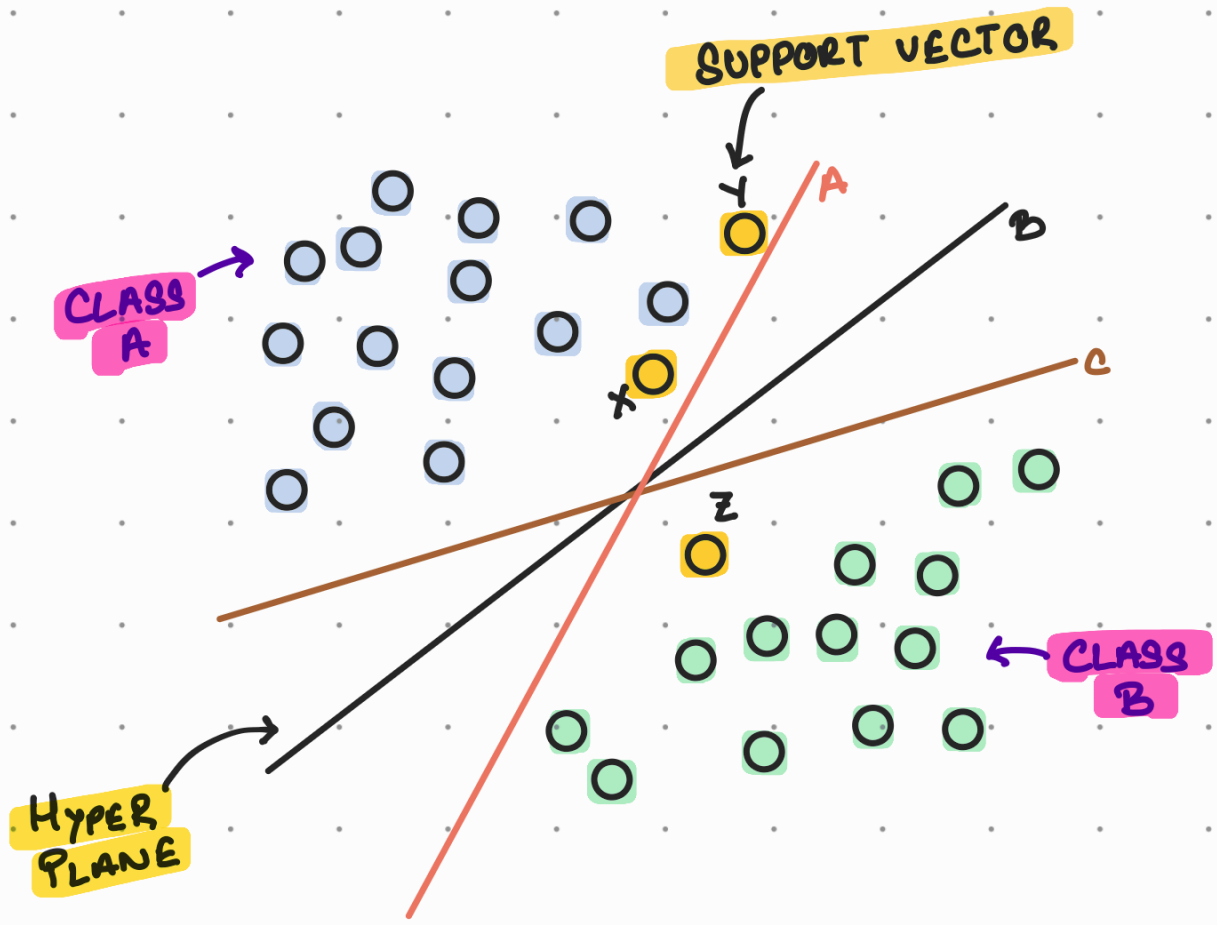
The Machine Learning Journey

Chapter-06

Support Vector Machine



So consider we have 2 classes of data points.

Now we are asked to divide these data points by drawing a straight line.

We can draw several straight lines like 'A', 'B' and 'C'.

But from the above diagram, we can say that it is line 'B'. Why? Because the line 'B' is splitting the data such that the distance between the 2 groups is maximum.

