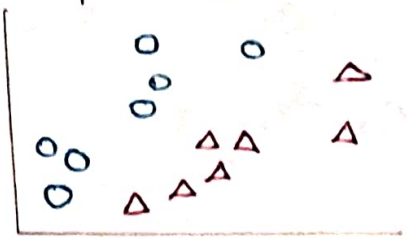
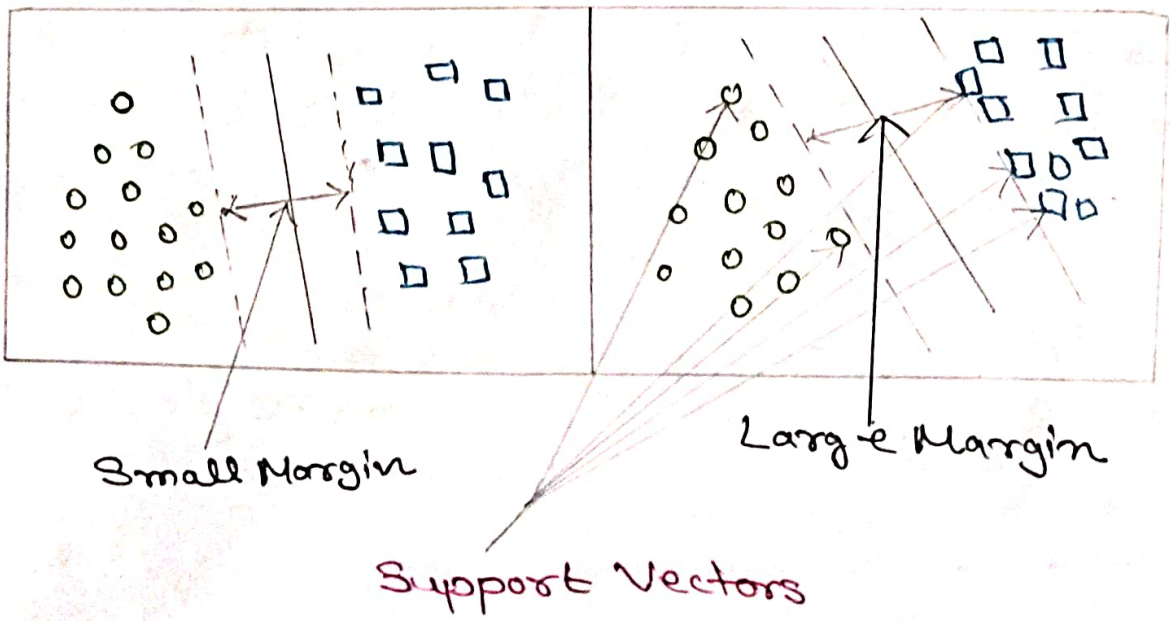
SVM

How does SVM work?

- Let's imagine we have two tags: red and blue, and our data has two features: x and y.
- We want a classifier that, given a pair of (x,y) coordinates, outputs if it's either red or blue.
- We plot our already labeled training data on a plane.



- A support vector machine takes these data points and outputs the hyperplane (which in two dimensions it's simply a line) that best separates the tags.
- This line is the decision boundary: anything that falls to one side of it we will classify as blue and anything that falls to the other as red.



What is Support Vector Machine?

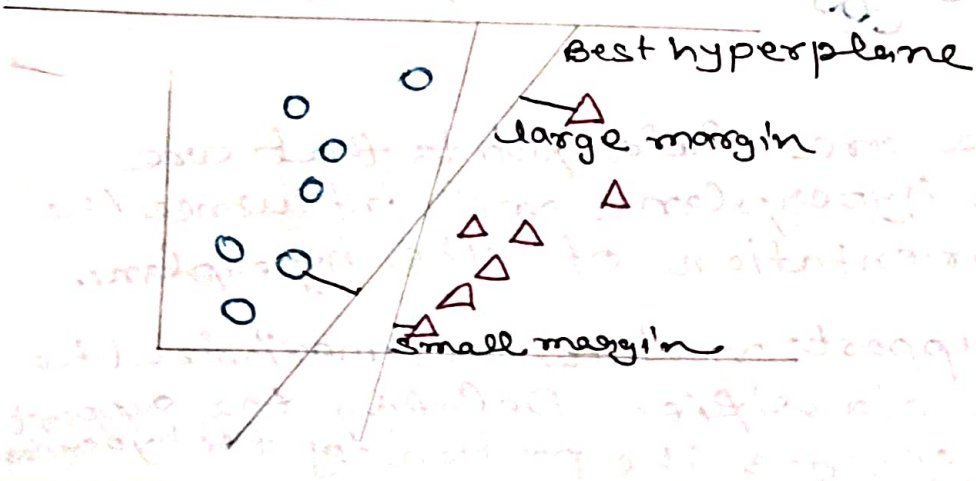
- The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N - the no. of features) that distinctly classifies the data points.
- To separate the two classes of data points, there are many possible hyperplanes that could be chosen.
- Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes.
- Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

hyperplane

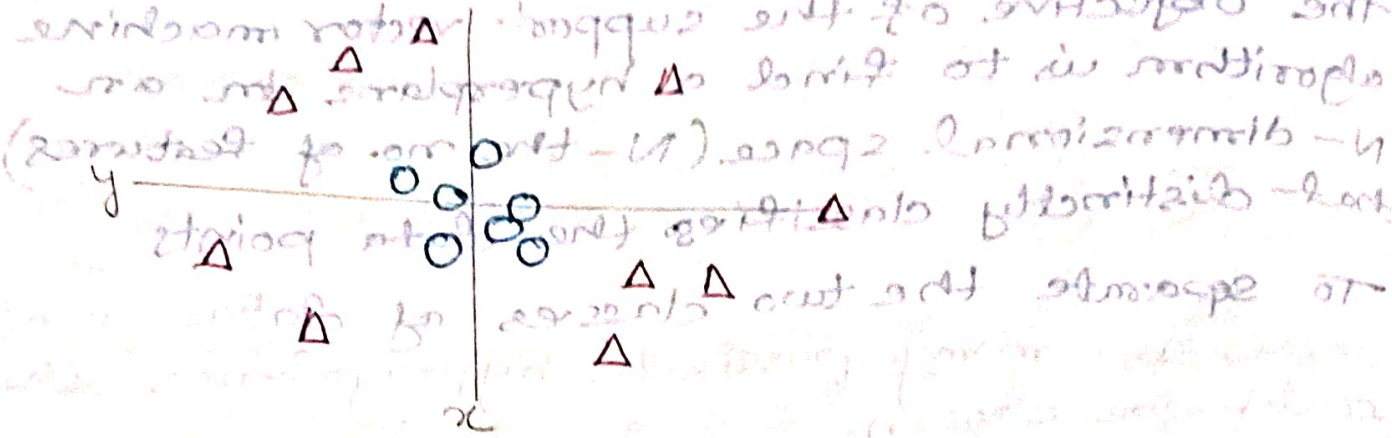
Hyperplanes are decision boundaries that help classify the data points.

Data points falling on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyperplane depends upon the no. of features. If the no. of input features is 2, then the hyperplane is just a line.

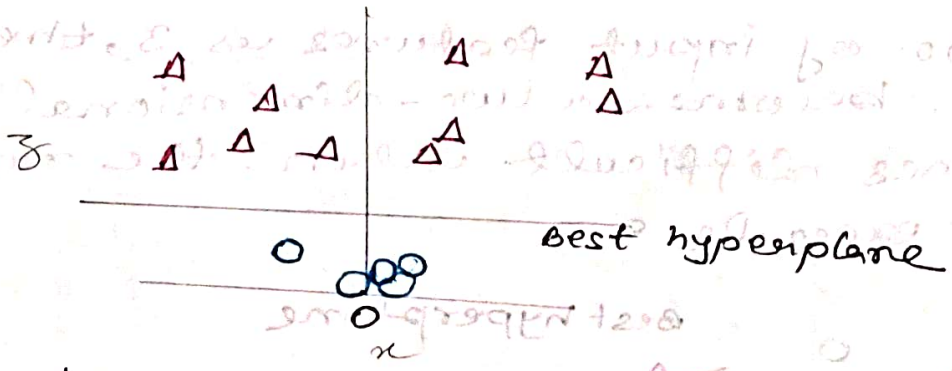
If the no. of input features is 3, then the hyperplane becomes a two-dimensional plane. It becomes difficult when the no. of features exceeds 3.



Nonlinear data

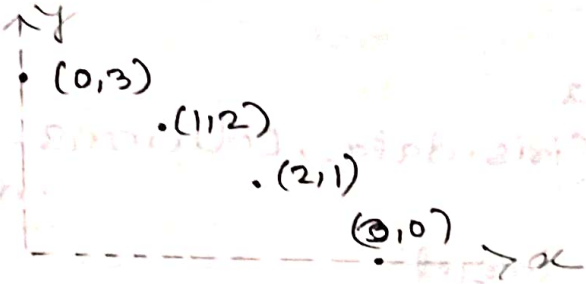


- It's pretty clear that there's not a linear decision boundary (a single straight line that separates both tags).
- However, the vectors are very clearly segregated and it looks as though it should be easy to separate them.
- So here's what we'll do: we will add a third dimension.
- Up until now we had two dimensions x and y.
- We create a new dimension, and we rule that it be calculated a certain way that is convenient for us: $= x^2 + y^2$ (you'll notice that's the equation for a circle).



- Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane.
- Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperp

- These are the points that helps us build our SVM.
- SVM has a technique called the kernel trick.
- These are functions which takes low dimensional input space and transform it to a higher dimensional space i.e. it converts not separable problem to separable problem, these functions are called Kernels.
- It is mostly useful in non-linear separation problem.



