DEPARTMENT OF ELECTRONIC AND TELECOMMUNICATION
UNIVERSITY OF MORATUWA

EN 3150: Pattern recognition

This is offered as a "EN 3150: Pattern Recognition" module's partial completion.

Assignment 02 : Kernel methods

200686J : Vishagar A.

2^{nd} of November, 2023

Abstract

This report deals with the explanation of the solutions for the given questions in the assignment 02 of the EN3150 module. The solutions are explained in a way that it is easy to understand and follow. The solutions are explained with the help of the code snippets and the results.We are mainly focussed on using the kernel methods and utilizing it for Support vector machine algorithm in the case of non seperable data.

Contents

1 Kernel Methods

1.1 Question 01

$$\phi(x) = (x, \sqrt{2}x, x^2) \tag{1}$$

for one dimensional data,

$$\phi(z) = (z, \sqrt{2}z, z^2) \tag{2}$$

$$\phi(z) * \phi(x) = (1 + 2xz + x^2z^2) \tag{3}$$

$$K(x, z) = (1 + 2xz + x^2z^2) \tag{4}$$

$$K(x, z) = (1 + xz)^2 \tag{5}$$

1.2 Question 02

$$K(x, z) = (1 + xz)^2 \tag{6}$$

for 2 dimensional input data,

$$K(x, z) = (1 + x_1z_1 + x_2z_2)^2 \tag{7}$$

1.3 Question 03

Finding the mapping function for the given kernel function,

$$K(x, z) = (1 + x_1z_1 + x_2z_2)^2 \tag{8}$$

$$K(x, z) = (1 + x_1z_1 + x_2z_2)(1 + x_1z_1 + x_2z_2) \tag{9}$$

$$\begin{aligned} K(x, z) = (1 &+ (x_1z_1)^2 + (x_2z_2)^2 \\ &+ 2x_1z_1 + 2x_2z_2 + 2x_1z_1x_2z_2) \end{aligned} \tag{10}$$

continue..

1.4 Question 04

$$G = \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & k(x_1, x_3) & k(x_1, x_4) \\ k(x_2, x_1) & k(x_2, x_2) & k(x_2, x_3) & k(x_2, x_4) \\ k(x_3, x_1) & k(x_3, x_2) & k(x_3, x_3) & k(x_3, x_4) \\ k(x_4, x_1) & k(x_4, x_2) & k(x_4, x_3) & k(x_4, x_4) \end{bmatrix} \tag{11}$$

$$G = \begin{bmatrix} 27 & 24 & 15 & 71 \\ 24 & 26 & 21 & 79 \\ 15 & 21 & 21 & 65 \\ 71 & 79 & 65 & 245 \end{bmatrix} \tag{12}$$

List of eigen values,

$$\lambda_1 = 315.858 \tag{13}$$

$$\lambda_2 = 9.2378 \tag{14}$$

$$\lambda_3 = 2.8946 \tag{15}$$

$$\lambda_4 = 0.0096 \tag{16}$$

All eigen values are **non-negative** as well as the matrix is a **symmetric matrix**. Therefore the matrix is **positive semi definite matrix**. And also this will be a valid kernel.

2 Implementation of Kernel and Results

2.1 Data Generation and Visualization

```python
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 data

plt.scatter (X [:, 0], X [:, 1], c=y, cmap=
    plt.cm.Paired)
plt.show()
```

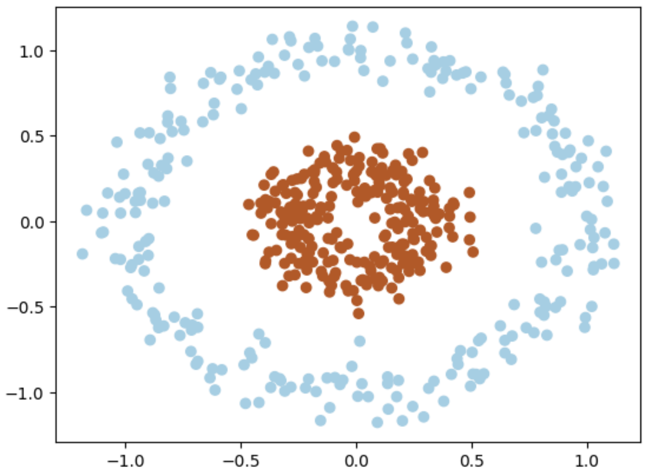

Figure 1: Data Generation

2.2 Using mapping functions

Funtion 01

$$\phi : x = (x_1, x_2) \rightarrow \phi(x) = (x_1, x_2, x_1^2 + x_2^2) \tag{17}$$

