LINEAR ALGEBRA LAB MANUAL

SYBSc Computer Science (III Semester) For Academic Year (2023-24)

JNAN VIKAS MANDAL'S

PADMASHREE DR. R.T.DOSHI DEGREE COLLEGE OF INFORMATION TECHNOLOGY

MOHANLAL RAICHAND MEHTA COLLEGE OF COMMERCE

DIWALIMAA DEGREE COLLEGE OF SCIENCE

CERTIFICATE

This is to certify that the Mr./Miss. _____ having Roll No _____ of B.Sc. CS Semester III has completed the practical work in the subject of **Linear Algebra** under the guidance of **Ms. Shakuntala Kulkarni** during the Academic year 2023-24 being the partial requirement for the fulfillment of the curriculum of Degree of Bachelor of Science in Computer Science, University of Mumbai.

Place: **Date:**

---------- ----------
-

Sign of Subject Incharge Sign of External Examiner

Sign of Incharge / H.O.D

INDEX

Sr. no	Name of Practical	Date	Signature
1	Write a program which demonstrates the following: • Addition of two complex numbers • Displaying the conjugate of a complex number • Plotting a set of complex numbers • Creating a new plot by rotating the given number by a degree 90, 180, 270 degrees and also by scaling by a number a=1/2, a=1/3, a=2 etc.	13/07/2023	
2	Write a program to do the following: • Enter a vector u as a n-list • Enter another vector v as a n-list • Find the vector au+bv for different values of a and b • Find the dot product of u and v	20/07/2023	
3	Write a program to do the following: • Basic Matrix Operations: Matrix Addition, Subtraction, Multiplication • Check if matrix is invertible, if yes find Inverse	27/07/2023	
4	Write a program to enter a vector b and find the projection of b orthogonal to a given vector u.	03/08/2023	
5	Write a program to calculate eigenvalue and eigenvector.	10/08/2023	
6	Write a program to convert a matrix into its row echelon form.	24/08/2023	

	Write a program to find rank of a matrix.		
7	Vector Applications: Classify given data using support vector machine (SVM)	31/08/2023	
8	Implement Google's Page Rank Algorithm	07/09/2023	

PRACTICAL – 1

Write a program which demonstrates the following:

- **Addition of two complex numbers**
- **Displaying the conjugate of a complex number**
- **Plotting a set of complex numbers**
- **Creating a new plot by rotating the given number by a degree 90, 180, 270 degrees and also by scaling by a number a=1/2, a=1/3, a=2 etc.**

Solution

Addition of two complex numbers:

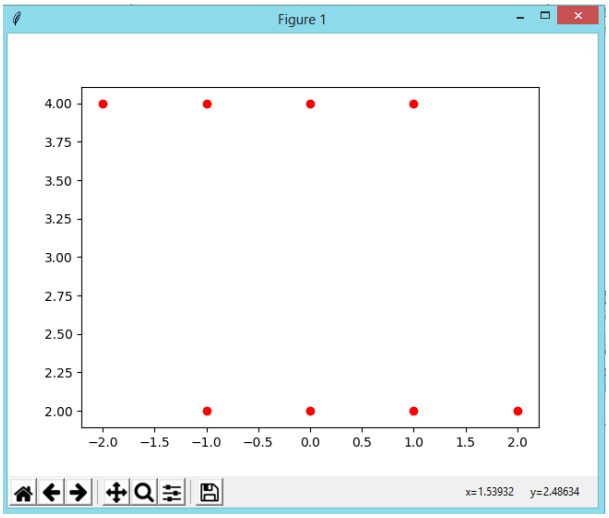
```
>>> a=3+4j
>>> b=7+5j
>>> print ("addition of two complex numbers is", a+b)
addition of two complex numbers is (10+9j)
>>>
```

Displaying the conjugate of a complex number

```
>>> a=4+5j
>>> a.conjugate()
(4-5j)
>>>
```

