SVM Kernels Indepth Intuition And Practical Explanation

In [1]:

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

x = np.linspace(-5.0, 5.0, 100) # ye line space -5 se 5 k bichme 100 point create krdega
y = np.sqrt(10**2 - x**2)
```

In [2]:

```python
x
```

Out[2]:

```
array([-5.        , -4.8989899 , -4.7979798 , -4.6969697 , -4.5959596 ,
       -4.49494949, -4.39393939, -4.29292929, -4.19191919, -4.09090909,
       -3.98989899, -3.88888889, -3.78787879, -3.68686869, -3.58585859,
       -3.48484848, -3.38383838, -3.28282828, -3.18181818, -3.08080808,
       -2.97979798, -2.87878788, -2.77777778, -2.67676768, -2.57575758,
       -2.47474747, -2.37373737, -2.27272727, -2.17171717, -2.07070707,
       -1.96969697, -1.86868687, -1.76767677, -1.66666667, -1.56565657,
       -1.46464646, -1.36363636, -1.26262626, -1.16161616, -1.06060606,
       -0.95959596, -0.85858586, -0.75757576, -0.65656566, -0.55555556,
       -0.45454545, -0.35353535, -0.25252525, -0.15151515, -0.05050505,
        0.05050505,  0.15151515,  0.25252525,  0.35353535,  0.45454545,
        0.55555556,  0.65656566,  0.75757576,  0.85858586,  0.95959596,
        1.06060606,  1.16161616,  1.26262626,  1.36363636,  1.46464646,
        1.56565657,  1.66666667,  1.76767677,  1.86868687,  1.96969697,
        2.07070707,  2.17171717,  2.27272727,  2.37373737,  2.47474747,
        2.57575758,  2.67676768,  2.77777778,  2.87878788,  2.97979798,
        3.08080808,  3.18181818,  3.28282828,  3.38383838,  3.48484848,
        3.58585859,  3.68686869,  3.78787879,  3.88888889,  3.98989899,
        4.09090909,  4.19191919,  4.29292929,  4.39393939,  4.49494949,
        4.5959596 ,  4.6969697 ,  4.7979798 ,  4.8989899 ,  5.        ])
```

In [3]:
```
[x,-x]
```

Out[3]:
```
[array([-5.        , -4.8989899 , -4.7979798 , -4.6969697 , -4.5959596 ,
        -4.49494949, -4.39393939, -4.29292929, -4.19191919, -4.09090909,
        -3.98989899, -3.88888889, -3.78787879, -3.68686869, -3.58585859,
        -3.48484848, -3.38383838, -3.28282828, -3.18181818, -3.08080808,
        -2.97979798, -2.87878788, -2.77777778, -2.67676768, -2.57575758,
        -2.47474747, -2.37373737, -2.27272727, -2.17171717, -2.07070707,
        -1.96969697, -1.86868687, -1.76767677, -1.66666667, -1.56565657,
        -1.46464646, -1.36363636, -1.26262626, -1.16161616, -1.06060606,
        -0.95959596, -0.85858586, -0.75757576, -0.65656566, -0.55555556,
        -0.45454545, -0.35353535, -0.25252525, -0.15151515, -0.05050505,
         0.05050505,  0.15151515,  0.25252525,  0.35353535,  0.45454545,
         0.55555556,  0.65656566,  0.75757576,  0.85858586,  0.95959596,
         1.06060606,  1.16161616,  1.26262626,  1.36363636,  1.46464646,
         1.56565657,  1.66666667,  1.76767677,  1.86868687,  1.96969697,
         2.07070707,  2.17171717,  2.27272727,  2.37373737,  2.47474747,
         2.57575758,  2.67676768,  2.77777778,  2.87878788,  2.97979798,
         3.08080808,  3.18181818,  3.28282828,  3.38383838,  3.48484848,
         3.58585859,  3.68686869,  3.78787879,  3.88888889,  3.98989899,
         4.09090909,  4.19191919,  4.29292929,  4.39393939,  4.49494949,
         4.5959596 ,  4.6969697 ,  4.7979798 ,  4.8989899 ,  5.        ]),
 array([ 5.        ,  4.8989899 ,  4.7979798 ,  4.6969697 ,  4.5959596 ,
         4.49494949,  4.39393939,  4.29292929,  4.19191919,  4.09090909,
         3.98989899,  3.88888889,  3.78787879,  3.68686869,  3.58585859,
         3.48484848,  3.38383838,  3.28282828,  3.18181818,  3.08080808,
         2.97979798,  2.87878788,  2.77777778,  2.67676768,  2.57575758,
         2.47474747,  2.37373737,  2.27272727,  2.17171717,  2.07070707,
         1.96969697,  1.86868687,  1.76767677,  1.66666667,  1.56565657,
         1.46464646,  1.36363636,  1.26262626,  1.16161616,  1.06060606,
         0.95959596,  0.85858586,  0.75757576,  0.65656566,  0.55555556,
         0.45454545,  0.35353535,  0.25252525,  0.15151515,  0.05050505,
        -0.05050505, -0.15151515, -0.25252525, -0.35353535, -0.45454545,
        -0.55555556, -0.65656566, -0.75757576, -0.85858586, -0.95959596,
        -1.06060606, -1.16161616, -1.26262626, -1.36363636, -1.46464646,
        -1.56565657, -1.66666667, -1.76767677, -1.86868687, -1.96969697,
        -2.07070707, -2.17171717, -2.27272727, -2.37373737, -2.47474747,
        -2.57575758, -2.67676768, -2.77777778, -2.87878788, -2.97979798,
        -3.08080808, -3.18181818, -3.28282828, -3.38383838, -3.48484848,
        -3.58585859, -3.68686869, -3.78787879, -3.88888889, -3.98989899,
        -4.09090909, -4.19191919, -4.29292929, -4.39393939, -4.49494949,
        -4.5959596 , -4.6969697 , -4.7979798 , -4.8989899 , -5.        ])]
```

In [4]:

```
y=np.hstack([y,-y])
x=np.hstack([x,-x])
```

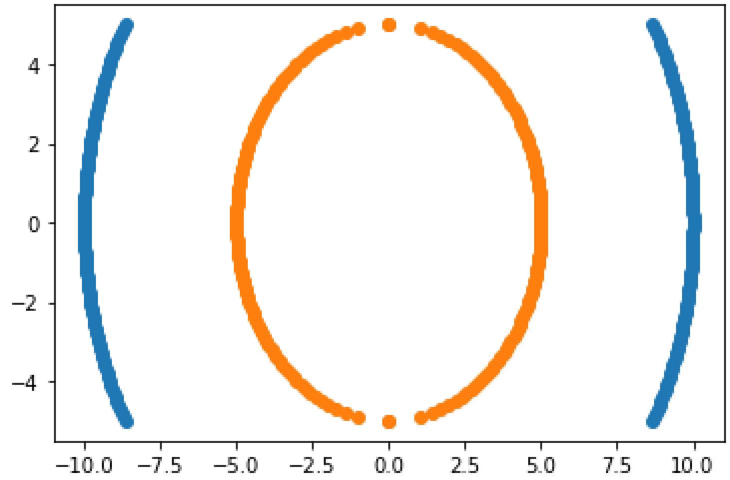
In [5]:
```
1 x
```

Out[5]:
```
array([-5.        , -4.8989899 , -4.7979798 , -4.6969697 , -4.5959596 ,
       -4.49494949, -4.39393939, -4.29292929, -4.19191919, -4.09090909,
       -3.98989899, -3.88888889, -3.78787879, -3.68686869, -3.58585859,
       -3.48484848, -3.38383838, -3.28282828, -3.18181818, -3.08080808,
       -2.97979798, -2.87878788, -2.77777778, -2.67676768, -2.57575758,
       -2.47474747, -2.37373737, -2.27272727, -2.17171717, -2.07070707,
       -1.96969697, -1.86868687, -1.76767677, -1.66666667, -1.56565657,
       -1.46464646, -1.36363636, -1.26262626, -1.16161616, -1.06060606,
       -0.95959596, -0.85858586, -0.75757576, -0.65656566, -0.55555556,
       -0.45454545, -0.35353535, -0.25252525, -0.15151515, -0.05050505,
        0.05050505,  0.15151515,  0.25252525,  0.35353535,  0.45454545,
        0.55555556,  0.65656566,  0.75757576,  0.85858586,  0.95959596,
        1.06060606,  1.16161616,  1.26262626,  1.36363636,  1.46464646,
        1.56565657,  1.66666667,  1.76767677,  1.86868687,  1.96969697,
        2.07070707,  2.17171717,  2.27272727,  2.37373737,  2.47474747,
        2.57575758,  2.67676768,  2.77777778,  2.87878788,  2.97979798,
        3.08080808,  3.18181818,  3.28282828,  3.38383838,  3.48484848,
        3.58585859,  3.68686869,  3.78787879,  3.88888889,  3.98989899,
        4.09090909,  4.19191919,  4.29292929,  4.39393939,  4.49494949,
        4.5959596 ,  4.6969697 ,  4.7979798 ,  4.8989899 ,  5.        ,
        5.        ,  4.8989899 ,  4.7979798 ,  4.6969697 ,  4.5959596 ,
        4.49494949,  4.39393939,  4.29292929,  4.19191919,  4.09090909,
        3.98989899,  3.88888889,  3.78787879,  3.68686869,  3.58585859,
        3.48484848,  3.38383838,  3.28282828,  3.18181818,  3.08080808,
        2.97979798,  2.87878788,  2.77777778,  2.67676768,  2.57575758,
        2.47474747,  2.37373737,  2.27272727,  2.17171717,  2.07070707,
        1.96969697,  1.86868687,  1.76767677,  1.66666667,  1.56565657,
        1.46464646,  1.36363636,  1.26262626,  1.16161616,  1.06060606,
        0.95959596,  0.85858586,  0.75757576,  0.65656566,  0.55555556,
        0.45454545,  0.35353535,  0.25252525,  0.15151515,  0.05050505,
       -0.05050505, -0.15151515, -0.25252525, -0.35353535, -0.45454545,
       -0.55555556, -0.65656566, -0.75757576, -0.85858586, -0.95959596,
       -1.06060606, -1.16161616, -1.26262626, -1.36363636, -1.46464646,
       -1.56565657, -1.66666667, -1.76767677, -1.86868687, -1.96969697,
       -2.07070707, -2.17171717, -2.27272727, -2.37373737, -2.47474747,
       -2.57575758, -2.67676768, -2.77777778, -2.87878788, -2.97979798,
       -3.08080808, -3.18181818, -3.28282828, -3.38383838, -3.48484848,
       -3.58585859, -3.68686869, -3.78787879, -3.88888889, -3.98989899,
       -4.09090909, -4.19191919, -4.29292929, -4.39393939, -4.49494949,
       -4.5959596 , -4.6969697 , -4.7979798 , -4.8989899 , -5.        ])
```

In [6]:
```
x1 = np.linspace(-5.0, 5.0, 100)
y1 = np.sqrt(5**2 - x1**2)
y1=np.hstack([y1,-y1])
x1=np.hstack([x1,-x1])
```

In [7]:
```
plt.scatter(y,x) #blue point
plt.scatter(y1,x1) #orange point inner one
```

Out[7]: <matplotlib.collections.PathCollection at 0x13fd9f9a550>



In [8]:

