt.show()
```

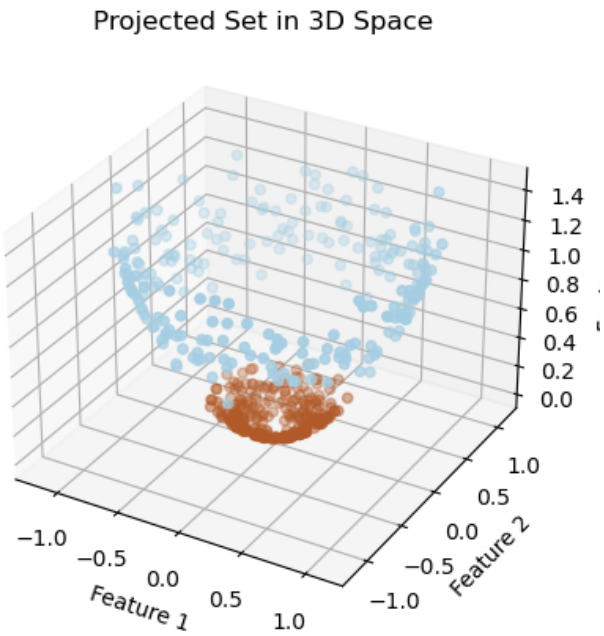

Figure 2 — map two-dimensional space to three-dimensional space

c) Now change the ”factor=0.5” and observe the visualization of the feature space in 3D. What changes can you observe?

Listing 1.5 — Map two-dimensional space to three-dimensional space at factor

```
import numpy as np
import matplotlib.pyplot as plt
```

```
from mpl_toolkits.mplot3d import Axes3D  # Import 3D plotting
functionality

# Generate the data in 2D space
np.random.seed(5)
X, y = make_circles(n_samples=500, factor=0.5, noise=0.1)

# Define the mapping function Φ
def phi_mapping(x):
    x1, x2 = x
    x3 = x1**2 + x2**2
    return np.array([x1, x2, x3])

# Apply the mapping to all data points
X_projected = np.apply_along_axis(phi_mapping, axis=1, arr=X)

# Create a 3D scatter plot to visualize the projected set
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Scatter plot in 3D
ax.scatter(X_projected[:, 0], X_projected[:, 1], X_projected[:, 2],
c=y, cmap=plt.cm.Paired)
ax.set_title('Projected Set in 3D Space')
ax.set_xlabel('Feature 1')
ax.set_ylabel('Feature 2')
ax.set_zlabel('Feature 3')

# Show the 3D plot
plt.show()
```

Observations:

* Changing the Factor to 0.5 affects the distribution of the data points in the 2D space. A smaller Factor value results in a tighter circle.
* The mapping function Φ remains the same, and it projects the 2D data into a 3D space by adding the square of the Euclidean distance from the origin $(x_1^2 + x_2^2)$ as the third feature.
* In the 3D feature space, the data points are still mapped in a circular pattern, but the shape and spread of the points in the third dimension will vary based on the 2D data distribution. With a Factor of 0.5, the third dimension will have values closer to the center of the circle.
* The visualization of the projected set in 3D will show a different pattern compared to when the Factor was 0.3, with a tighter circular distribution in the third dimension.

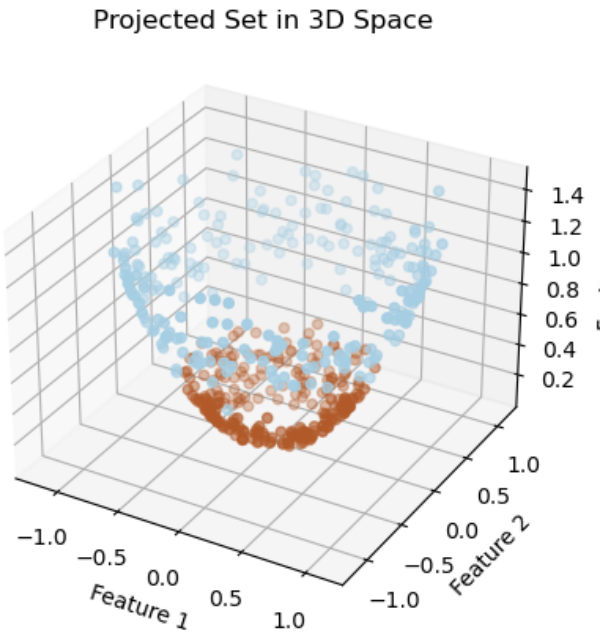

Figure 3 — map two-dimensional space to three-dimensional space, factor = 0.5

For the same data, use this mapping:

$$x = (x_1, x_2) \rightarrow \left(x_1^2, x_2^2, x_1 x_2\right)$$

Is this mapping better than the previous mapping?

Listing 1.6 — Map two-dimensional space to three-dimensional space

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Generate the data in 2D space with a different factor
np.random.seed(5)
X, y = make_circles(n_samples=500, factor=0.5, noise=0.1)

# Define the new mapping function x = (x1, x2) → (x1^2, x2^2, x1*x2)
def phi_mapping_new(x):
    x1, x2 = x
    return np.array([x1**2, x2**2, x1 * x2])

# Apply the new mapping to all data points
X_projected_new = np.apply_along_axis(phi_mapping_new, axis=1, arr=X)

# Create a 3D scatter plot to visualize the new projected set
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Scatter plot in 3D
ax.scatter(X_projected_new[:, 0], X_projected_new[:, 1],
```

```
X_projected_new[:, 2], c=y, cmap=plt.cm.Paired)
ax.set_title('Projected Set in 3D Space (New Mapping)')
ax.set_xlabel('Feature 1 (x1^2)')
ax.set_ylabel('Feature 2 (x2^2)')
ax.set_zlabel('Feature 3 (x1*x2)')

# Show the 3D plot with the new mapping
plt.show()
```