This is the hyperplane.

So let us remember that **hyper plane** is a straight line that is so drawn that there will be maximum margin between the 2 classes of data points.

When we observe the hyperplane, we can understand that some data points are much closer to this line. These are called **support vectors.**

Here X, Y and Z are the support vectors.

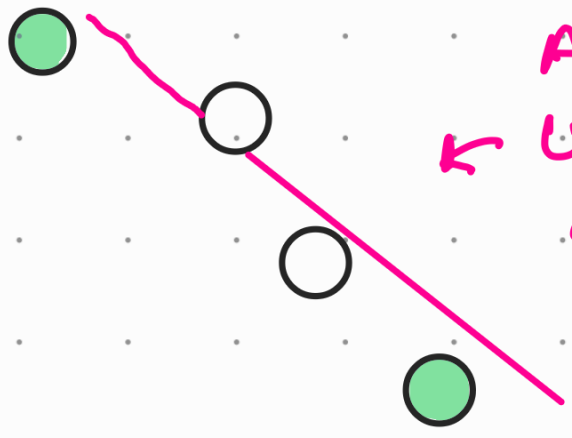
Now what happens is due to such close proximity to the hyperplane, there **arises difficulty regarding classification** of these support vectors, and the model cannot decide into which class they should put it.

So, **SVM** is a Machine Learning Model that works on the **principle of hyper plane.**

KERNEL FUNCTION

Sometimes the data points will be such that it may not be possible to divide them using a straight line.

eg-

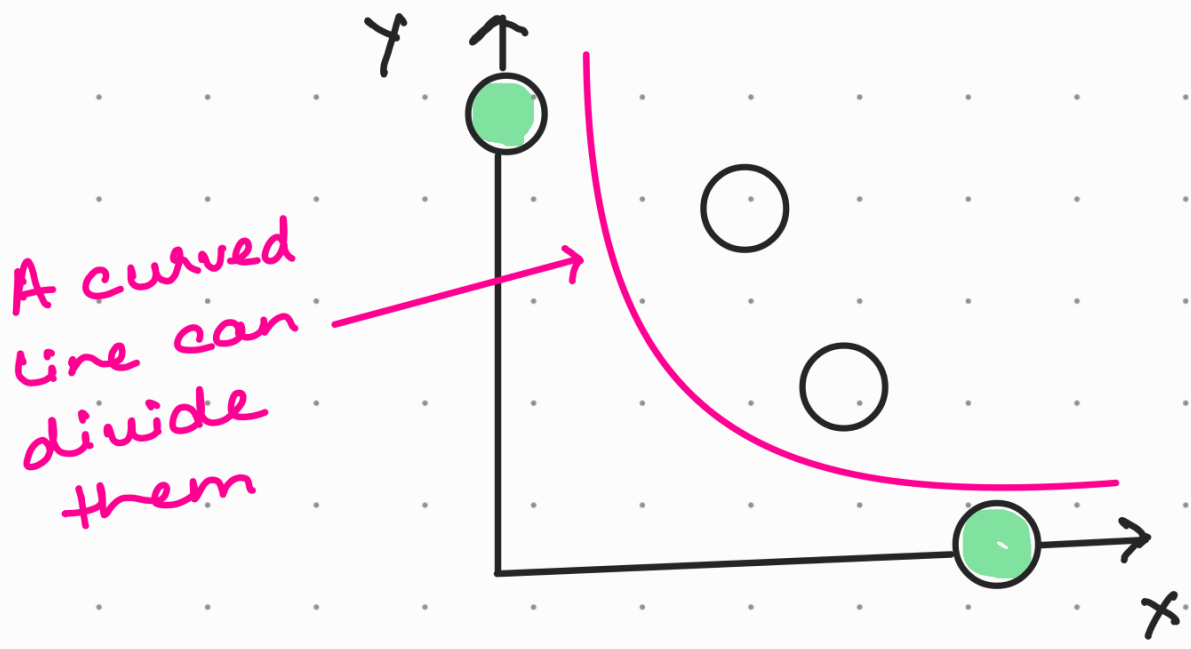


A straight line cannot divide this.