```
np.vstack([y,x]).T
```

Out[8]:

```
array([[ 8.66025404, -5.        ],
       [ 8.71779204, -4.8989899 ],
       [ 8.77378994, -4.7979798 ],
       [ 8.82827705, -4.6969697 ],
       [ 8.88128118, -4.5959596 ],
       [ 8.93282873, -4.49494949],
       [ 8.98294476, -4.39393939],
       [ 9.03165312, -4.29292929],
       [ 9.07897646, -4.19191919],
       [ 9.12493632, -4.09090909],
       [ 9.16955321, -3.98989899],
       [ 9.21284664, -3.88888889],
       [ 9.25483518, -3.78787879],
       [ 9.29553652, -3.68686869],
       [ 9.3349675 , -3.58585859],
       [ 9.37314414, -3.48484848],
       [ 9.41008171, -3.38383838],
       [ 9.44579475, -3.28282828],
       [ 9.4802971 , -3.18181818],
```

In [9]:

```
import pandas as pd
df1 =pd.DataFrame(np.vstack([y,x]).T,columns=['X1','X2'])
df1['Y']=0
df2 =pd.DataFrame(np.vstack([y1,x1]).T,columns=['X1','X2'])
df2['Y']=1
df = df1.append(df2)
df.head(5)
```

Out[9]:

	X1	X2	Y
0	8.660254	-5.00000	0
1	8.717792	-4.89899	0
2	8.773790	-4.79798	0
3	8.828277	-4.69697	0
4	8.881281	-4.59596	0

In [10]:
```
df.tail()
```

Out[10]:

	X1	X2	Y
195	-1.969049	-4.59596	1
196	-1.714198	-4.69697	1
197	-1.406908	-4.79798	1
198	-0.999949	-4.89899	1
199	-0.000000	-5.00000	1

In [11]:
```
### Independent and Dependent features
X = df.iloc[:, :2]
y = df.Y
```

In [12]:
```
y
```

Out[12]:
```
0      0
1      0
2      0
3      0
4      0
      ..
195    1
196    1
197    1
198    1
199    1
Name: Y, Length: 400, dtype: int64
```

In [13]:
```
## Split the dataset into train and test
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=0)
```

In [14]:
```
y_train
```

Out[14]:
```
50     1
63     0
112    1
159    0
83     1
      ..
123    1
192    0
117    0
47     0
172    0
Name: Y, Length: 300, dtype: int64
```

In [15]:
```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
classifier = SVC(kernel="linear")
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
accuracy_score(y_test, y_pred)
```

Out[15]: 0.45

In [16]:
```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
classifier = SVC(kernel="poly")
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
accuracy_score(y_test, y_pred)
```

Out[16]: 0.59

In [17]:
```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
classifier = SVC(kernel="rbf")
