# Assignment 04 - Kernel Methods

December 3, 2023

**Index No : 200462U**

**Name : Randika Perera**

https://github.com/randika-perera/EN3150-Pattern-Recognition/blob/main/Assignment%204/Assignment%2004%20Kernel%20Methods.ipynb

## 1 Kernel Methods

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

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

For a one-dimensional input space show that the corresponding kernel function is $k(x,z) = (1+xz)^2$.

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

* For a one-dimensional input space,

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

$\phi(x) \cdot \phi(z) = 1 + 2xz + x^2z^2$

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

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

2. Express the kernel function provided above for a two-dimensional input space, where $\boldsymbol{x} = (x_1, x_2)$ and $\boldsymbol{z} = (z_1, z_2)$.

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

* For a two-dimensional input space, where $\underline{x} = (x_1, x_2)$ and $\underline{z} = (z_1, z_2)$,

$$K(\underline{x},\underline{z}) = (1 + \underline{x} \cdot \underline{z})^2$$

$$K(\underline{x},\underline{z}) = (1 + x_1 z_1 + x_2 z_2)^2$$

3. What is the mapping function $\Phi(\boldsymbol{x})$ for the kernel function provided above in the above question (Q2).

Mapping function $\phi(\underline{x})$ ?

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

$$= (1 + x_1z_1 + x_2z_2)\cdot(1 + x_1z_1 + x_2z_2)$$

$$= 1 + x_1^2z_1^2 + x_2^2z_2^2 + 2x_1z_1 + 2x_2z_2 + 2x_1z_1x_2z_2$$

$$= (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2) \cdot (1, \sqrt{2}z_1, \sqrt{2}z_2, \sqrt{2}z_1z_2, z_1^2, z_2^2)$$

$$= \langle \phi(x_1, x_2) \cdot \phi(z_1, z_2) \rangle$$

$$\therefore \phi(\underline{x}) = \phi(x_1, x_2) = (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2)$$

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

| Sample index | Data sample ($\boldsymbol{x} = (x_1, x_2)$) | Feature 1 ($x_1$) | Feature 2 ($x_2$) |
|---|---|---|---|
| 1 | $\boldsymbol{x}_1$ | 1 | 5 |
| 2 | $\boldsymbol{x}_2$ | 3 | 4 |
| 3 | $\boldsymbol{x}_3$ | 4 | 2 |
| 4 | $\boldsymbol{x}_4$ | 10 | 12 |

Note:- The kernel matrix (gram matrix) is given by

$$\boldsymbol{G} = \begin{bmatrix} k(\boldsymbol{x}_1,\boldsymbol{x}_1) & k(\boldsymbol{x}_1,\boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_1,\boldsymbol{x}_N) \\ k(\boldsymbol{x}_2,\boldsymbol{x}_1) & k(\boldsymbol{x}_2,\boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_2,\boldsymbol{x}_N) \\ \vdots & \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_N,\boldsymbol{x}_1) & k(\boldsymbol{x}_N,\boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_N,\boldsymbol{x}_N) \end{bmatrix} \tag{2}$$

$$G = \begin{pmatrix} K(x_1,x_1) & K(x_1,x_2) & K(x_1,x_3) & K(x_1,x_2) \\ K(x_2,x_1) & K(x_2,x_2) & K(x_2,x_3) & K(x_2,x_2) \\ K(x_3,x_1) & K(x_3,x_2) & K(x_3,x_3) & K(x_3,x_2) \\ K(x_4,x_1) & K(x_4,x_2) & K(x_4,x_3) & K(x_4,x_2) \end{pmatrix}$$

$$G = \begin{pmatrix} 27^2 & 24^2 & 15^2 & 71^2 \\ 24^2 & 26^2 & 21^2 & 79^2 \\ 15^2 & 21^2 & 21^2 & 65^2 \\ 71^2 & 79^2 & 65^2 & 245^2 \end{pmatrix}$$

$$G = \begin{pmatrix} 729 & 576 & 225 & 5041 \\ 576 & 676 & 441 & 6241 \\ 225 & 441 & 441 & 4225 \\ 5041 & 6241 & 4225 & 60025 \end{pmatrix}$$

Eigen values of G are,

$\lambda_1 = 61,397.1$

$\lambda_2 = 382.21$

$\lambda_3 = 81.53$

$\lambda_4 = 10.12$

* All the eigen values we obtained are non-negative.
* Also G is a symmetric matrix.
* Therefore, Kernel matrix is positive semi-definite.
* Hence, this is a valid kernel.