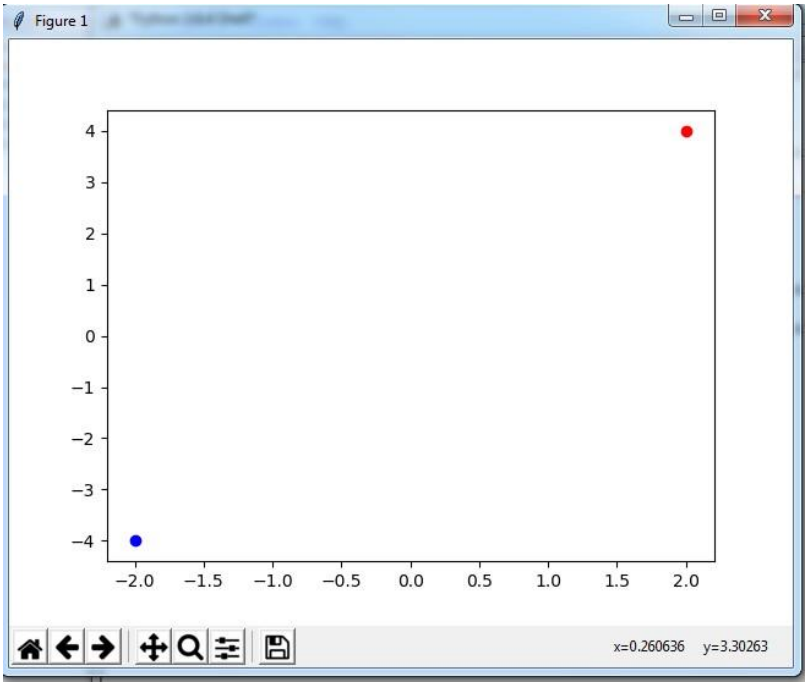
Plotting a set of complex numbers

```
import matplotlib.pyplot as plt
x = 2+2j
     a = [-2+4j, -1+2j, 0+2j, 1+2j, 2+2j, -1+4j, 0+4j, 1+4j]
     X = [x.real for x in a]
     Y = [x.imag for x in a]
     plt.scatter (X, Y, color = 'red')
 plt.show()
```



Rotation by 180^0

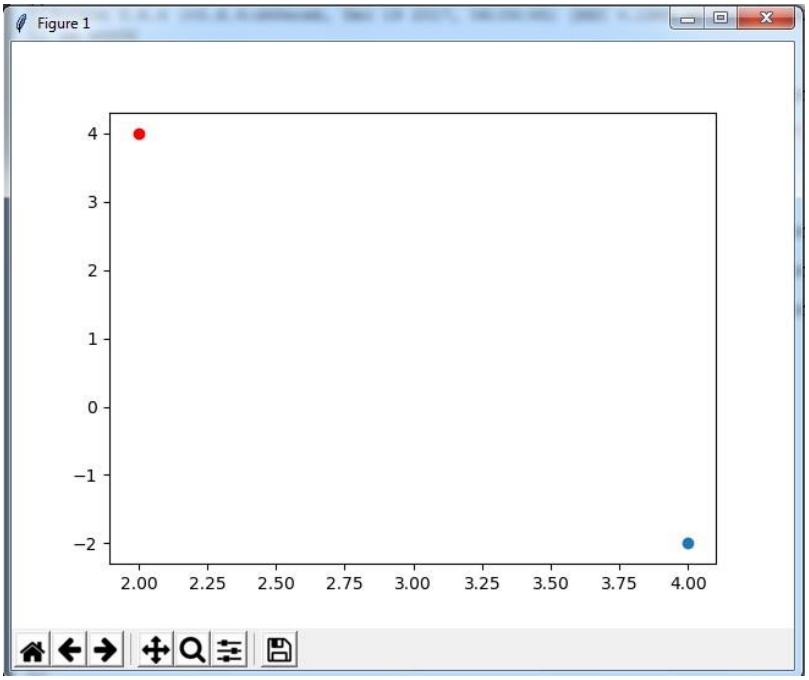
```
import matplotlib
import matplotlib.pyplot as plt
x=2+4j
plt.scatter (x.real, x.imag, color='red')
plt.scatter(-1*x.real, -1*x.imag, color='blue')
plt.show()
```



