Assignment No 5

To study Supervised Learning and Kernel Methods.

Aim
To Design, Implement SVM for classification with proper data set of your choice.

Objective(s)	
1	To design SVM for classification with proper data set.
2	To study about Kernel methods.

Theory

Introduction:

Support Vector Machine

Support Vector Machines are based on the concept of decision planes that define decision boundaries. A decision plane is one that separates between a set of objects having different class memberships. A schematic example is shown in the illustration below. In this example, the objects belong either to class GREEN or RED. The separating line defines a boundary on the right side of which all objects are GREEN and to the left of which all objects are RED. Any new object (white circle) falling to the right is labeled, i.e., classified, as GREEN (or classified as RED should it fall to the left of the separating line). Classification tasks based on drawing separating lines to distinguish between objects of different class memberships are known as hyperplane classifiers. Support Vector Machines are particularly suited to handle such tasks.

Support Vector Machine (SVM) is primarily a classier method that performs classification tasks by constructing hyperplanes in a multidimensional space that separates cases of different class labels. SVM supports both regression and classification tasks and can handle multiple continuous and categorical variables. For categorical variables a dummy variable is created with case values as either 0 or 1. Thus, a categorical dependent variable consisting of three levels, say (A, B, C), is represented by a set of three dummy variables:

A: {1 0 0}, B: {0 1 0}, C: {0 0 1}

To construct an optimal hyperplane, SVM employs an iterative training algorithm, which is used to minimize an error function. According to the form of the error function, SVM models can be classified into four distinct groups:

- Classification SVM Type 1 (also known as C-SVM classification)
- Classification SVM Type 2 (also known as nu-SVM classification)
- Regression SVM Type 1 (also known as epsilon-SVM regression)
- Regression SVM Type 2 (also known as nu-SVM regression)

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature

spaces.

The advantages of support vector machines are:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of support vector machines include:

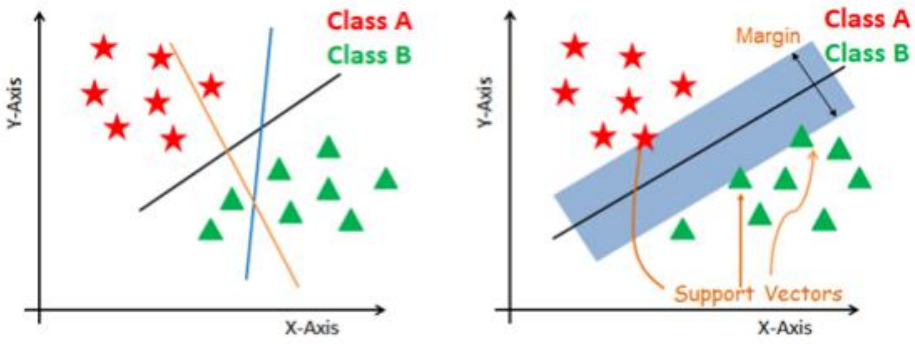
- If the number of features is much greater than the number of samples, the method is likely to give poor performances.
- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation (see Scores and probabilities, below).

The points nearest to the separating hyperplane are called Support Vectors. Only they determine the position of the hyperplane. All other points have no influence!