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

```
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)
```

Listing 1: Data generation.

(a) Plot the scatter plot of the data (plt.scatter (X [:, 0], X [:, 1], c=y, cmap=plt.cm.Paired)).

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

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

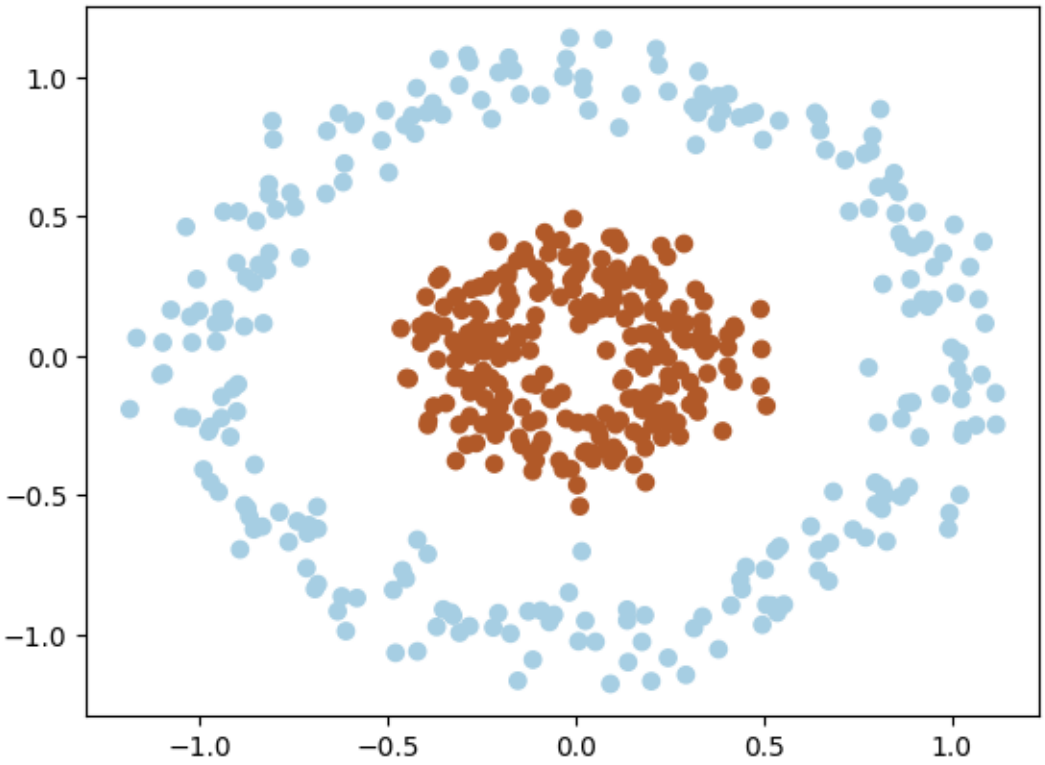
(c) Now change the "factor=0.5" and observe the visualize the feature space in 3-d. What changes can you observe?
For the same data, use this mapping: $\boldsymbol{x} = (x_1,\ x_2) \rightarrow \left(x_1^2,\ x_2^2,\ x_1 x_2\right)$. Is this mapping better than the previous mapping?