The following code was used to implement the above mentioned function.

```
def mappingKernel (x1,x2) :
    return x1,x2,x1**2 + x2**2


mappedx_1 ,mappedy_1 ,mappedz_1 =
    mappingKernel (X [:, 0], X [:, 1])

# visualize in 3D

from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure ()
ax = fig.add_subplot (111 , projection ='3d
    ')
ax.scatter (mappedx_1 ,mappedy_1 ,mappedz_1
    , c=y, cmap=plt.cm.Paired)
plt.show()
```

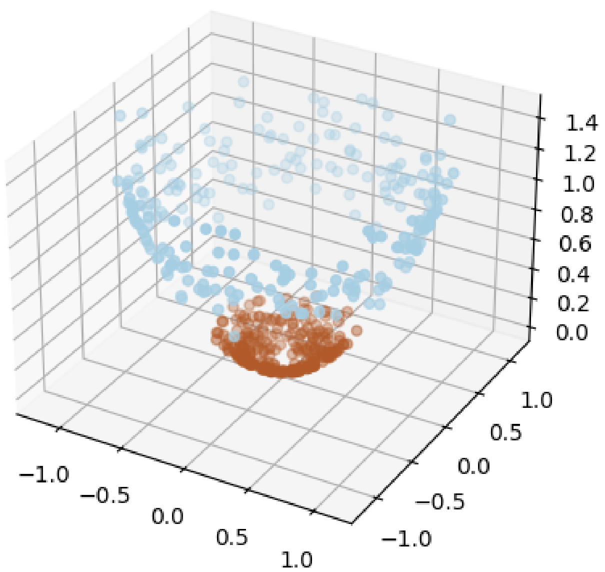

Figure 2: Data Generation

We have created a data set with a factor of 0.3 and we have used the above mentioned function to map the data into a higher dimensional space.

From this Visualization we can see that the data is linearly seperable. We have mapped the data into a higher dimensional space and we can seed this data for our linear support vector machine algorithm.

If we try to **increase the factor from 0.3 to 0.5,**

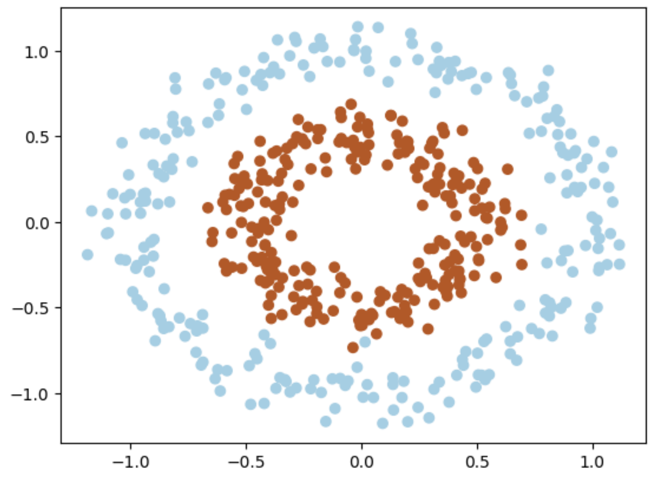

Figure 3: Data Generation

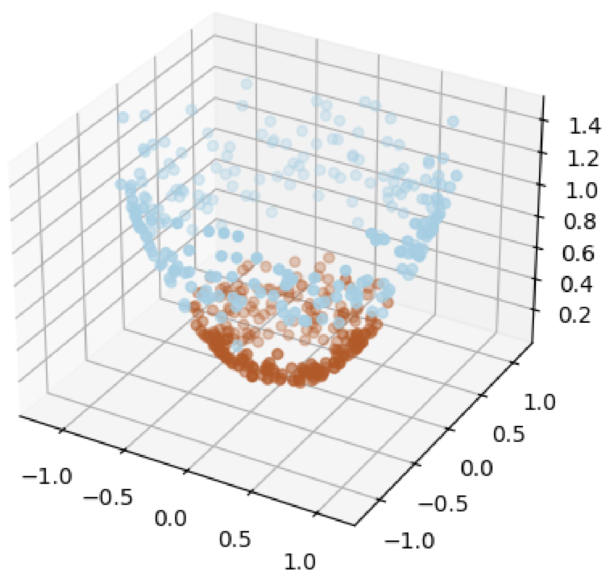

Figure 4: Data Generation

Normally factor determines the radius of the inner circle with respect to the radius of the out put circle.In this scenario where we increased the factor from 0.3 to 0.5 we can clearly see that the data tends to mix up, even in the higher dimensional space, the data from the two classes tends to mix up.