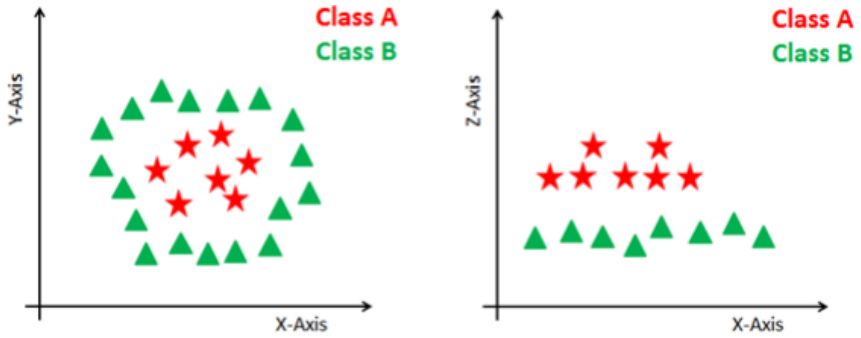
SVM Process

The main objective is to segregate the given dataset in the best possible way. The distance between the either nearest points is known as the margin. The objective is to select a hyperplane with the maximum possible margin between support vectors in the given dataset. SVM searches for the maximum marginal hyperplane in the following steps:

1. Generate hyperplanes which segregates the classes in the best way. Left-hand side figure showing three hyperplanes black, blue and orange. Here, the blue and orange have higher classification error, but the black is separating the two classes correctly.
2. Select the right hyperplane with the maximum segregation from the either nearest data points as shown in the right-hand side figure



Dealing with non-linear and inseparable planes Some problems can't be solved using linear hyperplane, as shown in the figure below (left-hand side). In such situation, SVM uses a kernel trick to transform the input space to a higher dimensional space as shown on the right. The data points are plotted on the x-axis and z-axis (Z is the squared sum of both x and y: z=x^2=y^2). Now you can easily segregate these points using linear separation.



SVM Kernels

The SVM algorithm is implemented in practice using a kernel. A kernel transforms an input data space into the required form. SVM uses a technique called the kernel trick. Here, the kernel takes a low-dimensional input space and transforms it into a higher dimensional space. In other words, you can say that it converts nonseparable problem to separable problems by adding more dimension to it. It is most useful in non-linear separation problem. Kernel trick helps you to build a more accurate classifier.

- **Linear Kernel**

A linear kernel can be used as normal dot product any two given observations. The product between two vectors is the sum of the multiplication of each pair of input values.

K(x, xi) = sum(x * xi)

- **Polynomial Kernel**

A polynomial kernel is a more generalized form of the linear kernel. The polynomial kernel can distinguish curved or nonlinear input space.

K(x,xi) = 1 + sum(x * xi)^d

Where d is the degree of the polynomial.

d=1 is similar to the linear transformation.

The degree needs to be manually specified in the learning algorithm. The most applicable machine learning algorithm for our problem is Linear SVC. Before hopping into Linear SVC with our data, we're going to show a very simple example that should help solidify your understanding of working with Linear SVC. The objective of a Linear SVC (Support Vector Classifier) is to fit to the data you provide, returning a "best fit" hyperplane that divides, or categorizes, your data. From there, after getting the hyperplane, you can then feed some features to your classifier to see what the "predicted" class is. This makes this specific algorithm rather suitable for our uses, though you can use this for many situations. Let's get started. First, we're going to need some basic dependencies:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import style
style.use("ggplot")
from sklearn import svm
```

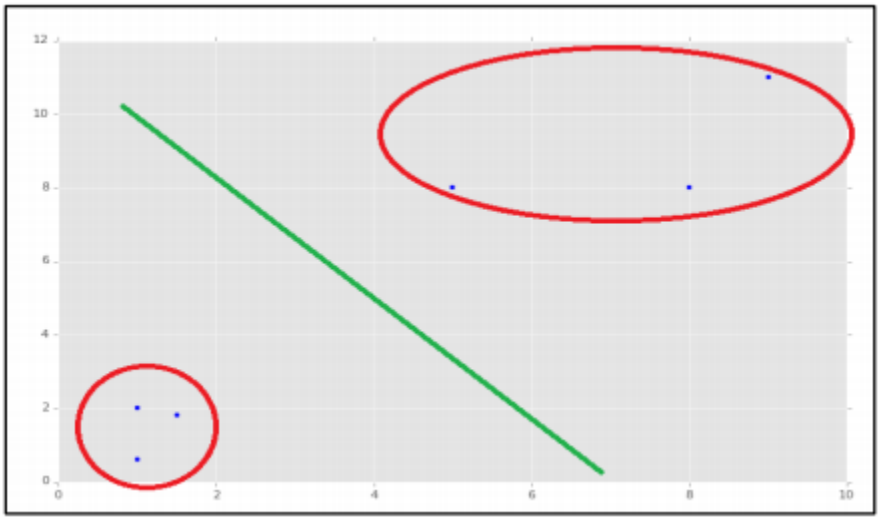
Next, let's consider that we have two features to consider. These features will be visualized as axis on our graph. So something like:

```
x = [1, 5, 1.5, 8, 1, 9]
y = [2, 8, 1.8, 8, 0.6, 11]
```

Then we can graph this data using:

```
plt.scatter(x,y)
plt.show()
```

So this is with two features, and we see we have a 2D graph. If we had three features, we could have a 3D graph. The 3D graph would be a little more challenging for us to visually group and divide, but still do-able. The problem occurs when we have four features, or four-thousand features.



Now, in order to feed data into our machine learning algorithm, we first need to compile an array of the features, rather than having them as x and y coordinate values. Generally, you will see the feature list being stored in a capital X variable. Let's translate our above x and y coordinates into an array that is compiled of the x and y coordinates, where x is a feature and y is a feature

sv

```
X = np.array([[1,2],
              [5,8],
              [1.5,1.8],
              [8,8],
              [1,0.6],
              [9,11]])
```

Now that we have this array, we need to label it for training purposes. There are forms of machine learning called "unsupervised learning," where data labeling isn't used, as is the case with clustering, though this example is a form of supervised learning. For our labels, sometimes referred to as "targets," we're going to use 0 or 1.

```
y = [0,1,0,1,0,1]
```

Just by looking at our data set, we can see we have coordinate pairs that are "low" numbers and coordinate pairs that are "higher" numbers. We've then assigned 0 to the lower coordinate pairs and 1 to the higher feature pairs. These are the labels. In the case of our project, we will wind up having a list of numerical features that are various statistics about stock companies, and then the "label" will be either a 0 or a 1, where 0 is under-perform the market and a 1 is out-perform the market. Moving along, we are now going to define our classifier:

```
clf = svm.SVC(kernel='linear', C = 1.0)
```

We're going to be using the SVC (support vector classifier) SVM (support vector machine). Our kernel is going to be linear, and C is equal to 1.0. What is C you ask? Don't worry about it for now, but, if you must know, C is a valuation of "how badly" you want to properly classify, or fit, everything. The machine learning field is relatively new, and experimental. There exist many debates about the value of C, as well as how to calculate the value for C. We're going to just stick with 1.0 for now, which is a nice default parameter.

```
clf.fit(X,y)
```

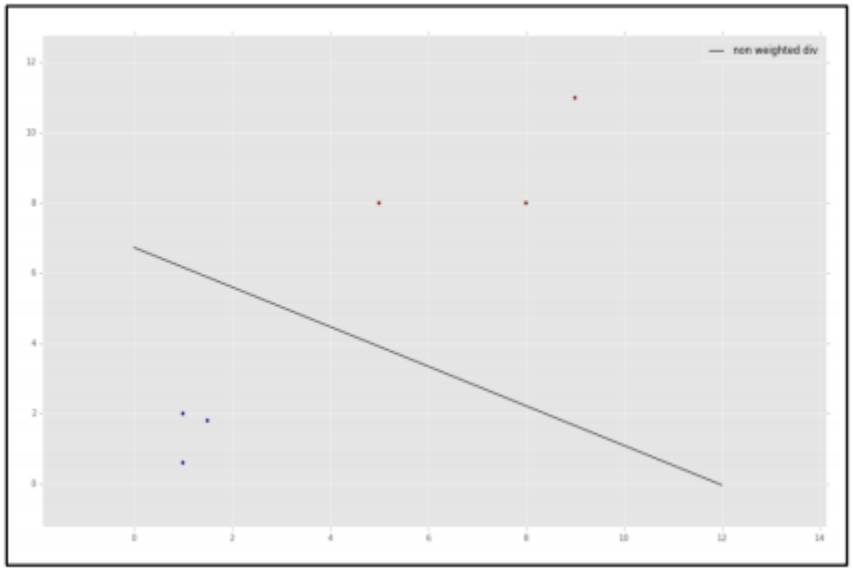
Next, we can predict and test. Let's print a prediction:

```
print(clf.predict([0.58,0.76]))
```

to visualize your data:

```
w = clf.coef_[0]
print(w)
a = -w[0] / w[1]
xx = np.linspace(0,12)
yy = a * xx - clf.intercept_[0] / w[1]
h0 = plt.plot(xx, yy, 'k-', label="non weighted div")
plt.scatter(X[:, 0], X[:, 1], c = y)
plt.legend()
plt.show()
```

The result:



Conclusion
Implementation of SVM for classification for a data set was Successfully done.

Kernel methods give a systematic and principled approach to training learning machines and the good generalization performance achieved can be readily justified using statistical learning theory or Bayesian arguments.

Popular kernels:

- Fisher kernel
- Graph kernels
- Kernel smoother
- Polynomial kernel
- RBF kernel
- String kernels

[1]import numpy as np

[2]import matplotlib.pyplot as plt

[3]from matplotlib import style

[4]style.use("ggplot")

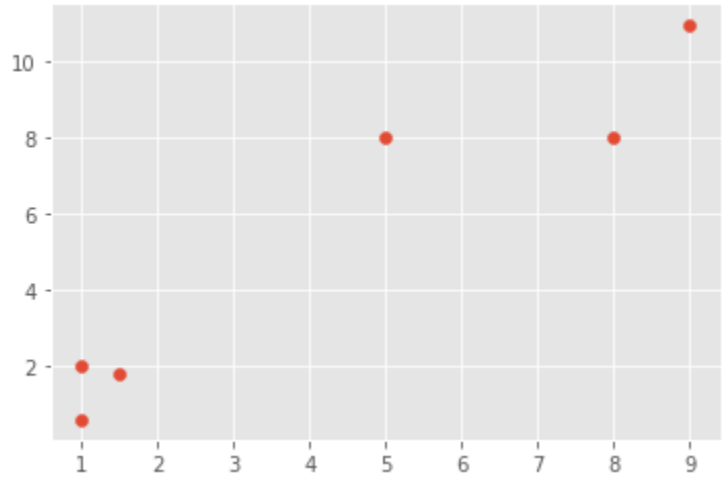
from sklearn import svm

[5]x = [1, 5, 1.5, 8, 1 , 9]

y = [2, 8 , 1.8 , 8 , 0.6 ,11]

[6]plt.scatter(x,y)

plt.show()



[7]X = np.array([[1,2],[5,8],[1.5,1.8],[8,8],[1,0.6],[9,11]])

[8]Y = [0 ,1,0 ,1 ,0 , 1]

[9]clf = svm.SVC(kernel = 'linear', C = 1.0)

[10]clf.fit(X,Y)

>SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,

decision_function_shape='ovr', degree=3, gamma='auto_deprecated',

kernel='linear', max_iter=-1, probability=False, random_state=None,

shrinking=True, tol=0.001, verbose=False)

[11]accuracy = clf.score(X,Y)

print(accuracy)

>1.0

[12]print(clf.predict([[20.58 , 20.76]]))

>1

[13]w =clf.coef_[0]

print(w)

a = -w[0]/w[1]

xx = np.linspace(0,12)

yy = a* xx -clf.intercept_[0]/w[1]

h0 = plt.plot(xx, yy , 'k-', label = "non weigthed div")

>[0.1380943 0.24462418]