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

[ ]:
```
# Imports
import numpy as np
import matplotlib.pyplot as plt
import sklearn.svm as svm
from sklearn.datasets import make_circles
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score
```

[ ]:
```
# Generate data with make_circles
np.random.seed (5)
X,y = make_circles(n_samples=500,factor=0.3,noise=0.1)
```

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



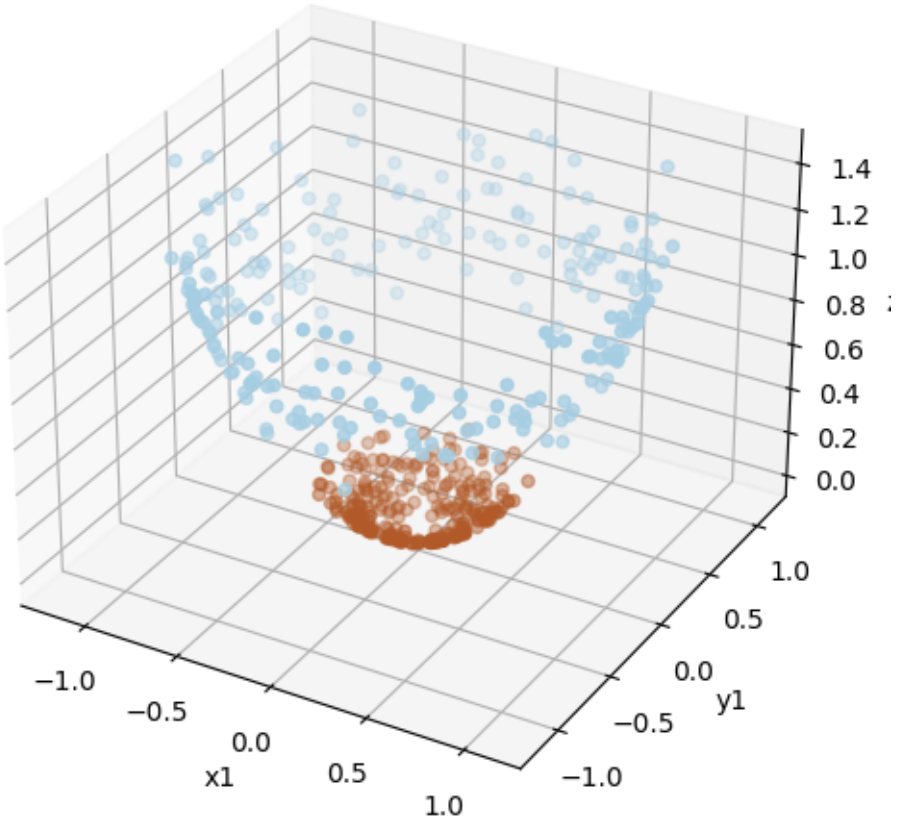

[ ]:
```
def Map_Kernel(x1,x2) :
    return x1,x2,x1**2+x2**2

mapped_x1,mapped_y1,mapped_z1 = Map_Kernel(X[:,0],X [:,1])


from mpl_toolkits.mplot3d import Axes3D


fig = plt.figure(figsize=(6,6))
ax  = fig.add_subplot(111,projection='3d')
ax.scatter(mapped_x1,mapped_y1,mapped_z1 ,c=y,cmap=plt.cm.Paired)
#ax.scatter(mapped_x1,mapped_y1,mapped_z1 ,c=y,cmap=plt.cm.coolwarm)

ax.set_xlabel('x1')
ax.set_ylabel('y1')
ax.set_zlabel('z1')
plt.show()
```




From the above plot we can see that the data is easily linearly seperable in 3D space even though it is not linearly seperable in 2D space.