We thus need to use a curved line.

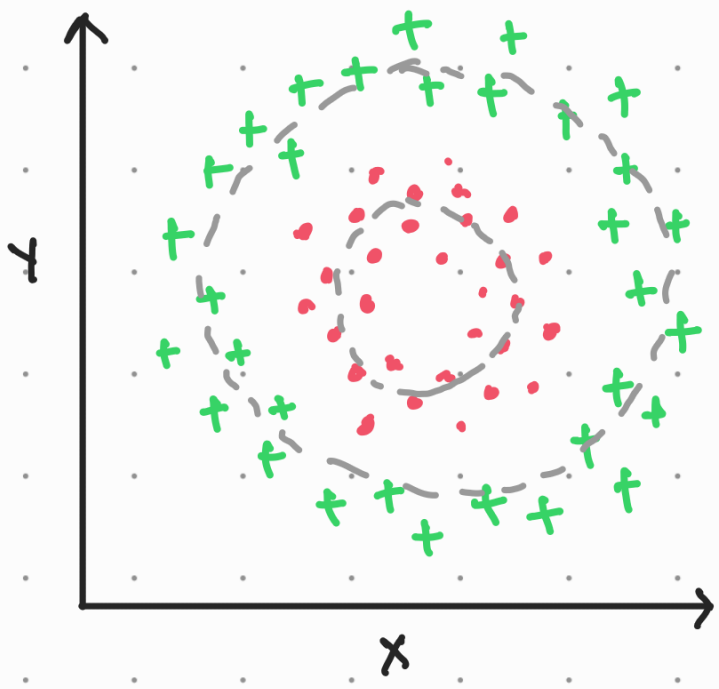
To use this curved line, we require a mathematical function that will be used by SVM and transform the data to fit by the curved line.

That mathematical function is called 'kernel function'.



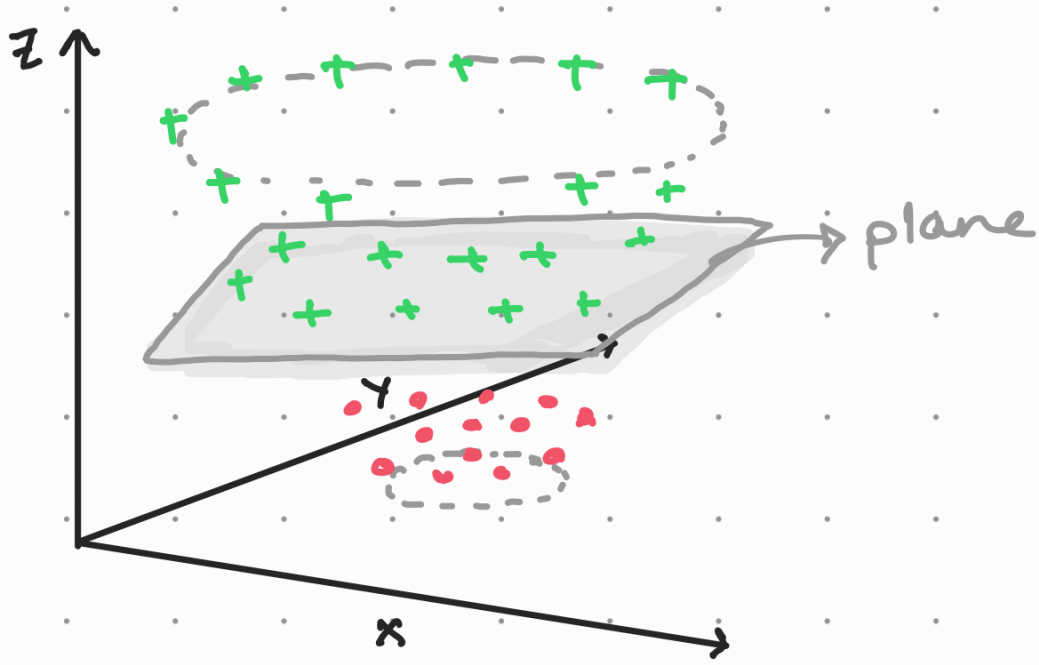
Sometimes, the data can be divided using a plane.