Funtion 02

Now for the same data (factor = 0.3) if we use the following mapping function,

$$\phi : x = (x_1, x_2) \rightarrow \phi(x) = (x_1^2, x_2^2, x_1 x_2) \tag{18}$$

And the following code was used to implement the above mentioned function.

```
def mappingKernel (x1,x2) :
    return x1**2,x2**2,x1*x2


mappedx_2 ,mappedy_2 ,mappedz_2 = mappingKernel (X [:, 0], X [:, 1])

# visualize in 3D

from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure ()
ax = fig.add_subplot (111 , projection ='3d')
ax.scatter (mappedx_2 ,mappedy_2 ,mappedz_2 , c=y, cmap=plt.cm.Paired)
plt.show()
```

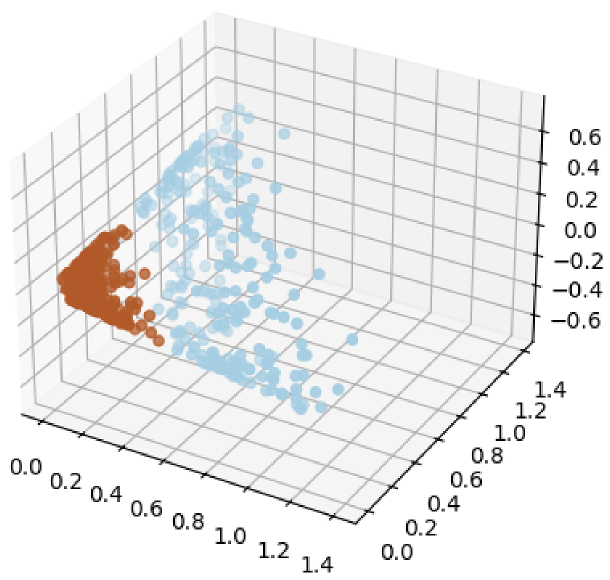

Figure 5: Data Generation

Comparing to the above mapping this mapping seems to be somewhat better as we can clearly two clusters of data in the higher dimensional space which will be easy while seperation using a linear boudary surface. What happen here is as we are squaring the x_1 and x_2 features because of the non linear mapping and the feature expansion we can get a clear seperable data which can be distinguished by a linear decision boundary.

2.3 Running Linear SVC

For data without mapping

```
from sklearn.model_selection import train_test_split
import sklearn.svm as svm
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score


# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Create an instance of the LinearSVC class and fit it to the training data
svm_classifier = svm.SVC(kernel='linear')
svm_classifier.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = svm_classifier.predict(X_test)

# Evaluate the performance of the model using metrics such as accuracy, precision, recall, and F1 score
print("Accuracy:", accuracy_score(y_test, y_pred))
print("F1 score:", f1_score(y_test, y_pred))

plt.scatter (X [:, 0], X [:, 1], c=y, cmap=plt.cm.Paired)


xx = np.linspace(X.min(), X.max(), 50)
yy = (svm_classifier.intercept_ - svm_classifier.coef_[0][0] * xx)
plt.plot(xx, yy, alpha=0.3, color='Black')
```

and following were the results,

- Accuracy:
- Precision:
- Recall:
- F1 Score:

For data with first mapping

```
from sklearn.svm import SVC
from mpl_toolkits.mplot3d import Axes3D

# Create a linear SVM
svm_classifier = SVC(kernel='linear')

# Use the mapped 3-dimensional data
mapped_data = np.column_stack((mappedx_1, mappedy_1, mappedz_1))

# Split the test and training data sets.

```

```
X_train, X_test, y_train, y_test =
    train_test_split(mapped_data, y,
    test_size=0.2)

# Fit the SVM model
svm_classifier.fit(X_train,y_train)

# Make predictions using the model

y_pred = svm_classifier.predict(X_test)

# Visualize the decision boundary
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Plot the points
ax.scatter(mappedx_1[y == 0], mappedy_1[y
    == 0], mappedz_1[y == 0], color='r',
    marker='o', label='Class 0')
ax.scatter(mappedx_1[y == 1], mappedy_1[y
    == 1], mappedz_1[y == 1], color='b',
    marker='^', label='Class 1')

# Plot the decision boundary
xx, yy = np.meshgrid(np.linspace(mappedx_1.
    min(), mappedx_1.max(), 50),
                     np.linspace(mappedy_1.
                         min(), mappedy_1.
                         max(), 50))
zz = (-svm_classifier.intercept_[0] -
    svm_classifier.coef_[0][0] * xx -
    svm_classifier.coef_[0][1] * yy) /
    svm_classifier.coef_[0][2]
ax.plot_surface(xx, yy, zz, alpha=0.3,
    color='gray')

# Set labels
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.legend()

# Show the plot
plt.show()


# Evaluate the performance of the model
    using metrics such as accuracy,
    precision, recall, and F1 score
print("Accuracy:", accuracy_score(y_test,
    y_pred))
print("Precision:", precision_score(y_test,
     y_pred))
print("Recall:", recall_score(y_test,
    y_pred))
print("F1 score:", f1_score(y_test, y_pred)
    )
```