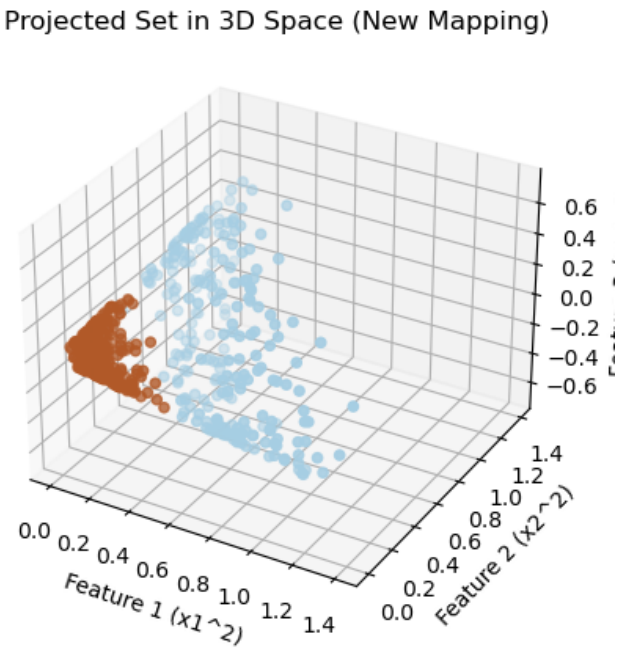

Figure 4 — map two-dimensional space to three-dimensional space

Whether one mapping is *better* than another depends on the specific goals and requirements of our analysis or machine learning task. The choice of a mapping function should be guided by the problem we are trying to solve and the characteristics of our data. Let's compare the two mappings:

- **Φ Mapping (Original Mapping):**
- Original features (x_1, x_2) are transformed into $(x_1, x_2, x_1^2 + x_2^2)$.
- The third feature represents the squared Euclidean distance from the origin.
- It separates the data into concentric circles.
- **New Mapping:**
- Original features (x_1, x_2) are transformed into $(x_1^2, x_2^2, x_1 \cdot x_2)$.
- The new mapping captures the squares of the original features and their interaction.
- The transformed feature space captures different aspects of the data.

Which mapping is *better* depends on the task at hand. Here are some considerations:

- **Linear Separability:** The new mapping captures interaction between the features, which may be useful for linear separation in the feature space. If our classification problem benefits from this, the new mapping could be better.

- **Non-linear Separability:** The original mapping emphasizes concentric circles, which might be useful if the decision boundary in the original feature space is non-linear. If this is the case, the original mapping could be better.
- **Simplicity:** The original mapping introduces only one new feature, making the feature space simpler. The new mapping introduces two new features. Simplicity can be advantageous in certain contexts.
- **Overfitting:** Adding too many features, especially if they are highly correlated, can lead to overfitting. We should consider the dimensionality of our dataset and the potential for overfitting when choosing a mapping.

In practice, we may want to experiment with both mappings and evaluate their performance in our specific machine learning or analysis pipeline. The choice of mapping depends on the characteristics of our data and the algorithms we plan to use for our task.

d) Run linear SVC (`svm.SVC(kernel='linear')` on the original dataset generated based on Listing 1 and the projected set using the mapping given in 5b, and report the classification accuracies for both cases.

Listing 1.7 — Classification accuracies for both cases

```
import numpy as np
from sklearn.datasets import make_circles
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Generate the original dataset
np.random.seed(5)
X_original, y_original = make_circles(n_samples=500, factor=0.3,
noise=0.1)

# Split the original dataset into training and testing sets
X_train_original, X_test_original, y_train_original,
y_test_original = train_test_split(X_original, y_original,
test_size=0.2, random_state=42)

# Train a linear SVM on the original dataset
svm_original = SVC(kernel='linear')
svm_original.fit(X_train_original, y_train_original)

# Make predictions on the test set for the original dataset
y_pred_original = svm_original.predict(X_test_original)

# Calculate the accuracy for the original dataset
accuracy_original = accuracy_score(y_test_original, y_pred_original)
```

```
print("Accuracy on the original dataset:", accuracy_original)

# Now, let's apply the same procedure to the projected set

# Define the mapping function x = (x1, x2) → (x1^2, x2^2, x1*x2)
def phi_mapping(x):
    x1, x2 = x
    x3 = x1**2 + x2**2
    return np.array([x1, x2, x3])

# Apply the new mapping to all data points
X_projected_new = np.apply_along_axis(phi_mapping_new,
axis=1, arr=X_original)

# Split the projected set into training and testing sets
X_train_projected, X_test_projected, y_train_projected,
y_test_projected = train_test_split(X_projected_new,
y_original, test_size=0.2, random_state=42)

# Train a linear SVM on the projected set
svm_projected = SVC(kernel='linear')
svm_projected.fit(X_train_projected, y_train_projected)

# Make predictions on the test set for the projected set
y_pred_projected = svm_projected.predict(X_test_projected)

# Calculate the accuracy for the projected set
accuracy_projected = accuracy_score(y_test_projected, y_pred_projected)
print("Accuracy on the projected set:", accuracy_projected)
```

```
Accuracy on the original dataset: 0.45
Accuracy on the projected set: 1.0
```

Github Repository Link :- PATTERN-RECOGNITION-Assignment-04