Let us take an example –



Here there are 2 classes of data points arranged in 2 circles. One class is represented by (●) and the other by (+)

Here we cannot use a straight line to divide these data points. In this case, we can transform the data into a 3D space. The data can be visualised in the form of a tumbler, as shown below-



Thus we see how a plane divides the data points into 2 classes. This is what SVM does using a kernel function.

Now there are various kernel functions which transforms data so that it can be divided properly by SVM.

eg – ✓ Linear kernel function,
✓ polynomial k. func.
✓ Radial basis func. (RBF)
✓ Sigmoid kernel func.

SVM model is implemented in the code like this —

```
from sklearn.svm import SVC
model = SVC()
```

the default kernel is rbf, so model = SVC() is equivalent to -

```
model = SVC(kernel = 'rbf')
```

PARAMETERS IN SVM

1. GAMMA PARAMETER →

- Indicates how far the influence of a single training example reaches.
- 'gamma' value decides whether the points close to the hyperplane should be considered to classify a data point or not.
- When gamma is low — points far

from the hyperplane are also considered in decision making.

- When gamma is high — points close to hyper plane are considered.
- Gamma can be 'auto' or 'scale'.

When gamma is 'auto', its value is - $\frac{1}{n\_features}$. If 4 input columns are there, then gamma = $\frac{1}{4} = 0.25$.

When gamma is 'scale', its value is -

$$\frac{1}{(n\_features * X.var())}$$

So, if there are 4 input columns, and variance of array is 1.75.

$$= \frac{1}{(4 \times 1.75)} = 0.1428$$

2. REGULARISATION →

- Helps to solve over-fitting in Machine Learning.
- 'Over-fitting' refers to high variance. To reduce variance, we have to add a penalty term to the formula.
- Shown as 'C' in SVM.

- When C is low, SVM uses a larger margin between data points.
- When C is high, margin is small, so accuracy will be high.
- Default C value is 1.0

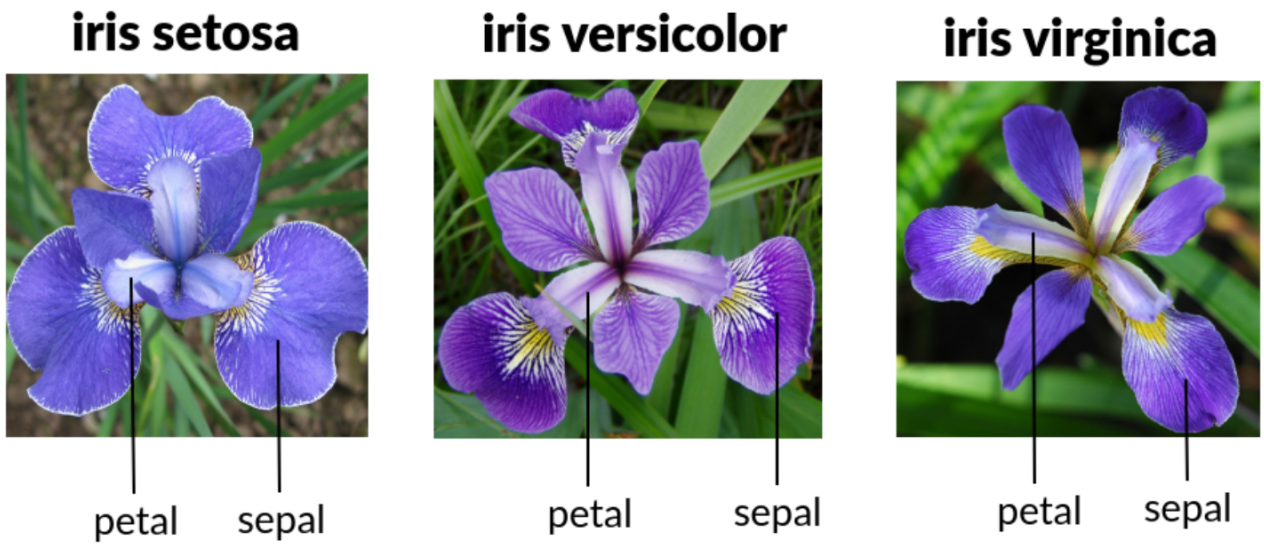
3. KERNEL FUNCTION ⟶

- A mathematical formula that is useful to rearrange the data points such that they can be divided clearly by the SVM.