**Using the new mapping function....**

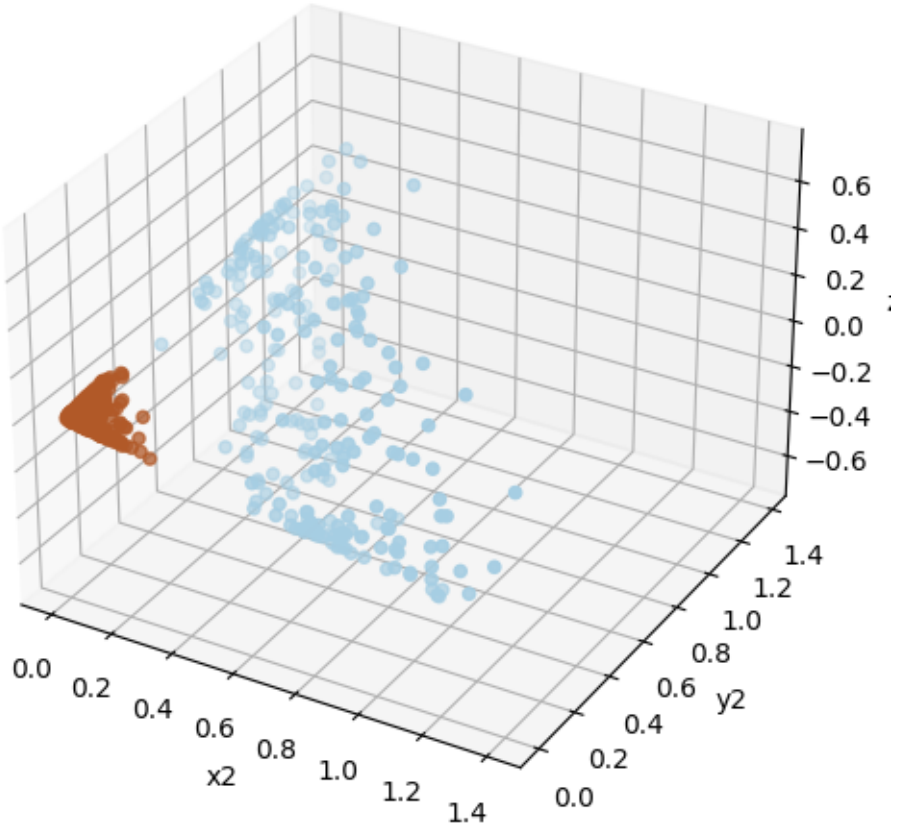
[ ]:
```
def Map_Kernel_Alt(x1,x2) :
    return x1**2,x2**2,x1*x2

mapped_x2,mapped_y2,mapped_z2 = Map_Kernel_Alt(X[:,0],X [:,1])


from mpl_toolkits.mplot3d import Axes3D


fig = plt.figure(figsize=(6,6))
ax  = fig.add_subplot(111,projection='3d')
ax.scatter(mapped_x2,mapped_y2,mapped_z2 ,c=y,cmap=plt.cm.Paired)
#ax.scatter(mapped_x1,mapped_y1,mapped_z1 ,c=y,cmap=plt.cm.coolwarm)
```

```
ax.set_xlabel('x2')
ax.set_ylabel('y2')
ax.set_zlabel('z2')
plt.show()
```




This new mapping function seems to have a more clear seperation between the two classes compared to the previous mapping function in the 3D space. Hence it seems to be a better mapping function for this dataset.

**Evaluation Metrics: Data without any mapping**

[ ]:
```
# Dataset splitting
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.
  ↪2,random_state=10)

# Create and train a linear SVM classifier
clf = svm.SVC(kernel='linear')
clf.fit(X_train,y_train)
```

```
# Predict the labels of the test set
y_pred = clf.predict(X_test)

# Evaluate the model
accuracy  = accuracy_score(y_test,y_pred)
precision = precision_score(y_test,y_pred)
recall    = recall_score(y_test,y_pred)
f1        = f1_score(y_test,y_pred)

# Print the results
print('For Dataset generated using factor=0.3')
print('Evaluation metrics without kernel mapping')
print('Accuracy   : %.2f' % accuracy)
print('Precision  : %.2f' % precision)
print('Recall     : %.2f' % recall)
print('F1 Score   : %.2f' % f1)
```

```
For Dataset generated using factor=0.3
Evaluation metrics without kernel mapping
Accuracy   : 0.64
Precision  : 0.56
Recall     : 0.98
F1 Score   : 0.71
```

**Evaluation Metrics: Data with first mapping: ( x1 , x2 , x1^2 + x2^2 )**

[ ]:
```
X_train,X_test,y_train,y_test = train_test_split(np.
  ↪c_[mapped_x1,mapped_y1,mapped_z1],y,test_size=0.2,random_state=10)

clf = svm.SVC(kernel='linear')
clf.fit(X_train,y_train)

y_pred = clf.predict(X_test)

accuracy  = accuracy_score(y_test,y_pred)
precision = precision_score(y_test,y_pred)
recall    = recall_score(y_test,y_pred)
f1        = f1_score(y_test,y_pred)

print('For Dataset generated using factor=0.3')
print('Evaluation metrics with first kernel mapping')
print('Accuracy   : %.2f' % accuracy)
print('Precision  : %.2f' % precision)
print('Recall     : %.2f' % recall)
print('F1 Score   : %.2f' % f1)
```

```
For Dataset generated using factor=0.3
Evaluation metrics with first kernel mapping
Accuracy   : 1.00
```

```
Precision  : 1.00
Recall     : 1.00
F1 Score   : 1.00
```