Rotation by 270^0

```
import matplotlib
import matplotlib.pyplot as plt
```

```
x=2+4j
z=-1j
plt.scatter (x.real, x.imag, color='red')
c=x*z
plt.scatter(c.real,c.imag)
plt.show()
```



scaling by a number a=1/2, a=1/3, a=2

```
import matplotlib
import matplotlib.pyplot as plt
x=2+4j
scale=0.5
scale1=0.33
scale2=2
plt.scatter (x.real, x.imag, color='red')
c=scale*x
d=scale1*x
e=scale2*x
plt.scatter(c.real,c.imag,color='green')
plt.scatter(d.real,d.imag,color='blue')
plt.scatter(e.real,e.imag,color='black')
plt.show()
```

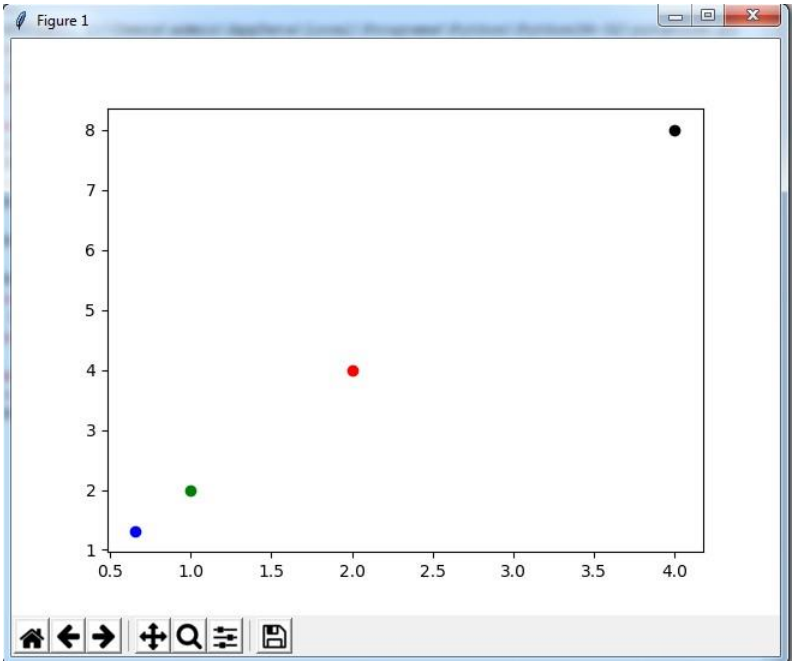


Figure 1
8
7
6
5
4
3
2
1
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

PRACTICAL – 2

Write a program to do the following:

- **Enter a vector u as a n-list**
- **Enter another vector v as a n-list**
- **Find the vector au+bv for different values of a and b**
- **Find the dot product of u and v**

```
import numpy as np
u=np.array((3,4,5))
v=np.array((1,2,7))
print ("vector u", u)
print ("vector v", v)
a=2
b=3
d=a*u
e=b*v
f= d+e
p= np.dot(u,v)
print("vector au+bv",f)
print ("dot product of u and v", p)
```

Output:-

```
vector u [3 4 5]
vector v [1 2 7]
vector au+bv [ 9 14 31]
dot product of u and v 46
```

PRACTICAL – 3

Write a program to do the following:

- **Basic Matrix Operations:**
  Matrix Addition, Subtraction, Multiplication
- **Check if matrix is invertible, if yes find Inverse**