(Rest has been discussed above)

✓EXERCISES :

The first exercise is on 3 varieties of 'iris' flower. Before diving in, lets see what we are dealing with —



Classify the iris flowers data set using SVM and find out the flower type depending on the given input data

In [1]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
from sklearn.datasets import load_iris
iris = load_iris()
```

In [3]:
```
#display info about the dataset
dir(iris)
```

Out[3]:
```
['DESCR',
 'data',
 'data_module',
 'feature_names',
 'filename',
 'frame',
 'target',
 'target_names']
```

In [4]:
```
iris.data[:10] #display data
```

← showing the first 10 arrays out of 150

Out[4]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1]])
```

In [5]:
```
len(iris.data) #no of arrays indicate the number of rows here.
```

Out[5]:
```
150
```

In [6]:
```
iris.target #0-satosa, 1- versicolor, 2-virginica
```

Out[6]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

In [7]:
```
iris.target_names
```

Out[7]:
```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

In [8]:
```
iris.feature_names
```

Out[8]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

← notice the sepal/petal parameters

In [9]:
```
#convert iris data set into a dataframe
import pandas as pd
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df
```

Out[9]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

In [10]:
```
#add target column to df
df['target'] = iris.target
df
```

Out[10]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

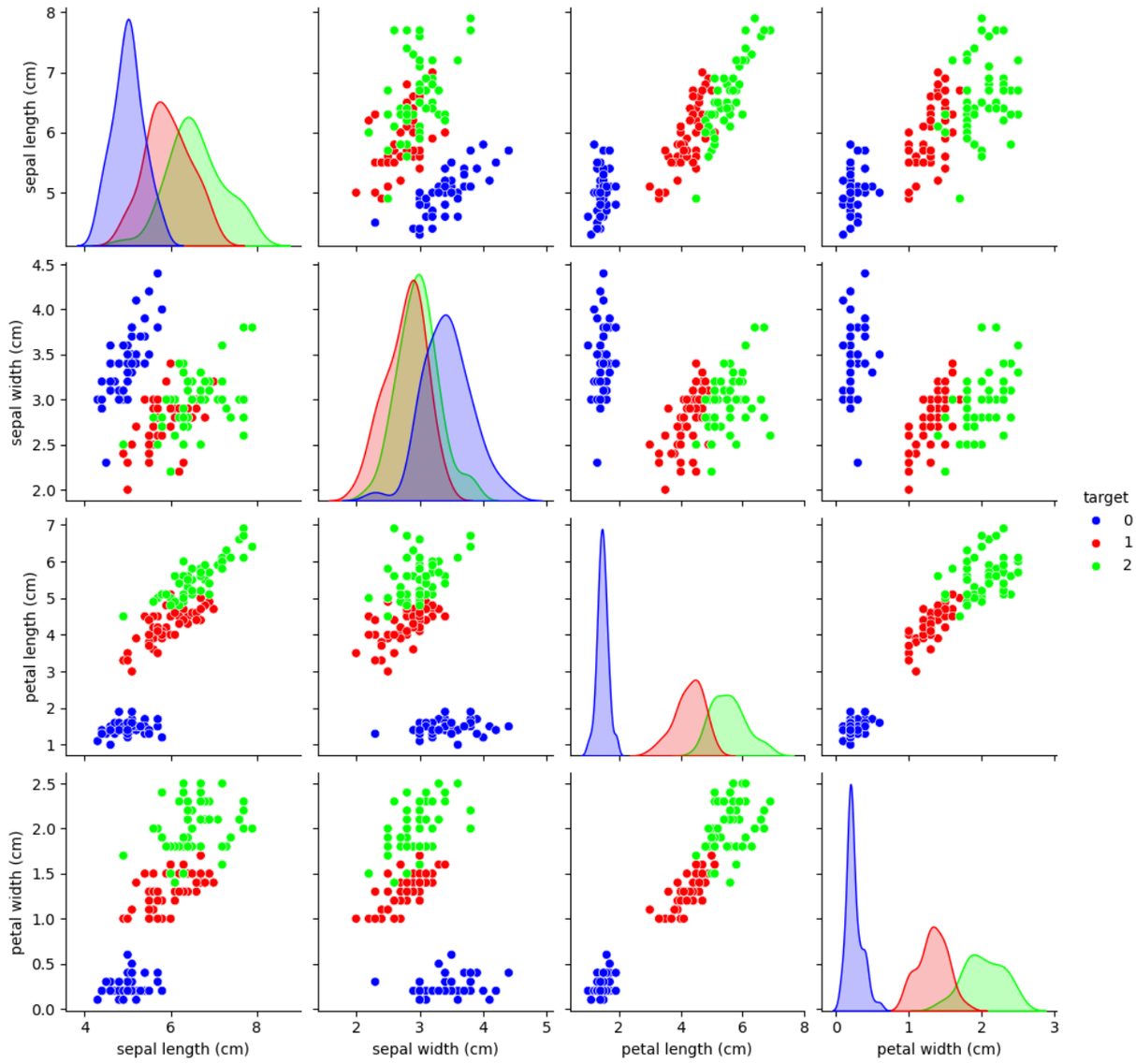
150 rows × 5 columns

In [11]:
```
#to display pair plots
import matplotlib.pyplot as plt
import seaborn as sns
```

Here below, `hue='target'` represents that the colour of the points should be changed depending on the 'target'.

So the 3 flowers are represented using 3 different colors and the colors are selected from the palette `"brg"` that represents blue, red and green colors

In [12]:
```
sns.pairplot(df, hue='target', palette="brg")
plt.show()
```



When we observe the 3 different colored ~~dat a~~ data points, we can understand that they are well separated even though some points are overlapped. Hence it is possible to divide them using a classification Machine Learning Model like SVM.

In [13]:
```
#x represents all columns except the target column
x = df.drop(['target'], axis = 'columns')
x
```

Out[13]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

In [14]:
```
#y represents the target column
y = df.target
y
```

Out[14]:
```
0      0
1      0
2      0
3      0
4      0
      ..
145    2
146    2
147    2
148    2
149    2
Name: target, Length: 150, dtype: int64
```

In [15]:
```
#let us split the data
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.3)
```

In [16]:
```
#train the svm model
from sklearn.svm import SVC
model = SVC()
model.fit(x_train, y_train)
```

Out[16]:
```
▾ SVC