**Evaluation Metrics: Data with second mapping: ( x1^2 , x2^2 , x1.x2 )**

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(np.
  ↪c_[mapped_x2,mapped_y2,mapped_z2],y,test_size=0.2,random_state=10)

clf = svm.SVC(kernel='linear')
clf.fit(X_train,y_train)

y_pred = clf.predict(X_test)

accuracy  = accuracy_score(y_test,y_pred)
precision = precision_score(y_test,y_pred)
recall    = recall_score(y_test,y_pred)
f1        = f1_score(y_test,y_pred)

print('For Dataset generated using factor=0.3')
print('Evaluation metrics with second kernel mapping')
print('Accuracy   : %.2f' % accuracy)
print('Precision  : %.2f' % precision)
print('Recall     : %.2f' % recall)
print('F1 Score   : %.2f' % f1)
```

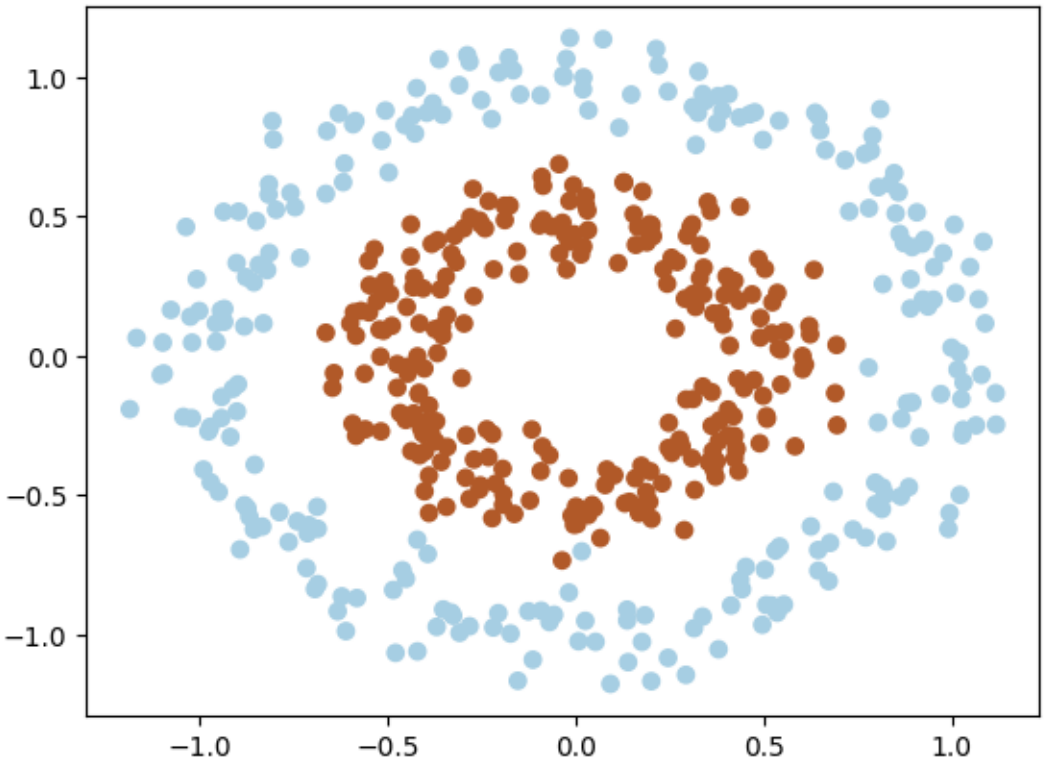
```
For Dataset generated using factor=0.3