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
accuracy_score(y_test, y_pred)
```

Out[17]: 1.0

In [18]:
```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
classifier = SVC(kernel="sigmoid")
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
accuracy_score(y_test, y_pred)
```

Out[18]: 0.51

accuracy before appying polynomial kernal

rbf has more accuracy , so for this circular type of data distribution use rbf

automatically below thing will be executed by rbf

Polynomial Kernel

$$K(x, y) = (x^\mathsf{T} y + c)^d$$

In [19]:
```
# We need to find components for the Polynomical Kernel
#X1,X2,X1_square,X2_square,X1*X2
df['X1_Square']= df['X1']**2
df['X2_Square']= df['X2']**2
df['X1*X2'] = (df['X1'] *df['X2'])
df.head()
```

Out[19]:

	X1	X2	Y	X1_Square	X2_Square	X1*X2
0	8.660254	-5.00000	0	75.000000	25.000000	-43.301270
1	8.717792	-4.89899	0	75.999898	24.000102	-42.708375
2	8.773790	-4.79798	0	76.979390	23.020610	-42.096467
3	8.828277	-4.69697	0	77.938476	22.061524	-41.466150
4	8.881281	-4.59596	0	78.877155	21.122845	-40.818009

In [20]:
```
### Independent and Dependent features
X = df[['X1','X2','X1_Square','X2_Square','X1*X2']]
y = df['Y']
```

In [21]:
```
y
```

Out[21]:
```
0      0
1      0
2      0
3      0
4      0
      ..
195    1
196    1
197    1
198    1
199    1
Name: Y, Length: 400, dtype: int64
```

In [ ]:
```

```

In [22]:
```

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size = 0.25,
                                                    random_state = 0)

```

In [23]:

```
1 X_train
```

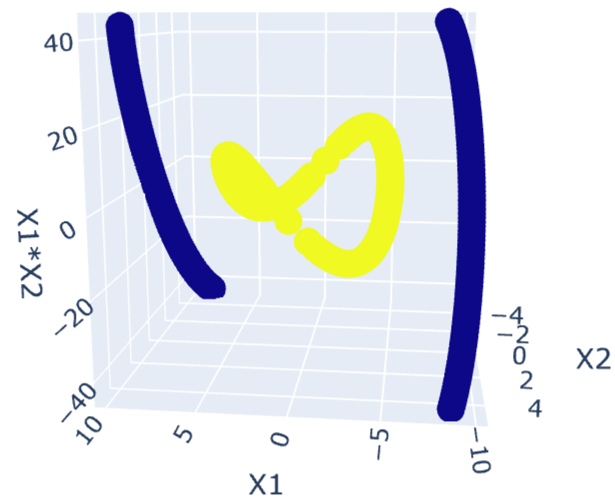
Out[23]:

	X1	X2	X1_Square	X2_Square	X1*X2
50	4.999745	0.050505	24.997449	0.002551	0.252512
63	9.906589	1.363636	98.140496	1.859504	13.508984
112	-3.263736	3.787879	10.651974	14.348026	-12.362637
159	-9.953852	-0.959596	99.079176	0.920824	9.551676
83	3.680983	3.383838	13.549638	11.450362	12.455852
...	...	...	...	...	...
123	-4.223140	2.676768	17.834915	7.165085	-11.304366
192	-9.031653	-4.292929	81.570758	18.429242	38.772248
117	-9.445795	3.282828	89.223038	10.776962	-31.008922
47	9.996811	-0.252525	99.936231	0.063769	-2.524447
172	-9.738311	-2.272727	94.834711	5.165289	22.132526

300 rows × 5 columns

In [24]:
```
import plotly.express as px