SVC()
```

Since we are not passing any parameters to SVC class object, **it uses default values.** The default value of 'C' is 1.0, kernel='rbf' and gamma is 'scale'

In [17]:
```
#find accuracy level
model.score(x_test, y_test)
```

Out[17]:
```
0.9777777777777777
```
← 97.7%

In [18]:
```
#take a random sample for prediction in the data, 0-49 = setosa, 50-99 for versicolo
model.predict([iris.data[50]])
```

Out[18]:
```
array([1])
```
→ 1 → versicolour. (hence correct)

In [19]:
```
#let us give our values for sepal length, width and petal length, width
model.predict([[7.7,2.6,6.9,2.3]])
```

Out[19]:
```
array([2])
```
→ 2 → virginica

Read data from hand written digits data set. Apply SVM on it and analyse data.

Observe the accuracy of the model with different kernels. Then predict the digits from the images of the hadnwritten digits.

In [1]:
```
#handwritten digits classification using svm
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
digits = load_digits()
```

In [2]:
```
#display the attributes of the digts object
dir(digits)
```

Out[2]:
```
['DESCR', 'data', 'feature_names', 'frame', 'images', 'target', 'target_names']
```

In [3]:
```
#display the data of image zero - showsa an array
digits.data[0]
```

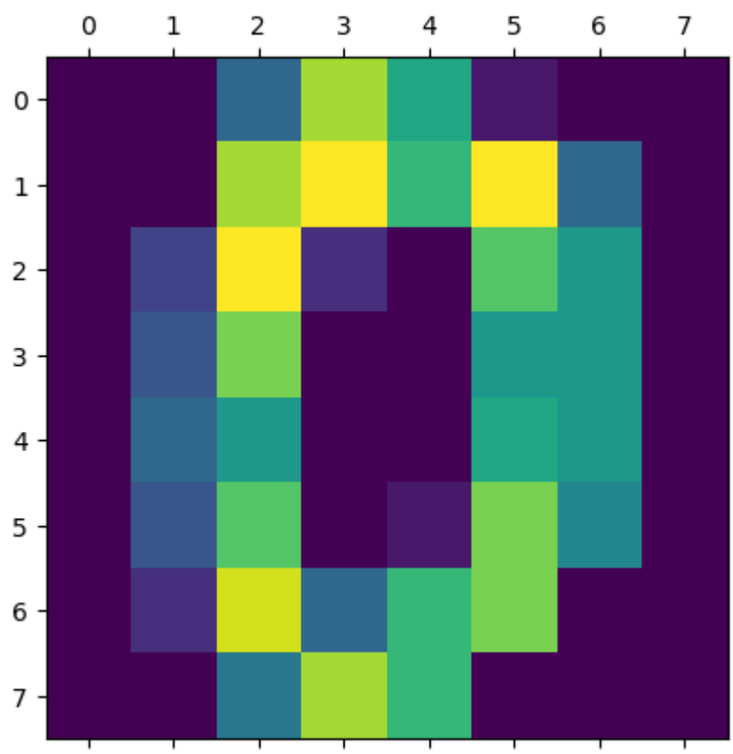
Out[3]:
```
array([ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.,  0.,  0., 13., 15., 10.,
       15.,  5.,  0.,  0.,  3., 15.,  2.,  0., 11.,  8.,  0.,  0.,  4.,
       12.,  0.,  0.,  8.,  8.,  0.,  0.,  5.,  8.,  0.,  0.,  9.,  8.,
        0.,  0.,  4., 11.,  0.,  1., 12.,  7.,  0.,  0.,  2., 14.,  5.,
       10., 12.,  0.,  0.,  0.,  0.,  6., 13., 10.,  0.,  0.,  0.])
```

Since every image is stored in the form of 8x8 pixels = 64 pixels, each pixel value is available in the array shown above. This is the array of the image 0.

Let us now diplay image using matsow() function as -

In [4]:
```
#display the image zero
plt.matshow(digits.images[0])
```

Out[4]:
```
<matplotlib.image.AxesImage at 0x7f9a50667d10>
```



In [5]:
```
#display target or original digir - shows 0
digits.target[0]
```

Out[5]:
```
0
```

In [6]:
```
#let us split the digts.data into train and test data
#take the digtits.data as independent avrs and digts.target as target var
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(digits.data, digits.target, test
```

Let us create an object to SVC (Support Vector Classifier) class and train the data :

In [7]:
```
#create SVC class object and train it
from sklearn.svm import SVC