Evaluation metrics with second kernel mapping
Accuracy   : 1.00
Precision  : 1.00
Recall     : 1.00
F1 Score   : 1.00
```

---

**Now let's repeat all of the above steps with a different dataset generated using "factor=0.5".**

[ ]:
```
# Generate data with make_circles
np.random.seed (5)
X,y = make_circles(n_samples=500,factor=0.5,noise=0.1)
```

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




In the make_circles function, the factor parameter is used to scale the inner and outer circles with respect to each other. Increasing the factor value from 0.3 to 0.5 makes the data more difficult to classify because the inner circle is scaled up and the outer circle is scaled down.

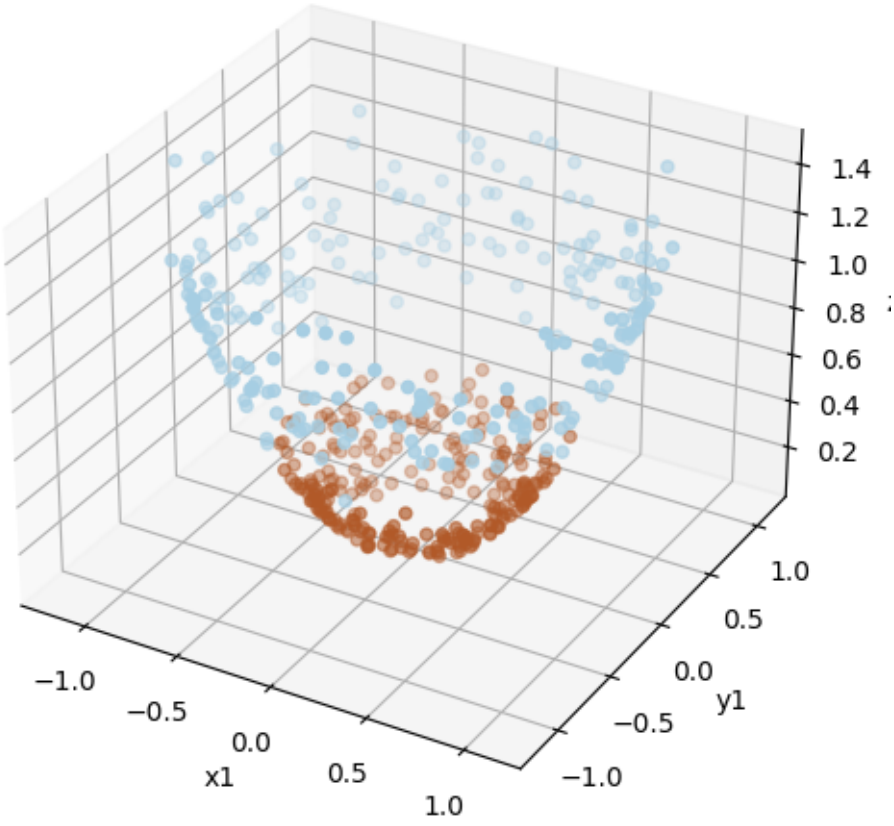
[ ]:
```
def Map_Kernel(x1,x2) :
    return x1,x2,x1**2+x2**2