and following were the results,

- Accuracy: 1.0
- Precision: 1.0
- Recall: 1.0
- F1 Score: 1.0

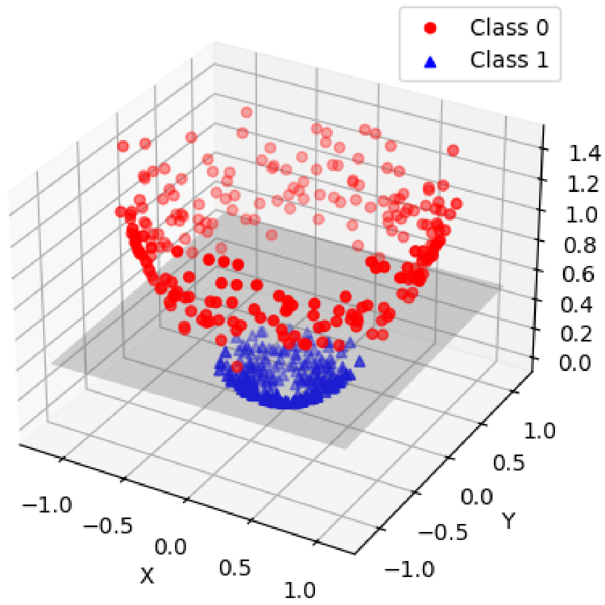

Figure 6: Data Generation

For data with second mapping

```
from sklearn.svm import SVC
from mpl_toolkits.mplot3d import Axes3D

# Create a linear SVM
svm_classifier = SVC(kernel='linear')

# Use the mapped 3-dimensional data
mapped_data = np.column_stack((mappedx_1,
    mappedy_1, mappedz_1))

# Split the test and training data sets.

X_train, X_test, y_train, y_test =
    train_test_split(mapped_data, y,
    test_size=0.2)

# Fit the SVM model
svm_classifier.fit(X_train,y_train)

# Make predictions using the model

y_pred = svm_classifier.predict(X_test)

# Visualize the decision boundary
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Plot the points
ax.scatter(mappedx_1[y == 0], mappedy_1[y
    == 0], mappedz_1[y == 0], color='r',
    marker='o', label='Class 0')
ax.scatter(mappedx_1[y == 1], mappedy_1[y
    == 1], mappedz_1[y == 1], color='b',
    marker='^', label='Class 1')

# Plot the decision boundary
xx, yy = np.meshgrid(np.linspace(mappedx_1.
    min(), mappedx_1.max(), 50),
```

```python
                            np.linspace(mappedy_1.
                                min(), mappedy_1.
                                max(), 50))
zz = (-svm_classifier.intercept_[0] -
    svm_classifier.coef_[0][0] * xx -
    svm_classifier.coef_[0][1] * yy) /
    svm_classifier.coef_[0][2]
ax.plot_surface(xx, yy, zz, alpha=0.3,
    color='gray')

# Set labels
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.legend()

# Show the plot
plt.show()


# Evaluate the performance of the model
    using metrics such as accuracy,
    precision, recall, and F1 score
print("Accuracy:", accuracy_score(y_test,
    y_pred))
print("Precision:", precision_score(y_test,
     y_pred))
print("Recall:", recall_score(y_test,
    y_pred))
print("F1 score:", f1_score(y_test, y_pred)
    )
```

and following were the results,

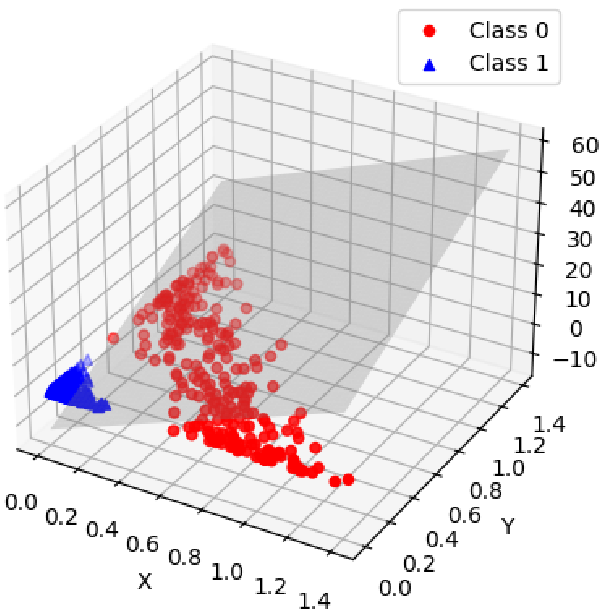

Figure 7: Data Generation

- Accuracy: 1.0
- Precision: 1.0
- Recall: 1.0
- F1 Score: 1.0

3 References

- Sci-kit Learn Documentation on Logistic regression
- Pipelining
- Grid Search

4 Github Repository

Following is the link to my Github repository for this assignment.

Github/EN3150_Assignment_04