Matrix Addition

```
X = [[12,7,3],
    [4 ,5,6],
    [7 ,8,9]]

Y = [[5,8,1],
    [6,7,3],
    [4,5,9]]

result = [[0,0,0],
          [0,0,0],
          [0,0,0]]

# iterate through rows
for i in range(len(X)):
   # iterate through columns
   for j in range(len(X[0])):
       result[i][j] = X[i][j] + Y[i][j]

for r in result:
   print(r)
```

Output:-

```
[17, 15, 4]

[10, 12, 9]

[11, 13, 18]
```

Matrix Subtraction

```
import numpy as np

# creating first matrix

A = np.array([[1, 2], [3, 4]])

# creating second matrix

B = np.array([[4, 5], [6, 7]])

print("Printing elements of first matrix")
```

```
print(A)

print("Printing elements of second matrix")

print(B)


# subtracting two matrix

print("Subtraction of two matrix")

print(np.subtract(A, B))
```

Output:

```
Printing elements of first matrix

[[1 2]

 [3 4]]

Printing elements of second matrix

[[4 5]

 [6 7]]

Subtraction of two matrix

[[-3 -3]

 [-3 -3]]
```

Write a program to find the matrix-matrix product of M with a c by p matrix N.

```
X = [[34,1,77],
 [2,14,8],
 [3 ,17,11]]

Y = [[6,8,1],
 [9,27,5],
 [2,43,31]]

result = [[0,0,0],
 [0,0,0],
 [0,0,0]]

for i in range(len(X)):
   for j in range(len(Y[0])):
     for k in range(len(Y)):
         result[i][j] += X[i][k] * Y[k][j]
```

```
for r in result:
 print(r)
```

Output:-

```
[367, 3610, 2426]
[154, 738, 320]
[193, 956, 429]
```

Write a program to enter a matrix and check if it is invertible. If the inverse exists, find the inverse.

```
import numpy as np
M = np.array([[1,2,1], [2,1,0], [3,0,2]])
print ("The Matrix M is:", M)
a=np.linalg.det(M)
print ("Determinant of matrix M is", a)
if a<=0:
    Minv = np.linalg.inv(M)
    print("The inverse of matrix M is:", Minv)
else:
    print ("Matrix is not invertible")
```

Output:-

```
The Matrix M is: [[1 2 1]
 [2 1 0]
 [3 0 2]]
Determinant of matrix M is -8.999999999999998
The inverse of matrix M is: [[-0.22222222  0.44444444
0.11111111]
 [ 0.44444444  0.11111111 -0.22222222]
 [ 0.33333333 -0.66666667  0.33333333]]
```

PRACTICAL – 4

Write a program to enter a vector b and find the projection of b orthogonal to a given vector u.

```
import numpy as np
def oprojection (of_vec, on_vec):
    v1= np.array(of_vec)
    v2 = np.array(on_vec)
    scal = np.dot(v2,v1)/np.dot(v2,v2)
    vec = scal*v2
    return round (scal, 10), np.around (vec, decimals=10)
print (oprojection ([4.0, 4.0], [1.0,0.0]))
print (oprojection ([4.0, 4.0], [8.0,2.0]))
```

Output:-

```
(4.0, array([4., 0.]))
(0.5882352941, array([4.70588235, 1.17647059]))
```

PRACTICAL – 5

Write a program to calculate eigenvalue and eigenvector.

```
import numpy as np
A=np.mat("3 -2; 1 0")
print ("A\n", A)
print ("Eigenvalues", np.linalg.eigvals(A))
eigenvalues, eigenvectors = np.linalg.eig(A)
print ("First tuple of eig", eigenvalues)
for i in range (len(eigenvalues)):
    print("Left", np.dot(A, eigenvectors[:,i]))
    print(...)
```

Output:-

```
A
 [[ 3 -2]
 [ 1  0]]
Eigenvalues [2. 1.]
First tuple of eig [2. 1.]
Left [[1.78885438]
 [0.89442719]]
Ellipsis
Left [[0.70710678]
 [0.70710678]]
Ellipsis
```

PRACTICAL – 6

Write a program to convert a matrix into its row echelon form.