mapped_x1,mapped_y1,mapped_z1 = Map_Kernel(X[:,0],X [:,1])


from mpl_toolkits.mplot3d import Axes3D


fig = plt.figure(figsize=(6,6))
ax  = fig.add_subplot(111,projection='3d')
ax.scatter(mapped_x1,mapped_y1,mapped_z1 ,c=y,cmap=plt.cm.Paired)
#ax.scatter(mapped_x1,mapped_y1,mapped_z1 ,c=y,cmap=plt.cm.coolwarm)

ax.set_xlabel('x1')
ax.set_ylabel('y1')
ax.set_zlabel('z1')
plt.show()
```




In this 3D projection, we can see that the data has become more difficult to linearly seperate in 3D space compared to the previous dataset which was generated using "factor=0.3".

**Using the new mapping function....**

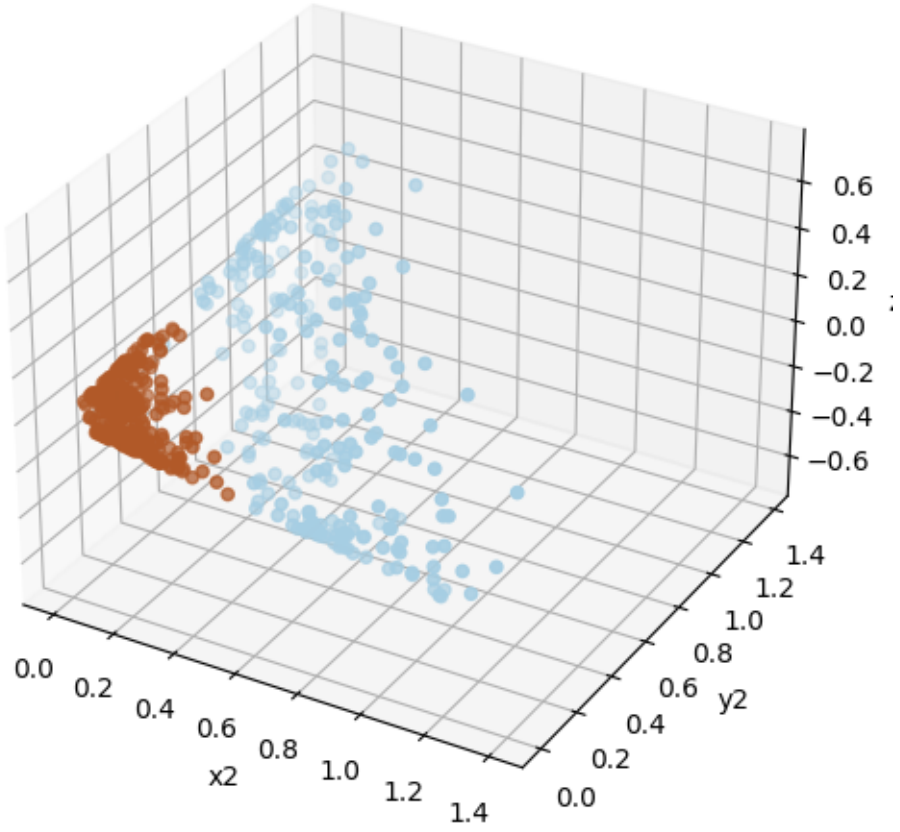
[ ]:
```
def Map_Kernel_Alt(x1,x2) :
    return x1**2,x2**2,x1*x2

mapped_x2,mapped_y2,mapped_z2 = Map_Kernel_Alt(X[:,0],X [:,1])


from mpl_toolkits.mplot3d import Axes3D


fig = plt.figure(figsize=(6,6))
ax  = fig.add_subplot(111,projection='3d')
ax.scatter(mapped_x2,mapped_y2,mapped_z2 ,c=y,cmap=plt.cm.Paired)
#ax.scatter(mapped_x1,mapped_y1,mapped_z1 ,c=y,cmap=plt.cm.coolwarm)
```

```
ax.set_xlabel('x2')
ax.set_ylabel('y2')
ax.set_zlabel('z2')
plt.show()
```




Even in this new dataset, this 2nd mapping function seems to have a more clear seperation between the two classes compared to the previous mapping function in the 3D space. Hence it seems to be a better mapping function for this dataset as well.

**Evaluation Metrics: Data without any mapping**

[ ]:
```
# Dataset splitting
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.
  ↪2,random_state=10)

# Create and train a linear SVM classifier
clf = svm.SVC(kernel='linear')
clf.fit(X_train,y_train)
```

```
# Predict the labels of the test set
y_pred = clf.predict(X_test)

# Evaluate the model
accuracy  = accuracy_score(y_test,y_pred)
precision = precision_score(y_test,y_pred)
recall    = recall_score(y_test,y_pred)
f1        = f1_score(y_test,y_pred)

# Print the result
print('For Dataset generated using factor=0.5')
print('Evaluation metrics without kernel mapping')
print('Accuracy   : %.2f' % accuracy)
print('Precision  : %.2f' % precision)
print('Recall     : %.2f' % recall)
print('F1 Score   : %.2f' % f1)
```

