Q1. What is the mathematical formula for a linear SVM?

A linear support vector machine (SVM) is a binary classification algorithm that tries to find the best hyperplane that separates the two classes in a high-dimensional space. The mathematical formula for a linear SVM can be expressed as:

```
f(x) = sign(w . x + b)
```

where:

x is the input vector of dimension n

w is the weight vector of dimension n

b is the bias term . denotes the dot product of w and x

sign is the sign function that returns 1 if the argument is positive, -1 if it's negative, and 0 if it's zero.

Q2. What is the objective function of a linear SVM?

The objective function of a linear SVM is to maximize the margin between the decision boundary (i.e., the hyperplane that separates the two classes) and the closest points from each class. The margin is defined as the distance between the decision boundary and the closest points from each class.

minimize: 1/2 * ||w||^2 subject to: y_i (w . x_i + b) >= 1 for i = 1, 2, ..., n

where:

w is the weight vector

b is the bias term

x_i is the i-th input vector

y_i is the i-th label (+1 or -1)

n is the number of training examples

Q3. What is the kernel trick in SVM?

The kernel trick in SVM (Support Vector Machines) is a technique used to transform non-linearly separable data into a higher-dimensional space, where it becomes linearly separable. This is achieved by applying a non-linear transformation to the input data without explicitly computing the transformation. Instead, the transformation is done using a kernel function, which is a mathematical function that calculates the dot product of two points in the transformed space without computing the transformation explicitly.

In SVM, the kernel trick is used to transform the input data into a higher-dimensional space, where it becomes linearly separable. The transformed data is then classified using a linear SVM, which can easily find the optimal hyperplane in the higher-dimensional space.

The most commonly used kernel functions in SVM are:

Linear kernel: This kernel function is used when the data is already linearly separable.

Polynomial kernel: This kernel function is used when the data has a curved decision boundary.

Gaussian kernel (RBF): This kernel function is used when the data has a complex decision boundary that is not easy to define mathematically.

Q4. What is the role of support vectors in SVM Explain with example

Support vectors play a crucial role in SVMs (Support Vector Machines) as they define the decision boundary or the hyperplane that separates the two classes. These are the data points that are closest to the decision boundary and have the maximum influence on the position and orientation of the hyperplane.

During the training process of an SVM, the algorithm tries to find the optimal hyperplane that maximizes the margin between the two classes while minimizing the classification error. However, not all the data points are equally important in determining the hyperplane. Only the support vectors, which are the data points closest to the hyperplane, are used to define the position and orientation of the hyperplane.

Example Suppose we have a dataset of points in a two-dimensional space with two classes, A and B. The objective of the SVM algorithm is to find the optimal hyperplane that separates the two classes while maximizing the margin between them.

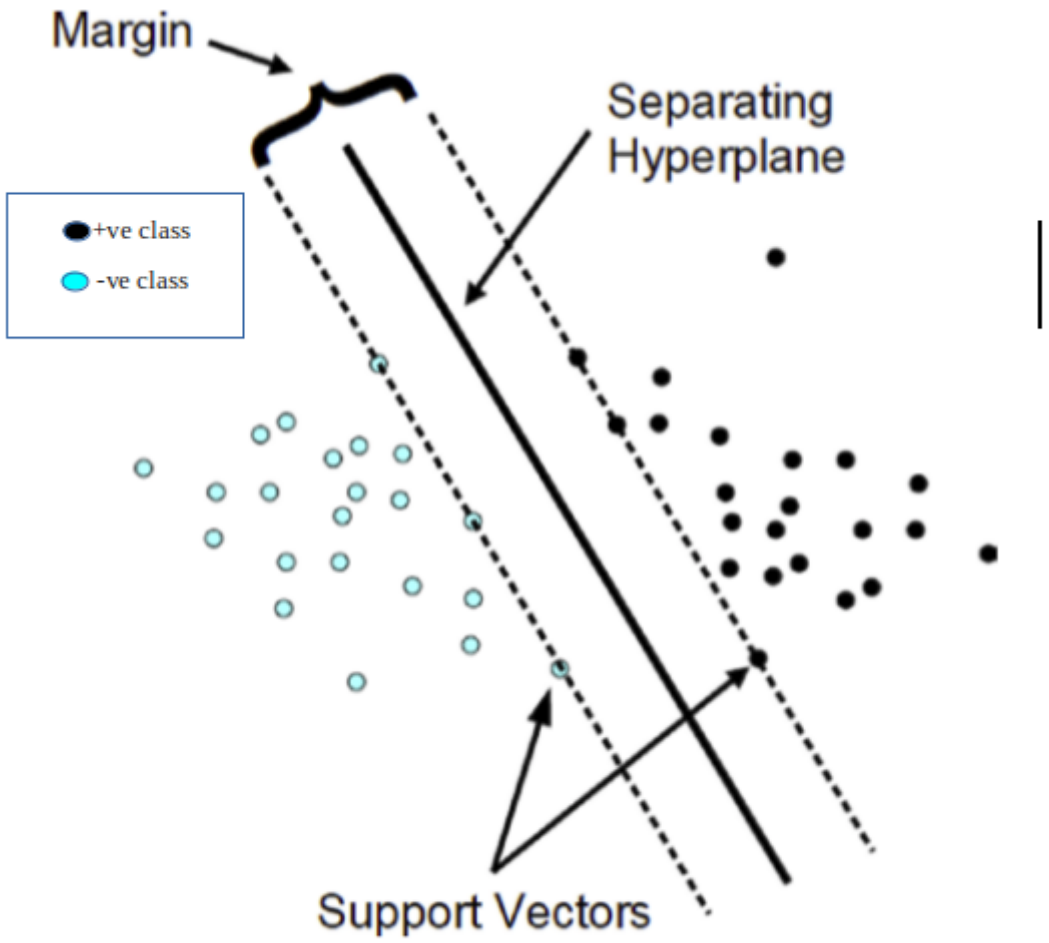
During the training process, the algorithm identifies the support vectors, which are the data points that lie closest to the hyperplane. These support vectors determine the position and orientation of the hyperplane, while all other points are not relevant for the final decision boundary.