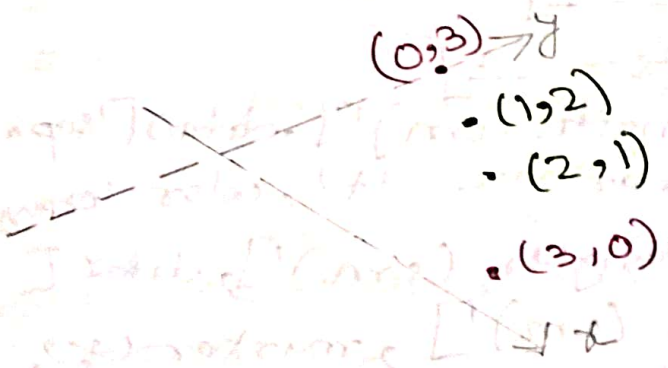
$(x, y) \longrightarrow (x, y, xy)$

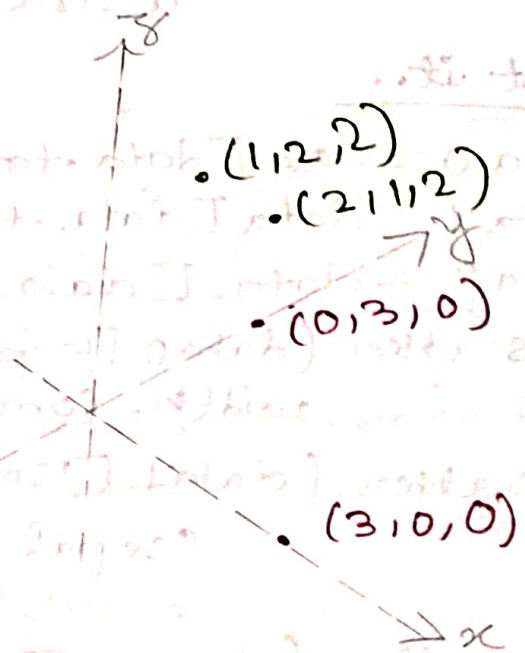
$(0,3) \longrightarrow (0,3,0)$

$(1,2) \longrightarrow (1,2,2)$

$(2,1) \longrightarrow (2,1,2)$

$(3,0) \longrightarrow (3,0,0)$





SVM in notebook..

```
import pandas as pd
from sklearn.datasets import load_iris
iris = load_iris()
from matplotlib import pyplot as plt
from sklearn.model_selection import
                        train_test_split.

from sklearn.svm import SVC
```

Load data..

```
dir(iris)
iris.feature_names
data= pd.DataFrame(iris.data, columns=iris.feature_
                                        names)

data['targets'] = iris.target
iris.target_names
data['flower_name'] = data.targets.apply(lambda
                x: iris.target_names[x])
```

Plot it..

```
data0 = data[data.targets==0]
data1 = data[data.targets ==1]
data2 = data[data.targets ==2]
plt.scatter(data0['sepal length (cm)'],data0['sepa
            width (cm)'], marker ='+', color='oran
plt.scatter(data1['sepal length (cm)'], data1[
            'sepal width (cm)'], marker='*',
                color='green')
plt.scatter(data2['petal length (cm)'],
        data2['petal width (cm)'], marker='+',
        color='orange') pl
plt.scatter(data1['petal length (cm)'],
        data1['petal width (cm)'], marker='*
                color='green')
```

Split..

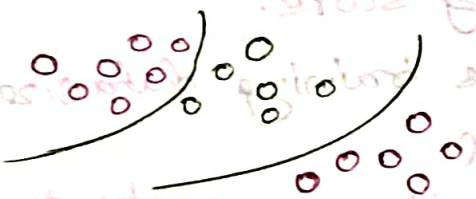
```
x = data.drop(['targets','flower_name'],
              axis='columns')

y = data.targets
xtrain, xtest, ytrain, ytest = train_test_split
                    (x, y, test_size = .2)
```

Create the model and test it..

```
model = SVC()
model.fit(xtrain, ytain)
model.score(xtest, ytest)
model.predict(xtest)
```

Kernel SVM

In this case of non-linearly separable data, such as the one shown below, a straight line cannot be used as a decision boundary



In case of non-linearly separable data, the simple SVM algorithm cannot be used. Rather a modified version of SVM, called Kernal SVM, is used.

In case of linear the kernel is linear and by default also its linear.

Linear kernal

```
svclassifier = SVC(kernel='linear')
```

Different Kernels..

① Polynomial Kernel → In this case of polynomial Kernel, you also have to pass a value for the degree parameter of the SVC class. This basically is the degree of the polynomial. Take a look at how we can use a polynomial kernel to implement Kernel SVM:

svclassifier = SVC(kernel='poly', degree=8)

② Gaussian Kernel – Take a look at how we can use polynomial kernel to implement kernel SVM:

svclassifier = SVC (kernel='rbf')

Exercise

1) Train SVM classifier using sklearn digits dataset (i.e. from sklearn.datasets import load_digits) and then.

a) Measure accuracy of your model

b) Tune your model further using regularization and gamma parameters and try to come up with highest accuracy score.

c) Use 80% of samples as training data size.

```
import pandas as pd
from sklear.model_selection import train_test_split
from matplot import pyplot as plt.
from sklearn.datasets import load_digits
from sklearn.svm import SVC
digits = load_digits()
dir(digits)
df = pd.DataFrame(digits.data, digits.features_name)
df['target'] = digits.target
digits.target_names
df[                    ] =
x = df.drop(['target'], axis=1)
y = df['target']
xtrain, xtest, ytrain, ytest = test_train_split(x,
                                    y, test_size=0.2)
```

```
model = SVC(gamma='auto')
model.fit(xtrain, ytrain)
model.predict(xtest)
model.score(xtest, ytest)
```