```

In [8]:
```
#C=1.0, kernel ='rbf', gamma ='scale' ← default values
model = SVC()
model.fit(x_train, y_train)
model.score(x_test, y_test)
```

Out[8]: 0.9944444444444445

In [9]:
```
#C=1.0, kernel ='linear', gamma ='scale'
model = SVC(kernel='linear')
model.fit(x_train, y_train)
model.score(x_test, y_test)
```

Out[9]: 0.9722222222222222

In [10]:
```
#C=1.0, kernel ='sigmoid', gamma ='scale'
model = SVC(kernel='sigmoid')
model.fit(x_train, y_train)
model.score(x_test, y_test)
```

Out[10]: 0.9222222222222223

In [11]:
```
#C=1.0, kernel ='poly', gamma ='scale'
model = SVC(kernel='poly')
model.fit(x_train, y_train)
model.score(x_test, y_test)
```

Out[11]: 0.9916666666666667

In [12]:
```
#C=1.0, kernel ='poly', gamma ='auto'
model = SVC(kernel='poly', gamma ='auto')
model.fit(x_train, y_train)
model.score(x_test, y_test)
```

Out[12]: 0.9916666666666667

In [13]:
```
#take a sample test from test data - take 100th row
x_test[100]
```

Out[13]:
```
array([ 0.,  0.,  9., 15., 14.,  2.,  0.,  0.,  0.,  0.,  9.,  3.,  9.,
        8.,  0.,  0.,  0.,  0.,  0.,  0.,  6., 10.,  0.,  0.,  0.,  0.,
        0., 10., 15.,  2.,  0.,  0.,  0.,  0.,  2., 10., 11., 15.,  2.,
        0.,  0.,  3.,  1.,  0.,  0., 14.,  4.,  0.,  0., 10., 13.,  7.,
        2., 12.,  4.,  0.,  0.,  0.,  7., 14., 16., 10.,  0.,  0.])
```

In [14]:
```
#predict which digit it is
model.predict([x_test[100]])
```

Out[14]: array([3])

In [15]:
```
#compare it with actual digit - it is
y_test[100]
```

Out[15]: 3

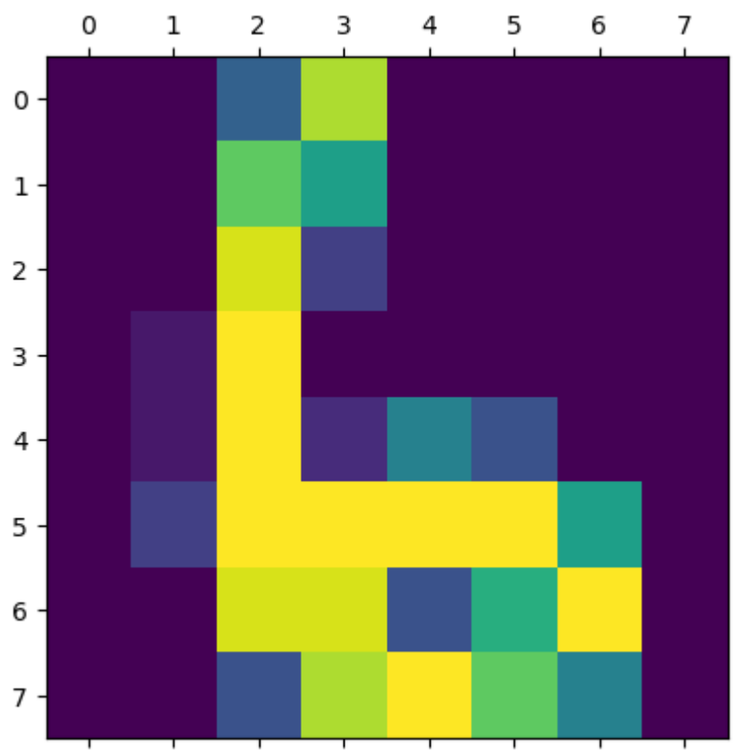
Another way of predicting the digits directly from the datasets, let us take the 67th row in the digits object

In [16]:
```
digits.data[67]
```

Out[16]:
```
array([ 0.,  0.,  5., 14.,  0.,  0.,  0.,  0.,  0.,  0., 12.,  9.,  0.,
        0.,  0.,  0.,  0.,  0., 15.,  3.,  0.,  0.,  0.,  0.,  0.,  1.,
       16.,  0.,  0.,  0.,  0.,  0.,  0.,  1., 16.,  2.,  7.,  4.,  0.,
        0.,  0.,  3., 16., 16., 16., 16.,  9.,  0.,  0.,  0., 15., 15.,
        4., 10., 16.,  0.,  0.,  0.,  4., 14., 16., 12.,  7.,  0.])
```

In [17]:
```
#this is the 67th image, looks like 6
plt.matshow(digits.images[67])
```

Out[17]:
```
<matplotlib.image.AxesImage at 0x7f9a4725fd10>
```



In [18]:
```
#let us predict which image this is
model.predict([digits.data[67]])
```

Out[18]:
```
array([6])
```

In [19]:
```
#compare it with the actual target
digits.target[67]
```

Out[19]:
```
6
```

As the predict & target are equal, hence the prediction is correct.

~ Sheryans