For example, consider a dataset with three data points: two blue points representing class A and one red point representing class B. If we assume that the optimal hyperplane is a straight line, then the middle blue point and the red point are the support vectors. These two points define the hyperplane, while the other blue point is not a support vector and does not have any influence on the position and orientation of the hyperplane.

Q5. Illustrate with examples and graphs of Hyperplane, Marginal plane, Soft margin and Hard margin in SVM?

Hyperplane: The hyperplane is the decision boundary that separates the data points of two different classes. In a binary classification problem, the hyperplane is a line in 2D space or a plane in 3D space that separates the two classes. The hyperplane is determined by the SVM algorithm during the training process.

Example: In the graph below, the hyperplane is a straight line that separates the two classes (blue and red) in a 2D space.



Marginal plane: The marginal plane is a plane parallel to the hyperplane that is equidistant from the support vectors. The distance between the marginal plane and the hyperplane is called the margin. The margin is maximized in SVM to achieve better generalization performance.

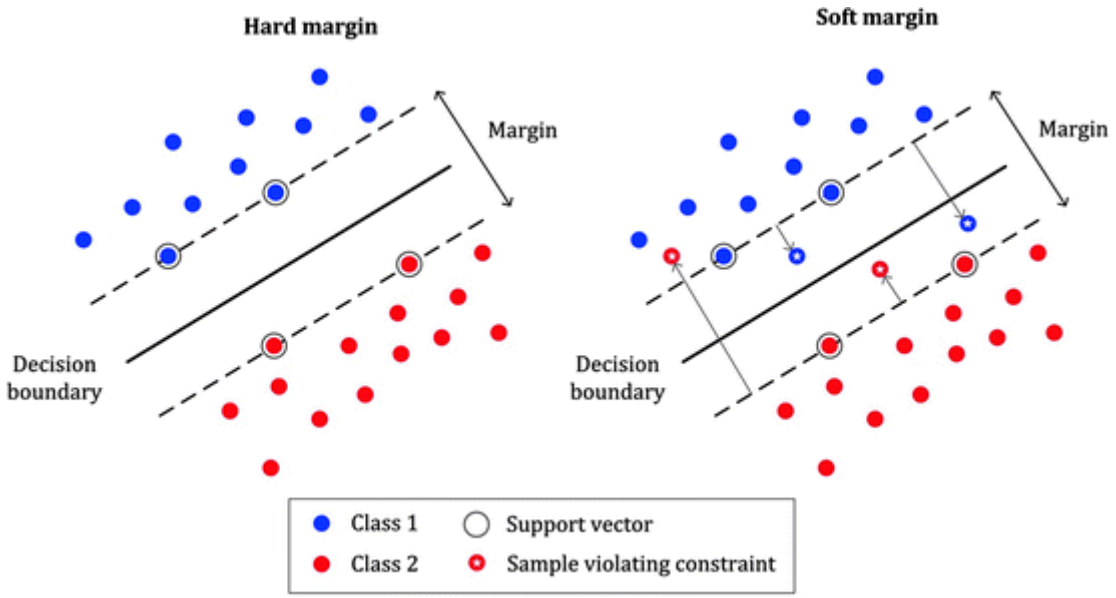
Example: In the graph below, the marginal plane is the dotted line parallel to the hyperplane and equidistant from the support vectors. The margin is the distance between the hyperplane and the marginal plane.