Write a program to find rank of a matrix.

```
# import sympy
import sympy

# find the reduced row echelon form
sympy.Matrix([[4,0,1],[2,0,2],[3,0,3]]).rref()

# find the rank of matrix
print("Rank of matrix
:",sympy.Matrix([[4,0,1],[2,0,2],[3,0,3]]).rank())
```

Output:-

```
(Matrix([
 [1, 0, 0],
 [0, 0, 1],
 [0, 0, 0]]), (0, 2))
  Rank of matrix : 2
```

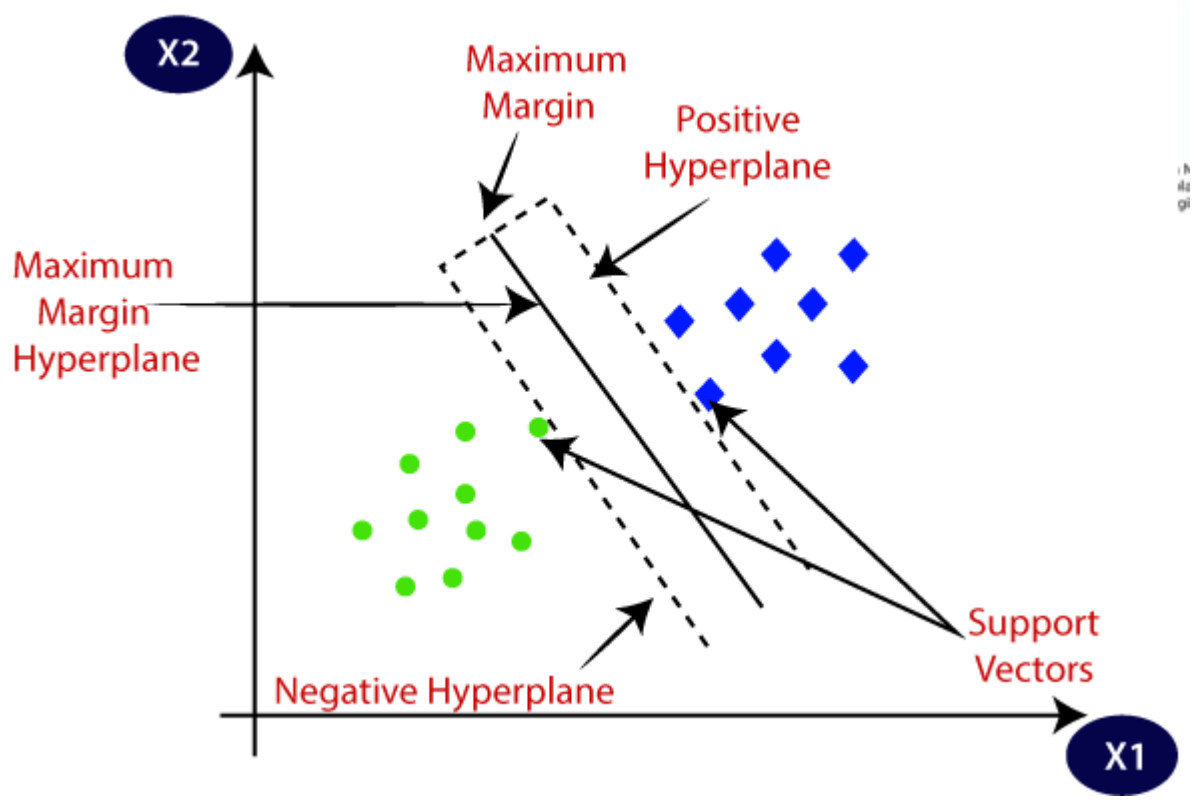
Practical – 7

Support Vector Machine

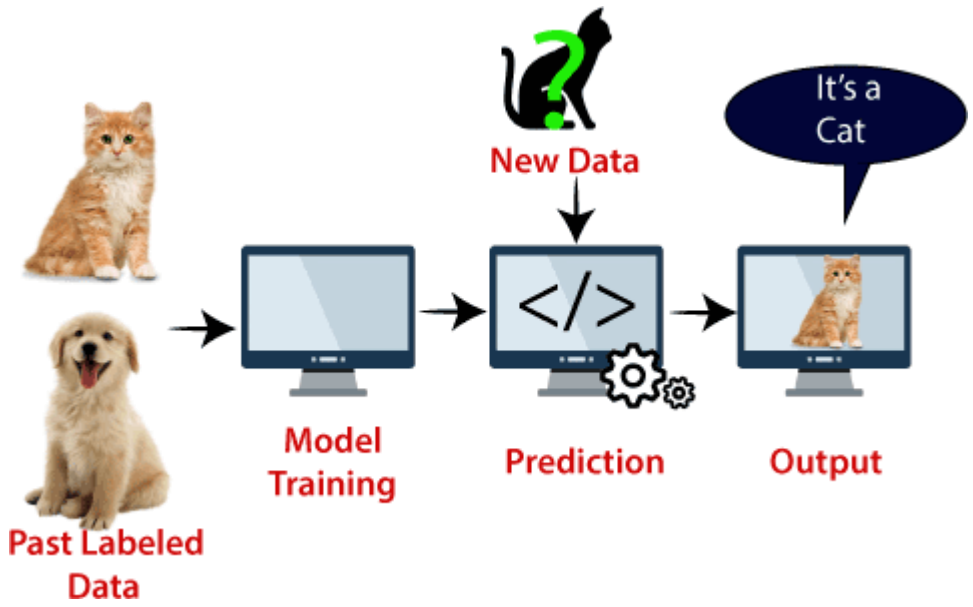
Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:



Example: SVM can be understood with the example that we have used in the KNN classifier. Suppose we see a strange cat that also has some features of dogs, so if we want a model that can accurately