```
For Dataset generated using factor=0.5
Evaluation metrics without kernel mapping
Accuracy   : 0.46
Precision  : 0.46
Recall     : 1.00
F1 Score   : 0.63
```

**Evaluation Metrics: Data with first mapping: ( x1 , x2 , x1^2 + x2^2 )**

[ ]:
```
X_train,X_test,y_train,y_test = train_test_split(np.
  ↪c_[mapped_x1,mapped_y1,mapped_z1],y,test_size=0.2,random_state=10)

clf = svm.SVC(kernel='linear')
clf.fit(X_train,y_train)

y_pred = clf.predict(X_test)

accuracy  = accuracy_score(y_test,y_pred)
precision = precision_score(y_test,y_pred)
recall    = recall_score(y_test,y_pred)
f1        = f1_score(y_test,y_pred)

print('For Dataset generated using factor=0.5')
print('Evaluation metrics with first kernel mapping')
print('Accuracy   : %.2f' % accuracy)
print('Precision  : %.2f' % precision)
print('Recall     : %.2f' % recall)
print('F1 Score   : %.2f' % f1)
```

```
For Dataset generated using factor=0.5
Evaluation metrics with first kernel mapping
Accuracy   : 1.00
```

```
Precision  : 1.00
Recall     : 1.00
F1 Score   : 1.00
```

**Evaluation Metrics: Data with second mapping: ( x1^2 , x2^2 , x1.x2 )**

```
[ ]: X_train, X_test, y_train, y_test = train_test_split(np.
       ↪c_[mapped_x2,mapped_y2,mapped_z2],y,test_size=0.2,random_state=10)

     clf = svm.SVC(kernel='linear')
     clf.fit(X_train,y_train)

     y_pred = clf.predict(X_test)

     accuracy  = accuracy_score(y_test,y_pred)
     precision = precision_score(y_test,y_pred)
     recall    = recall_score(y_test,y_pred)
     f1        = f1_score(y_test,y_pred)

     print('For Dataset generated using factor=0.5')
     print('Evaluation metrics with second kernel mapping')
     print('Accuracy   : %.2f' % accuracy)
     print('Precision  : %.2f' % precision)
     print('Recall     : %.2f' % recall)
     print('F1 Score   : %.2f' % f1)
```

```
For Dataset generated using factor=0.5
Evaluation metrics with second kernel mapping
Accuracy   : 1.00
Precision  : 1.00
Recall     : 1.00
F1 Score   : 1.00
```

**Evaluation Metrics Summary**

```
For Dataset generated using factor=0.3
Evaluation metrics without kernel mapping
Accuracy   : 0.64
Precision  : 0.56
Recall     : 0.98
F1 Score   : 0.71

For Dataset generated using factor=0.3
Evaluation metrics with first kernel mapping
Accuracy   : 1.00
Precision  : 1.00
Recall     : 1.00
F1 Score   : 1.00

For Dataset generated using factor=0.3
Evaluation metrics with second kernel mapping
Accuracy   : 1.00
Precision  : 1.00
Recall     : 1.00
F1 Score   : 1.00
```

```
For Dataset generated using factor=0.5
Evaluation metrics without kernel mapping
Accuracy   : 0.46
Precision  : 0.46
Recall     : 1.00
F1 Score   : 0.63

For Dataset generated using factor=0.5
Evaluation metrics with first kernel mapping
Accuracy   : 1.00
Precision  : 1.00
Recall     : 1.00
F1 Score   : 1.00

For Dataset generated using factor=0.5
Evaluation metrics with second kernel mapping
Accuracy   : 1.00
Precision  : 1.00
Recall     : 1.00
F1 Score   : 1.00
```

- By observing the above performance metrics, we can see that by changing the factor value from 0.3 to 0.5, the performance of the model has decreased. This is because the data has become more difficult to classify.
- We can also observe that by using both of the new mapping functions, the performance of the model has significantly increased compared to the data without any mapping. This is because the data has become more linearly seperable in 3D space after applying the new mapping functions.
- Both of the mapping functions seem to have very similar performance metrics. Both of them have a 100% accuracy and a 0.0 loss. Hence both of them seem to be good mapping functions for this particular dataset.

---

**Index No : 200462U**

**Name : Randika Perera**

https://github.com/randika-perera/EN3150-Pattern-Recognition/blob/main/Assignment%204/Assignment%2004%20Kernel%20Methods.ipynb

---

***End of Assignment 04: Kernel Methods***