fig = px.scatter_3d(df, x='X1', y='X2', z='X1*X2',
              color='Y')
fig.show()
```

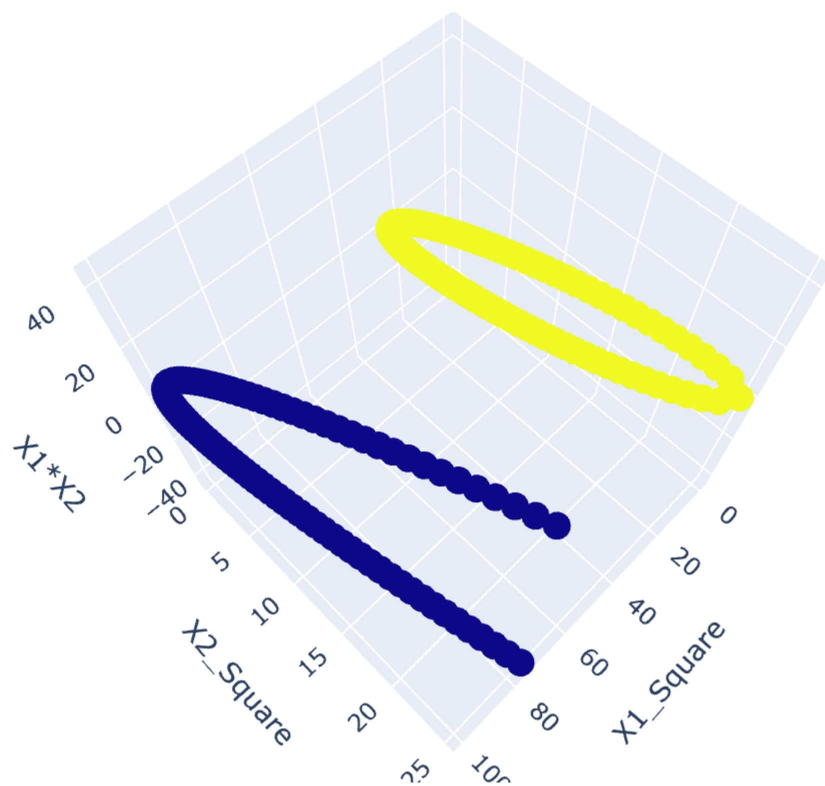


In [25]:
```
# ye upar wala graph jb apan ne normally data plot kr re hai tbka hai
# now isme dekh skte toh ye easily seprable nahi hai
# toh apan ne X1_square, X2_square nikala jissne niche wala graph bnra hai
```

In [26]:

```

fig = px.scatter_3d(df, x='X1_Square', y='X2_Square', z='X1*X2',
           color='Y')
fig.show()
```



In [27]:

```
# data segregate krdiya pura
# ab niche wali accuracy sari 1 ari hai, kyoki seprate kr skte hai
```

In [28]:
```python
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
classifier = SVC(kernel="linear")
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
accuracy_score(y_test, y_pred)
```

Out[28]: 1.0

In [29]:
```python
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
classifier = SVC(kernel="poly")
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
accuracy_score(y_test, y_pred)
```

Out[29]: 1.0

In [30]:
```python
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
classifier = SVC(kernel="rbf")
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
accuracy_score(y_test, y_pred)
```

Out[30]: 1.0

In [31]:
```python
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
classifier = SVC(kernel="sigmoid")
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
accuracy_score(y_test, y_pred)
```

Out[31]: 1.0

In [32]:
```python
# yaha se ye nishkarsh nikalta hai ki
# data ko split 3d me krdege toh accuracy nikal skti hai
# toh uske liye apan ne polynomail kernal k formule lagaye or new data point create krdiye , jisse sari accuracy 1.0 agai
# or rbf kernal jo hai vo usko automatically 3D me seprate krdega , bina polynomial lagye . jese line 17 and 30 me "rbf" apply kia hai
```