identify whether it is a cat or dog, so such a model can be created by using the SVM algorithm. We will first train our model with lots of images of cats and dogs so that it can learn about different features of cats and dogs, and then we test it with this strange creature. So as support vector creates a decision boundary between these two data (cat and dog) and choose extreme cases (support vectors), it will see the extreme case of cat and dog. On the basis of the support vectors, it will classify it as a cat. Consider the below diagram:



SVM algorithm can be used for **Face detection, image classification, text categorization,** etc.

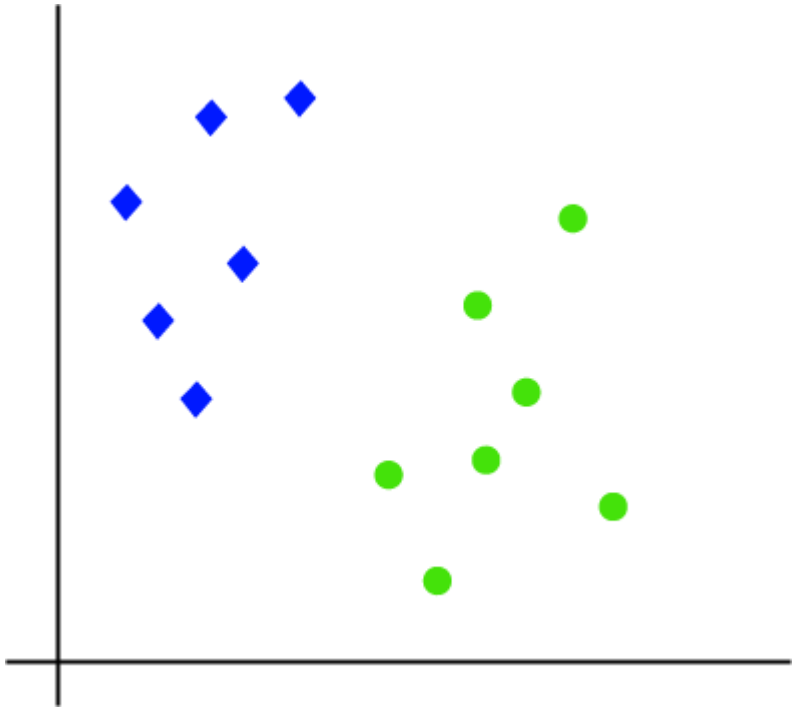
Types of SVM

SVM can be of two types:

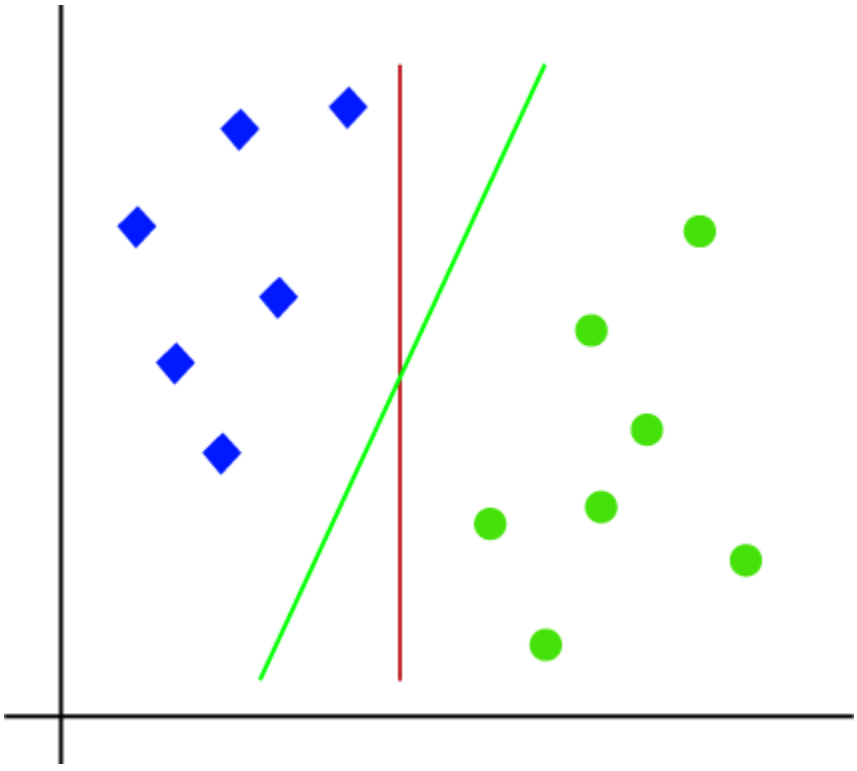
- **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
- **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

How does SVM works?

The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. Consider the below image:

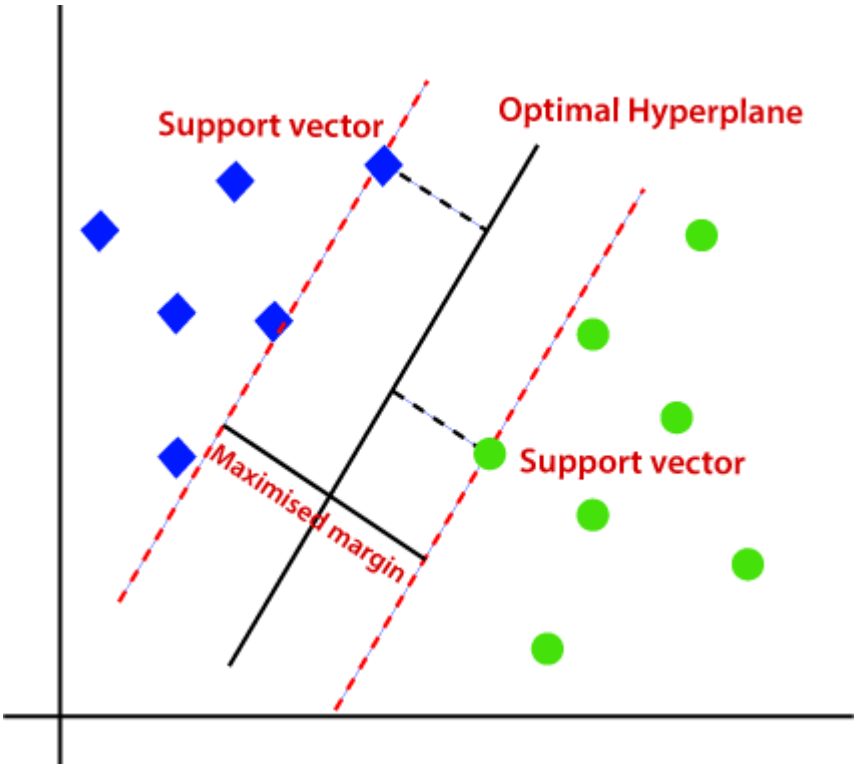


So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes. Consider the below image:



Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a **hyperplane**. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and

the hyperplane is called as **margin**. And the goal of SVM is to maximize this margin. The **hyperplane** with maximum margin is called the **optimal hyperplane**.



Practical – 8

Google's Page Rank Algorithm

PageRank (PR) is an algorithm used by Google Search to rank websites in their search engine results. PageRank was named after Larry Page, one of the founders of Google. PageRank is a way of measuring the importance of website pages. According to Google:

PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites.

It is not the only algorithm used by Google to order search engine results, but it is the first algorithm that was used by the company, and it is the best-known.
The above centrality measure is not implemented for multi-graphs.

Algorithm

The PageRank algorithm outputs a probability distribution used to represent the likelihood that a person randomly clicking on links will arrive at any particular page. PageRank can be calculated for collections of documents of any size. It is assumed in several research papers that the distribution is evenly divided among all documents in the collection at the beginning of the computational process. The PageRank computations require several passes, called "iterations", through the collection to adjust approximate PageRank values to more closely reflect the theoretical true value.

Simplified algorithm

Assume a small universe of four web pages: A, B, C, and D. Links from a page to itself, or multiple outbound links from one single page to another single page, are ignored. PageRank is initialized to the same value for all pages. In the original form of PageRank, the sum of PageRank over all pages was the total number of pages on the web at that time, so each page in this example would have an initial value of 1. However, later versions of PageRank, and the remainder of this section, assume a probability distribution between 0 and 1. Hence the initial value for each page in this example is 0.25.
The PageRank transferred from a given page to the targets of its outbound links upon the next iteration is divided equally among all outbound links.
If the only links in the system were from pages B, C, and D to A, each link would transfer 0.25 PageRank to A upon the next iteration, for a total of 0.75.
PR(A) = PR(B) + PR(C) + PR(D)

Suppose instead that page B had a link to pages C and A, page C had a link to page A, and page D had links to all three pages. Thus, upon the first iteration, page B would transfer half of its existing value, or 0.125, to page A and the other half, or 0.125, to page C. Page C would transfer all of its existing value, 0.25, to the only page it links to, A. Since D had three outbound links, it would transfer one-third of its existing value, or approximately 0.083, to

A. At the completion of this iteration, page A will have a PageRank of approximately 0.458.
PR(A) = PR(B)/2 + PR(C)/1 + PR(D)/3

In other words, the PageRank conferred by an outbound link is equal to the document's own PageRank score divided by the number of outbound links L( ).
PR(A) = PR(B)/L(B) + PR(C)/L(C) + PR(D)/L(D)

The PageRank value for a page u is dependent on the PageRank values for each page v contained in the set Bu (the set containing all pages linking to page u), divided by the number L(v) of links from page v. The algorithm involves a damping factor for the calculation of the PageRank. It is like the income tax which the govt extracts from one despite paying him itself.