Soft margin: The soft margin SVM allows some data points to be misclassified or fall within the margin. This is done to improve the generalization performance of the SVM algorithm.

Example: In the graph below, the soft margin SVM allows some data points to be misclassified or fall within the margin. The misclassified points are shown as white circles, and the margin is represented by the dotted line. The soft margin SVM allows for some errors in the classification to achieve better generalization performance.

Hard margin: The hard margin SVM does not allow any data points to be misclassified or fall within the margin. This is done to achieve better accuracy on the training set.

Example: In the graph below, the hard margin SVM does not allow any data points to be misclassified or fall within the margin. The margin is represented by the dotted line, and all the data points are correctly classified. However, the hard margin SVM may not perform well on new data due to overfitting.



Q6. SVM Implementation through Iris dataset.

Bonus task: Implement a linear SVM classifier from scratch using Python and compare its performance with the scikit-learn implementation.

~ Load the iris dataset from the scikit-learn library and split it into a training set and a testing setl

~ Train a linear SVM classifier on the training set and predict the labels for the testing setl

~ Compute the accuracy of the model on the testing setl

~ Plot the decision boundaries of the trained model using two of the featuresl

~ Try different values of the regularisation parameter C and see how it affects the performance of the model.

In [9]:

```
from sklearn.datasets import load_iris
df = load_iris()
df.keys()
```

Out[9]:

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_na
mes', 'filename', 'data_module'])
```

In [10]:

```
x  = df.data
y = df.target
```

In [13]:

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,random_state= 42,test_size= 0.2)
```

In [17]:

```
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

Out[17]:

```
((120, 4), (30, 4), (120,), (30,))
```

In [20]:

```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
param = {'C':[0.1,1,10,100]}
model = GridSearchCV(SVC(),param_grid=param,cv = 6,verbose = 3)
model.fit(x_train,y_train)
```

```
Fitting 6 folds for each of 4 candidates, totalling 24 fits
[CV 1/6] END .............................C=0.1;, score=0.950 total time=
0.0s
[CV 2/6] END .............................C=0.1;, score=0.800 total time=
0.0s
[CV 3/6] END .............................C=0.1;, score=0.900 total time=
0.0s
[CV 4/6] END .............................C=0.1;, score=0.900 total time=
0.0s
[CV 5/6] END .............................C=0.1;, score=0.950 total time=
0.0s
[CV 6/6] END .............................C=0.1;, score=0.850 total time=
0.0s
[CV 1/6] END ...............................C=1;, score=1.000 total time=
0.0s
[CV 2/6] END ...............................C=1;, score=0.950 total time=
0.0s
[CV 3/6] END ...............................C=1;, score=0.900 total time=
0.0s
[CV 4/6] END ...............................C=1;, score=0.900 total time=
0.0s
[CV 5/6] END ...............................C=1;, score=1.000 total time=
0.0s
[CV 6/6] END ...............................C=1;, score=0.950 total time=
0.0s
[CV 1/6] END ..............................C=10;, score=0.950 total time=
0.0s
[CV 2/6] END ..............................C=10;, score=0.950 total time=
0.0s
[CV 3/6] END ..............................C=10;, score=0.950 total time=
0.0s
[CV 4/6] END ..............................C=10;, score=0.900 total time=
0.0s
[CV 5/6] END ..............................C=10;, score=1.000 total time=
0.0s
[CV 6/6] END ..............................C=10;, score=0.950 total time=
0.0s
[CV 1/6] END .............................C=100;, score=0.950 total time=
0.0s
[CV 2/6] END .............................C=100;, score=0.950 total time=
0.0s
[CV 3/6] END .............................C=100;, score=0.900 total time=
0.0s
[CV 4/6] END .............................C=100;, score=0.900 total time=
0.0s
[CV 5/6] END .............................C=100;, score=1.000 total time=
0.0s
[CV 6/6] END .............................C=100;, score=1.000 total time=
0.0s
```

Out[20]:

In [21]:

```
GridSearchCV(cv=6, estimator=SVC(), param_grid={'C': [0.1, 1, 10, 100]},
             verbose=3)
model.best_params_
```

Out[21]:

```
{'C': 1}
```

In [22]:

```
y_pred = model.predict(x_test)
```

In [23]:

```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
print(confusion_matrix(y_test,y_pred))
print(accuracy_score(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

```
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

In [ ]: