01 - Supervised Learning - Classification - SVM(Solution)_st122097_Thantham

September 5, 2021

1 Programming for Data Science and Artificial Intelligence

1.1 6.3 Supervised Learning - Classification - SVM

1.2 Name: Thantham Khamyai

1.3 Student ID: 122097

1.4 Tasks to be completed

- **Load defined dataset** to numpy array, with first two columns as features and last as target
- **Plot the data using a scatter plot**
- **Perform the SVM** classification using our scratch code

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

The Dataset defined is below

[2]:

Then, we are going to load the multi-dimension list into numpy array form

[3]:
```
data = np.array(dataset) # convert list to numpy array
```

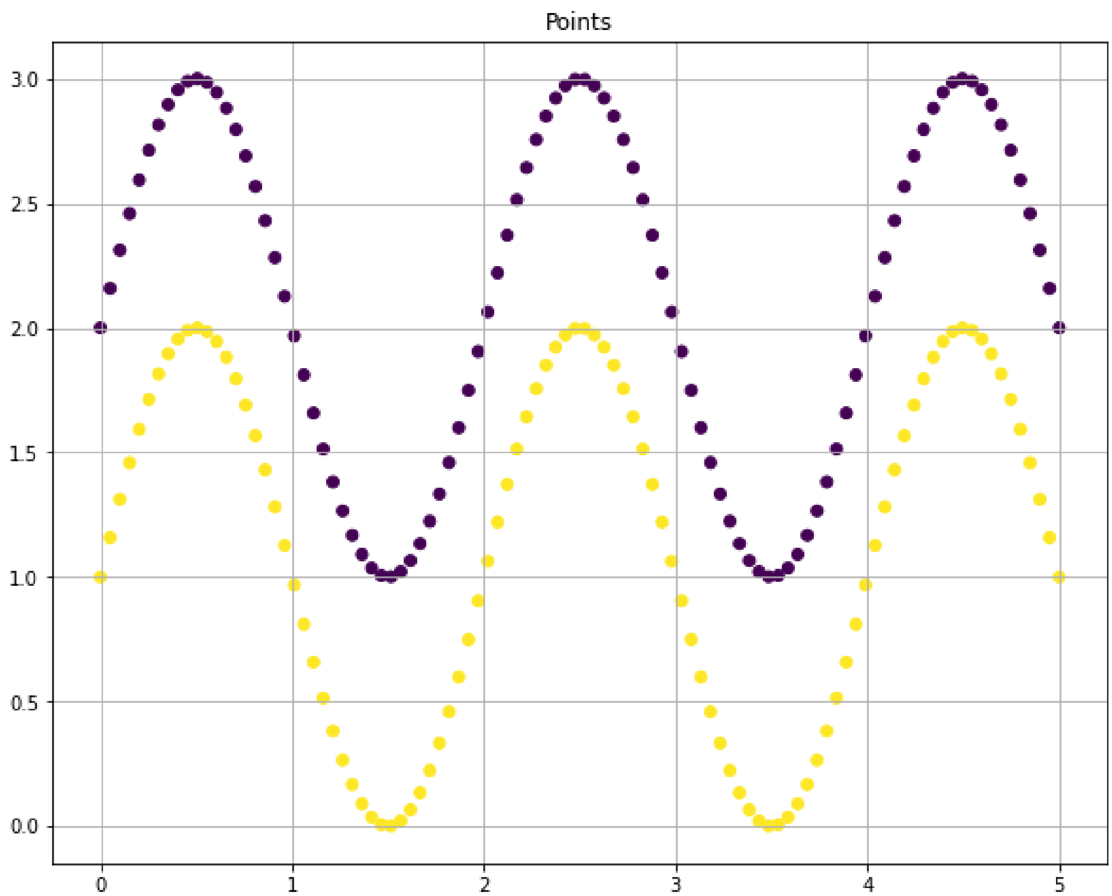
From Raw numpy array data, it will be extracted into X and y space. By saparating X as 2 first columns and y as last column, we should also **convert the binary class of y into space {-1, +1} instead of {0,1}** because SVM will solve the decision boundary problem using sign operation to classify classes -1 and 1. So, we programmatically define the class 0 into -1 instead.

[4]:
```
print(f'Raw data: \n {data[:10]}')

X = data[:,:2] # extract first 2 columns to be X
y = data[:, -1] # extract last column to be y

y[y==0] =-1 # change the values which are 0 in y to -1

print(f'Shape of X is {X.shape}')
print(f'Example of X is {X[:5]}')

print(f'Shape of y is {y.shape}')
print(f'Example of y is {y[:5]}')
```

```
Raw data:
 [[3.63636364 1.090368   0.        ]
 [4.09090909 2.28173256 0.        ]
 [0.1010101  1.31203345 1.        ]
 [0.45454545 1.98982144 1.        ]
 [1.91919192 1.74885201 0.        ]
 [1.76767677 0.333231   1.        ]
 [2.57575758 2.97181157 0.        ]
 [1.06060606 0.81074876 1.        ]
 [2.97979798 1.06342392 1.        ]
 [1.86868687 1.59906946 0.        ]]
Shape of X is (200, 2)
Example of X is [[3.63636364 1.090368  ]
 [4.09090909 2.28173256]
 [0.1010101  1.31203345]
 [0.45454545 1.98982144]
 [1.91919192 1.74885201]]
Shape of y is (200,)
Example of y is [-1. -1.  1.  1. -1.]
```

We would see that the target class that are conventionally 0 and 1, but now they are -1 and 1. It would be also better if we visualize those points into scatter plot with class saparation like this

[5]:
```
plt.figure(figsize=(10,8))
plt.scatter(X[:,0], X[:,1], c=y) # plot scatter graph
plt.title('Points')
plt.grid(True)
plt.show()
```



It seems to be curvy line arranged by a number of points. By the way, we will perform SVM classification by this data.

Next, the X and y will be splitted into training and testing set. This can prove that the model can perform classification peroperly.

[6]:
```
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,␣
↪random_state=48) # fix random state to control index of splitting
```

The SVM model from scratch cane be constructed by following. To explain in the proper way,

1. Initialize model - define kernel tricks method - define Regularization term (**C**) ~ None as default - ddefine degree of polynomial (If kerrnel 'poly' is activated, degree will be implemented)

2. fitting model - denerate K(x,x) using kernel trick one of three methods {'linear', 'poly', 'gs'} - define convex parameter (P, q, A, b) for dual problem that will be solved using Quadratic Problem - If C term is defined, G and h Quadratic parameters will be initialized - solve that Quadratic Problem to get α - extract support vector point indices from filtering α - calculate b from constraint term - if 'linear' kernel is implemented, calculate w, else w = None - completed calculating α, b and w

3. Predict y - if 'linear' kernel method implemented, perform simply calculating

$$\hat{y} = \operatorname{sign}\left(\mathbf{w}^{\mathbf{T}} \phi\left(\mathbf{x}^{\text{test}}\right) + b\right)$$

- else, implementing Kernel tricks following initiated in model

$$\hat{y} = \operatorname{sign}\left(\sum_{i=1}^{m} \alpha_i y^{(i)} \phi\left(x^{(i)}\right)^T \phi\left(\mathbf{x}^{\text{test}}\right) + b\right)$$

- Return predicted y within sign operation using numpy.sign

[7]:
```
from numpy import linalg
from cvxopt import matrix, solvers
import pylab as pl

class SVM:

    # 1. Model Constructor
    def __init__(self, kernel='linear', C=None, degree=None):
        self.kernel = kernel
        self.C = C
        self.degree = degree


    def linear_kernel(self,x1, x2):
        return np.dot(x1, x2) # X @ X'

    def polynomial_kernel(self,x, y, d):
        return (1 + np.dot(x, y)) ** d # (1 + X@X')^degree

    def gaussian_kernel(self,x, y, sigma=0.9999):
        return np.exp(-linalg.norm(x-y)**2 / (2 * (sigma ** 2))) # EXP ^ (␣
↪-norm(x-y)^2 / 2sig.^2 )

    # 2. Fitting model
    def fit(self, X, y):

        # Get no_samples, no_features
        m, n = X.shape
```

```
        # 2.1 initial Kernel trick X,X
        K = np.zeros((m,m))

        # 2.1 Check kernel trick and preform tranformation
        for i in range(m):
            for j in range(m):

                if self.kernel=='linear':
                    K[i,j] = self.linear_kernel(X[i], X[j])
                elif self.kernel =='poly':

                    K[i,j] = self.polynomial_kernel(X[i], X[j], 2 if self.
↪degree == None else self.degree)
                elif self.kernel == 'gs':
                    K[i, j] = self.gaussian_kernel(X[i], X[j])
                else:
                    raise ValueError("Invalid: {'linear', 'poly', 'gs'}")

        # 2.2 Define Quadratic problem parameters
        P = matrix(np.outer(y,y) * K)
        q = matrix(np.ones(m) *-1)
        A = matrix(y, (1, m))
        b = matrix(0.0)

        # 2.3 If C term is defined, generate G and h parameter following its␣
↪value
        if self.C is None:
            G = matrix(np.diag(np.ones(m) *-1))
            h = matrix(np.zeros(m))
        else:
            tmp1 = np.diag(np.ones(m) * -1)
            tmp2 = np.identity(m)
            G = matrix(np.vstack((tmp1, tmp2)))

            tmp1 = np.zeros(m)
            tmp2 = np.ones(m) * self.C
            h = matrix(np.hstack((tmp1, tmp2)))

        # 2.4 Solve Quadratic Problem
        solution = solvers.qp(P, q, G, h, A, b)

        # 2.4 get Alpha
        self.a = np.ravel(solution['x']) # get all alpha values that will be␣
↪define further support vectors

        # 2.5 Extract Support vectors from alpha indicies
```

```
        sv_idx = self.a > 1e-5 # extract support vector index from alpha thresholding
        ind = np.arange(len(self.a))[sv_idx] # get ind
        self.a = self.a[sv_idx] # get alpha at each support vector indx
        self.sv = X[sv_idx] # get support vector X
        self.sv_y = y[sv_idx] # get support vector y
        print("%d support vectors from %d points" % (len(self.a), m))

        # 2.6 Calculate b
        self.b = 0 # init b
        for i in range(len(self.a)): # for each alpha <- support vectors
            self.b += self.sv_y[i] # add y of support vector i
            self.b -= np.sum(self.a * self.sv_y * K[ind[i], sv_idx]) # Sum[ alpha * y * k(x,x')  ]
        self.b /= len(self.a) # divide n of alpha after summation

        # 2.7 If linear kernel used -> calcualte weight for each feature
        if self.kernel == 'linear':
            self.w = np.zeros(n)
            for i in range(len(self.a)):
                self.w += self.a[i] * self.sv_y[i] * self.sv[i] # w = alpha * y * x
        else:
            self.w = None

        # Return them if wanted
        return self.sv, self.sv_y, self.a, self.w, self.b


    def predict(self, X):

        # if 'linear' kernel used
        if self.w is not None:

            return np.dot(X, self.w) + self.b # wTx + b

        else:

            y_pred = np.zeros(len(X)) # init yhat for other kernel tricks

            for i in range(len(X)): # each predicted y
                s = 0 # init s for prediction
                for a_i, sv_y_i, sv_i in zip(self.a, self.sv_y, self.sv): # each support vector -> alpha, y, x

                    if self.kernel =='poly':
                        # s <- s + [ alpha * y * k(x,x') ] whether 'poly' kernel
```

```
                        s += a_i * sv_y_i * self.polynomial_kernel(X[i], sv_i,␣
↪2 if self.degree == None else self.degree)
                    else:
                        # s <- s + [ alpha * y * k(x,x') ] whether 'gs' kernel
                        s += a_i * sv_y_i * self.gaussian_kernel(X[i], sv_i)

                y_pred[i] = s # save that predicted

        return np.sign((y_pred + self.b)).astype(int) # sign opertation for wTx␣
↪+b

    def plot_contour(self, X, y):
        # plot the resulting classifier
        h = 0.1 # define interval of mesh
        x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1 # define Min Max X␣
↪axis
        y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1 # define Min Max y␣
↪axis

        xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min,␣
↪y_max, h)) # Compute mesh grid for that min max xy

        points = np.c_[xx.ravel(), yy.ravel()] # flattern mesh for further␣
↪predict classes

        Z = self.predict(points) # use that flattened xy to predict mesh output
        Z = Z.reshape(xx.shape) # reshape output the same as mesh dimension

        plt.figure(figsize=(10,10)) # define plot figure size
#         plt.contourf(xx, yy, Z, cmap='Wistia', alpha=0.8)
        plt.pcolormesh(xx, yy, Z,cmap='viridis',shading='auto',alpha=0.1) # put␣
↪that meshxy and output z to grid plot
        # plt the points
        plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap=plt.cm.Spectral) # plot␣
↪scatter for All X and y
        plt.grid(True) # show grid also
```

From the construced model, there are selective parameters to be used,

- **Kernel tricks method**: kernel = {'linear', 'poly', 'gs'} Stand for Linear, Polynomial and Gaussian methods
- **C**: e.g. C = 1, 3, 10 ;numeric for defining whether Hard margin or Soft margin (define some number to activate Soft margin)
- **Degree**: e.g. degree = 1, 3, 5 ;in case kernel trick 'poly' is activated, we can define degree of polynomial to power up X into K(X,X)

Then, lets implement the SVM classification and fitting model. The first try will be defining kernel as **'gs' or gaussian kernel, keeping it Hard margin (None C)**

[8]:
```python
model = SVM(kernel='gs', C=None) # 'gs' kernel and None C to activate Hard␣
↪margin
sv_x, sv_y, a, w, b = model.fit(X_train, y_train)

print(f'Example Support Vector X is: (shape -> {sv_x.shape} )\n {sv_x[:5]}')
print(f'Example Support Vector y is: (shape -> {sv_y.shape} ) \n {sv_y[:5]}')
print(f'Example alpha: (shape -> {a.shape} ){a[:5]}')
print(f'w: {w}')
print(f'b: {b}')
```

```
     pcost       dcost       gap    pres   dres
 0: -6.4106e+01 -1.8193e+02  4e+02  1e+01  2e+00
 1: -1.5864e+02 -2.8339e+02  2e+02  6e+00  1e+00
 2: -2.5746e+02 -3.4513e+02  1e+02  2e+00  4e-01
 3: -3.0230e+02 -3.8689e+02  1e+02  1e+00  2e-01
 4: -3.2106e+02 -3.4210e+02  2e+01  2e-01  4e-02
 5: -3.2845e+02 -3.3862e+02  1e+01  5e-02  9e-03
 6: -3.3342e+02 -3.3476e+02  1e+00  5e-03  8e-04
 7: -3.3424e+02 -3.3432e+02  8e-02  2e-04  4e-05
 8: -3.3429e+02 -3.3429e+02  1e-03  3e-06  5e-07
 9: -3.3429e+02 -3.3429e+02  1e-05  3e-08  5e-09
Optimal solution found.
19 support vectors from 140 points
Example Support Vector X is: (shape -> (19, 2) )
 [[1.06060606 0.81074876]
 [2.12121212 1.37166246]
 [3.48484848 1.00113266]
 [3.93939394 0.81074876]
 [3.03030303 0.90494396]]
Example Support Vector y is: (shape -> (19,) )
 [ 1.  1. -1.  1.  1.]
Example alpha: (shape -> (19,) )[18.47126197  5.43765551 50.5036981  24.95118545
17.59097043]
w: None
b: -1.1859329053398042
```

After fitting model, we can predict y hat using predict method to get it

[9]:
```python
y_pred = model.predict(X_test)
```

To see what are the result of y hat, we can print it by following,

[10]:
```python
print(f'Output from prediction are: \n {y_pred}')
```

```
Output from prediction are:
 [ 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 -1 -1  1  1 -1 -1  1 -1  1  1]
```

We would see that the output of predicted y are -1 and 1 classes. This is similar to the thing we mentioned before that SVM will saparate classes from calculation based on the side of decision boundary which define +1 and -1 sides

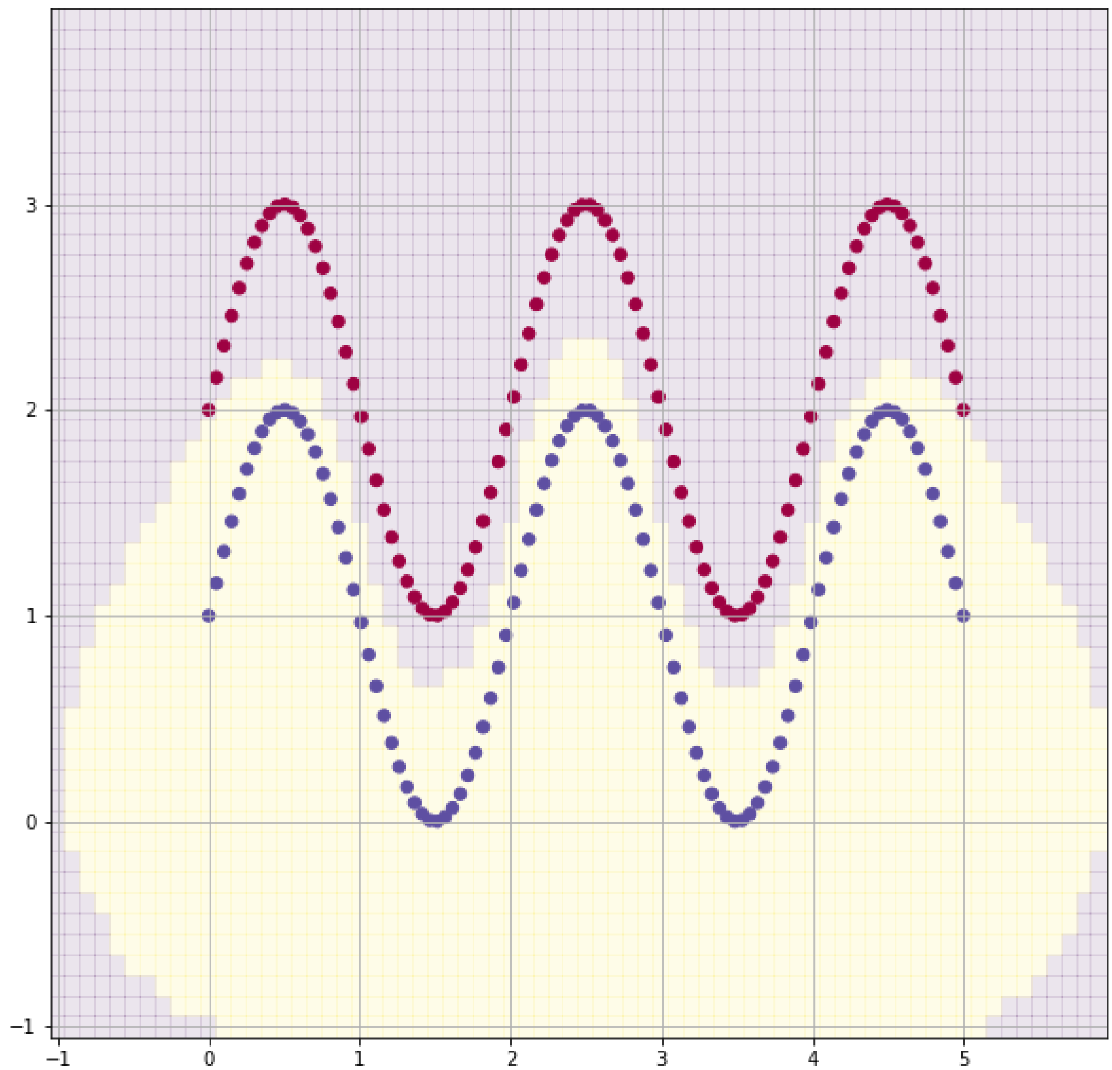
To see classification report, we simply implement a classification report tool form sklearn.metrics as

[11]:
```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

        -1.0       1.00      1.00      1.00        34
         1.0       1.00      1.00      1.00        26

    accuracy                           1.00        60
   macro avg       1.00      1.00      1.00        60
weighted avg       1.00      1.00      1.00        60
```

Moreover, SVM will be visually better if we illustrate it using plot and boundary side of the line graph. To do that, we already prepare this method in class to be used by following.

[12]:
```
model.plot_contour(X, y)
```

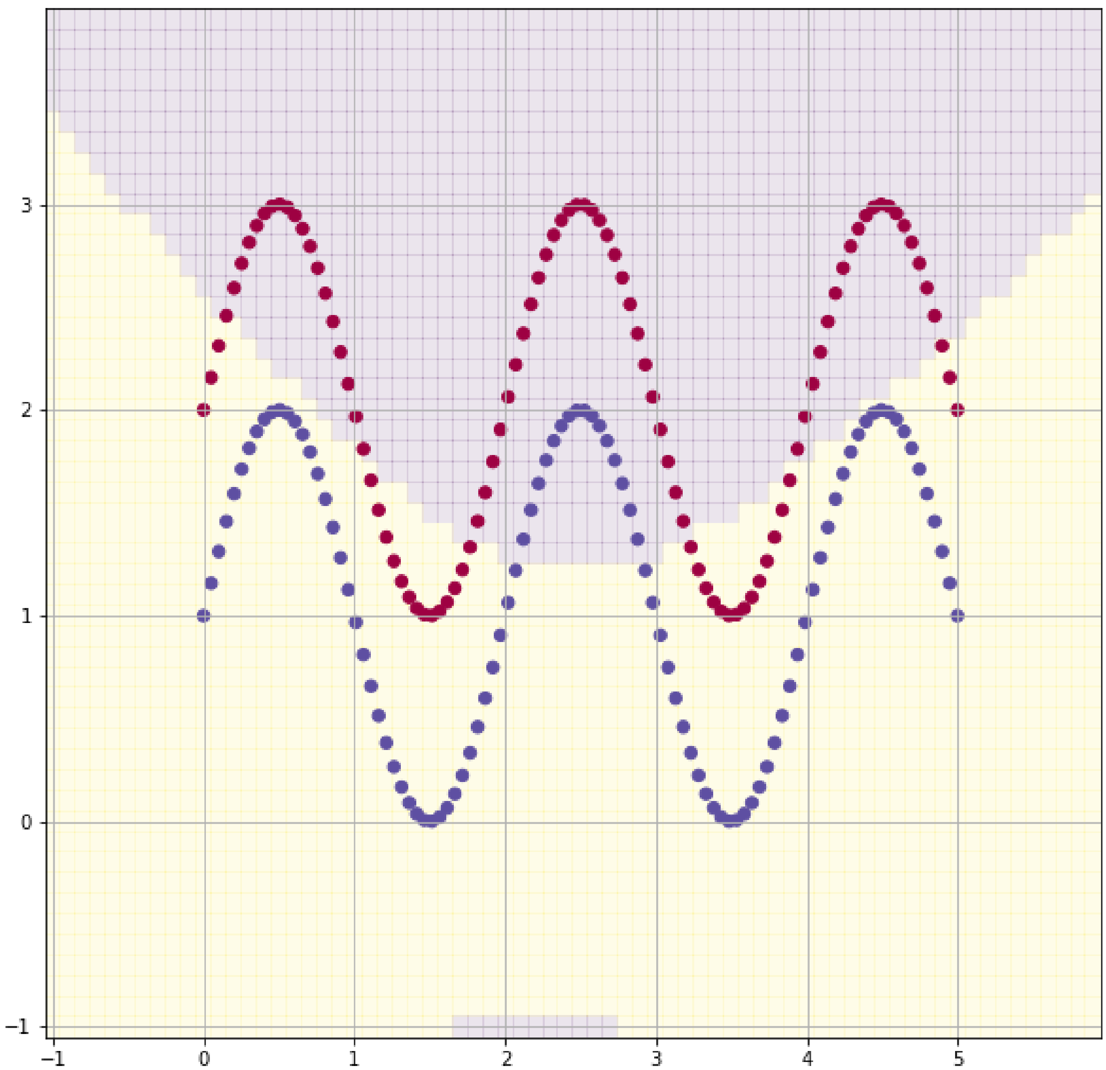


Lets try to perform the SVM model in some differently several ways First trying, Lets define kernel method as **'poly' to activate polynomial kernel**. As default, the polynomial degree initial as 2. Also, we use **Soft margin with 10** way to find decision boundary with classification

[13]:
```
model_poly = SVM(kernel='poly', C=10)
model_poly.fit(X_train, y_train)
y_pred_poly = model_poly.predict(X_test)
print(classification_report(y_test, y_pred_poly))
model_poly.plot_contour(X, y)
```

```
     pcost       dcost       gap    pres   dres
 0: -4.5353e+02 -9.2422e+03  2e+04  5e-01  3e-12
 1: -4.6310e+02 -2.1002e+03  2e+03  2e-15  1e-12
 2: -5.6173e+02 -9.1774e+02  4e+02  5e-15  2e-12
```

```
 3: -6.2705e+02 -8.6337e+02  2e+02  1e-14  2e-12
 4: -6.5651e+02 -8.1887e+02  2e+02  2e-14  2e-12
 5: -6.9183e+02 -7.6715e+02  8e+01  7e-15  2e-12
 6: -7.0684e+02 -7.4597e+02  4e+01  1e-14  3e-12
 7: -7.1464e+02 -7.3333e+02  2e+01  2e-14  3e-12
 8: -7.2070e+02 -7.2604e+02  5e+00  2e-15  2e-12
 9: -7.2188e+02 -7.2444e+02  3e+00  2e-15  3e-12
10: -7.2284e+02 -7.2325e+02  4e-01  1e-14  3e-12
11: -7.2303e+02 -7.2304e+02  5e-03  4e-14  4e-12
12: -7.2304e+02 -7.2304e+02  5e-05  1e-14  3e-12
Optimal solution found.
75 support vectors from 140 points
              precision    recall  f1-score   support

        -1.0       0.88      0.62      0.72        34
         1.0       0.64      0.88      0.74        26

    accuracy                           0.73        60
   macro avg       0.76      0.75      0.73        60
weighted avg       0.77      0.73      0.73        60
```



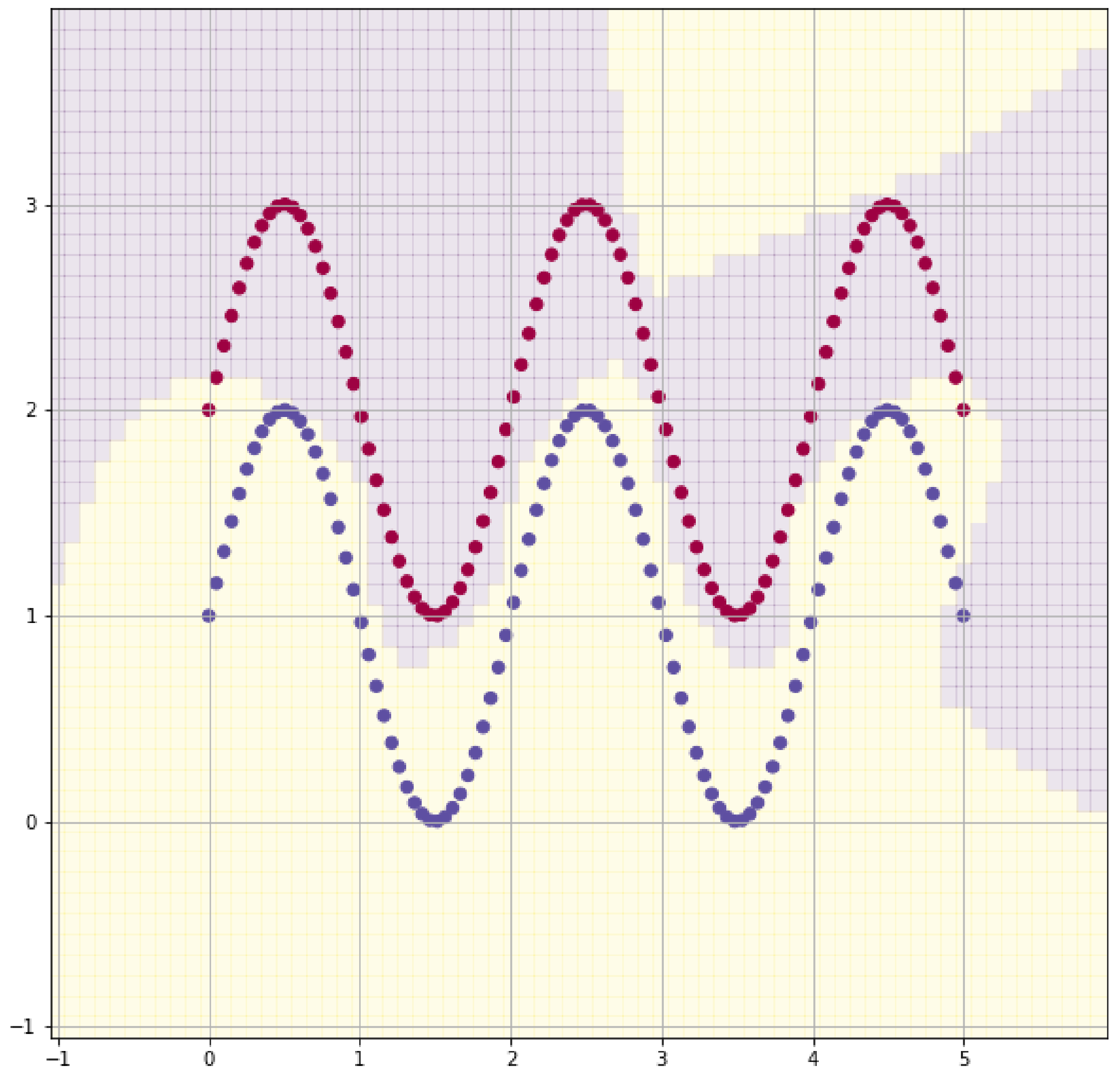
However, The result showed that it is simply curvy boundary to saparate classes, this indicates that low polynomial degree is not suitable to be used

Another try, we rise **polynomial degree to 7 with same C** and lets find out the result

[14]:
```
model_poly_7 = SVM(kernel='poly', C=10, degree=7)
model_poly_7.fit(X_train, y_train)
y_pred_poly_7 = model_poly_7.predict(X_test)
print(classification_report(y_test, y_pred_poly_7))
model_poly_7.plot_contour(X, y)
```

```
     pcost       dcost       gap    pres   dres
 0: -7.6060e+01 -1.2447e+04  4e+04  1e+00  2e-05
 1:  1.5844e+02 -6.4877e+03  1e+04  2e-01  2e-05
 2:  1.3624e+02 -2.2421e+03  3e+03  5e-02  6e-06
```

```
 3:  4.5984e+01 -3.7870e+02  5e+02  5e-03  1e-06
 4:  1.0845e+01 -8.1863e+01  1e+02  8e-04  6e-07
 5:  9.7862e-01 -3.3356e+01  4e+01  2e-04  3e-07
 6: -2.5018e+00 -1.6619e+01  2e+01  8e-05  2e-07
 7: -4.1233e+00 -8.4148e+00  5e+00  2e-05  2e-07
 8: -4.5013e+00 -6.2338e+00  2e+00  6e-06  1e-07
 9: -4.6938e+00 -5.6514e+00  1e+00  2e-06  1e-07
10: -4.8185e+00 -5.1708e+00  4e-01  6e-07  1e-07
11: -4.8708e+00 -4.9964e+00  1e-01  8e-08  1e-07
12: -4.8970e+00 -4.9339e+00  4e-02  2e-08  1e-07
13: -4.9054e+00 -4.9156e+00  1e-02  6e-16  8e-08
14: -4.9090e+00 -4.9104e+00  1e-03  5e-16  9e-08
15: -4.9095e+00 -4.9096e+00  9e-05  7e-16  1e-07
16: -4.9096e+00 -4.9096e+00  9e-07  5e-16  1e-07
17: -4.9096e+00 -4.9096e+00  9e-09  8e-16  1e-07
18: -4.9096e+00 -4.9096e+00  9e-11  5e-16  1e-07
19: -4.9096e+00 -4.9096e+00  9e-13  7e-16  1e-07
20: -4.9096e+00 -4.9096e+00  9e-15  1e-15  1e-07
21: -4.9096e+00 -4.9096e+00  9e-17  9e-16  1e-07
Optimal solution found.
18 support vectors from 140 points
              precision    recall  f1-score   support

        -1.0       0.97      0.94      0.96        34
         1.0       0.93      0.96      0.94        26

    accuracy                           0.95        60
   macro avg       0.95      0.95      0.95        60
weighted avg       0.95      0.95      0.95        60
```



Although the classisication report showed satisfying result, the boundary looks unsatisfying.

However, I am not yet enough to play with the model. I will try on another lesson, lets define **polynomial degree 5 and Hard margin** activated

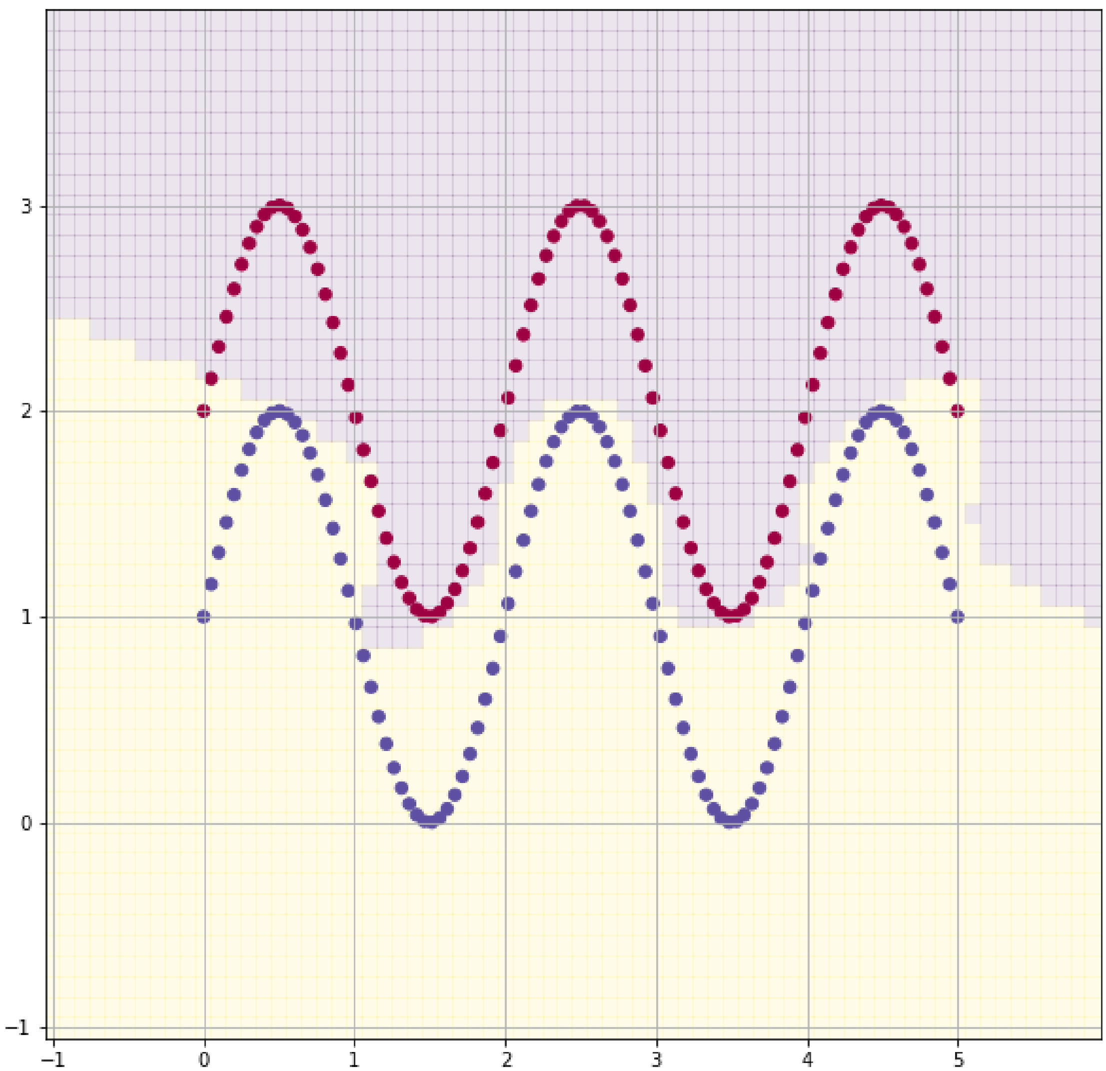
[15]:
```
model_poly_7 = SVM(kernel='poly', C=None, degree=5)
model_poly_7.fit(X_train, y_train)
y_pred_poly_7 = model_poly_7.predict(X_test)
print(classification_report(y_test, y_pred_poly_7))
model_poly_7.plot_contour(X, y)
```

```
     pcost       dcost       gap    pres   dres
 0: -7.3073e+01 -1.6676e+02  5e+02  2e+01  2e+00
 1: -2.4510e+02 -3.5017e+02  3e+02  1e+01  1e+00
 2: -3.9135e+02 -5.3211e+02  3e+02  1e+01  1e+00
```

```
 3: -1.2318e+03 -1.4203e+03  4e+02  9e+00  1e+00
 4: -1.7126e+03 -1.9497e+03  4e+02  9e+00  1e+00
 5: -3.3006e+03 -3.6981e+03  7e+02  9e+00  1e+00
 6: -4.8779e+03 -5.4682e+03  1e+03  8e+00  1e+00
 7: -6.7914e+03 -7.7771e+03  2e+03  8e+00  9e-01
 8: -7.1379e+03 -8.1993e+03  2e+03  7e+00  8e-01
 9: -7.1962e+03 -8.2749e+03  2e+03  7e+00  8e-01
10: -7.3268e+03 -8.4337e+03  2e+03  7e+00  8e-01
11: -7.7183e+03 -8.9204e+03  3e+03  6e+00  8e-01
12: -8.3245e+03 -9.6598e+03  3e+03  6e+00  7e-01
13: -9.5225e+03 -1.0889e+04  3e+03  4e+00  5e-01
14: -1.0285e+04 -1.1464e+04  3e+03  3e+00  4e-01
15: -1.0739e+04 -1.1537e+04  2e+03  2e+00  3e-01
16: -1.0755e+04 -1.1255e+04  2e+03  1e+00  2e-01
17: -1.0333e+04 -1.0931e+04  1e+03  7e-01  8e-02
18: -1.0378e+04 -1.0602e+04  5e+02  2e-01  3e-02
19: -1.0351e+04 -1.0469e+04  2e+02  5e-02  6e-03
20: -1.0397e+04 -1.0426e+04  4e+01  9e-03  1e-03
21: -1.0412e+04 -1.0417e+04  6e+00  1e-03  1e-04
22: -1.0415e+04 -1.0415e+04  3e-01  1e-05  2e-06
23: -1.0415e+04 -1.0415e+04  3e-03  2e-07  2e-06
24: -1.0415e+04 -1.0415e+04  3e-05  2e-09  3e-06
25: -1.0415e+04 -1.0415e+04  3e-07  2e-11  2e-06
26: -1.0415e+04 -1.0415e+04  3e-09  4e-12  2e-06
27: -1.0415e+04 -1.0415e+04  3e-11  1e-12  2e-06
28: -1.0415e+04 -1.0415e+04  3e-13  1e-12  1e-06
29: -1.0415e+04 -1.0415e+04  3e-15  4e-12  1e-06
30: -1.0415e+04 -1.0415e+04  3e-17  2e-12  1e-06
31: -1.0415e+04 -1.0415e+04  3e-19  1e-12  1e-06
32: -1.0415e+04 -1.0415e+04  3e-21  1e-12  1e-06
33: -1.0415e+04 -1.0415e+04  3e-23  2e-12  1e-06
34: -1.0415e+04 -1.0415e+04  3e-25  3e-12  2e-06
35: -1.0415e+04 -1.0415e+04  3e-27  2e-12  1e-06
36: -1.0415e+04 -1.0415e+04  3e-29  2e-12  1e-06
37: -1.0415e+04 -1.0415e+04  3e-31  2e-12  1e-06
38: -1.0415e+04 -1.0415e+04  3e-33  3e-12  1e-06
39: -1.0415e+04 -1.0415e+04  3e-35  1e-12  1e-06
40: -1.0415e+04 -1.0415e+04  3e-37  2e-12  9e-07
41: -1.0415e+04 -1.0415e+04  3e-39  1e-12  2e-06
42: -1.0415e+04 -1.0415e+04  3e-41  2e-12  1e-06
43: -1.0415e+04 -1.0415e+04  3e-43  1e-12  1e-06
44: -1.0415e+04 -1.0415e+04  3e-45  1e-12  1e-06
45: -1.0415e+04 -1.0415e+04  3e-47  1e-12  1e-06
46: -1.0415e+04 -1.0415e+04  3e-49  1e-12  1e-06
47: -1.0415e+04 -1.0415e+04  3e-51  1e-12  1e-06
48: -1.0415e+04 -1.0415e+04  3e-53  2e-12  8e-07
49: -1.0415e+04 -1.0415e+04  3e-55  1e-12  1e-06
50: -1.0415e+04 -1.0415e+04  3e-57  2e-12  1e-06
```

```
51: -1.0415e+04 -1.0415e+04  3e-59  1e-12  2e-06
52: -1.0415e+04 -1.0415e+04  3e-61  1e-12  1e-06
53: -1.0415e+04 -1.0415e+04  3e-63  2e-12  1e-06
54: -1.0415e+04 -1.0415e+04  3e-65  3e-12  1e-06
55: -1.0415e+04 -1.0415e+04  3e-67  2e-12  1e-06
56: -1.0415e+04 -1.0415e+04  3e-69  1e-12  1e-06
57: -1.0415e+04 -1.0415e+04  3e-71  2e-12  1e-06
58: -1.0415e+04 -1.0415e+04  3e-73  2e-12  1e-06
59: -1.0415e+04 -1.0415e+04  3e-75  2e-12  1e-06
60: -1.0415e+04 -1.0415e+04  3e-77  1e-12  9e-07
61: -1.0415e+04 -1.0415e+04  3e-79  1e-12  1e-06
62: -1.0415e+04 -1.0415e+04  3e-81  2e-12  1e-06
63: -1.0415e+04 -1.0415e+04  3e-83  1e-12  9e-07
64: -1.0415e+04 -1.0415e+04  3e-85  3e-12  9e-07
65: -1.0415e+04 -1.0415e+04  3e-87  2e-12  1e-06
66: -1.0415e+04 -1.0415e+04  3e-89  2e-12  1e-06
67: -1.0415e+04 -1.0415e+04  3e-91  1e-12  2e-06
68: -1.0415e+04 -1.0415e+04  3e-93  3e-12  1e-06
69: -1.0415e+04 -1.0415e+04  3e-95  3e-12  1e-06
70: -1.0415e+04 -1.0415e+04  3e-97  3e-12  1e-06
71: -1.0415e+04 -1.0415e+04  3e-99  1e-12  1e-06
72: -1.0415e+04 -1.0415e+04  3e-101  2e-12  1e-06
73: -1.0415e+04 -1.0415e+04  3e-103  2e-12  1e-06
74: -1.0415e+04 -1.0415e+04  3e-105  3e-12  1e-06
75: -1.0415e+04 -1.0415e+04  3e-107  1e-12  2e-06
76: -1.0415e+04 -1.0415e+04  3e-109  2e-12  2e-06
77: -1.0415e+04 -1.0415e+04  3e-111  2e-12  1e-06
78: -1.0415e+04 -1.0415e+04  3e-113  2e-12  1e-06
79: -1.0415e+04 -1.0415e+04  3e-115  1e-12  1e-06
80: -1.0415e+04 -1.0415e+04  3e-117  2e-12  1e-06
81: -1.0415e+04 -1.0415e+04  3e-119  1e-12  1e-06
82: -1.0415e+04 -1.0415e+04  3e-121  2e-12  1e-06
83: -1.0415e+04 -1.0415e+04  3e-123  2e-12  1e-06
84: -1.0415e+04 -1.0415e+04  3e-125  1e-12  1e-06
85: -1.0415e+04 -1.0415e+04  3e-127  1e-12  1e-06
86: -1.0415e+04 -1.0415e+04  3e-129  3e-12  1e-06
87: -1.0415e+04 -1.0415e+04  3e-131  2e-12  2e-06
88: -1.0415e+04 -1.0415e+04  3e-133  2e-12  1e-06
89: -1.0415e+04 -1.0415e+04  3e-135  2e-12  2e-06
90: -1.0415e+04 -1.0415e+04  3e-137  2e-12  2e-06
91: -1.0415e+04 -1.0415e+04  3e-139  2e-12  1e-06
92: -1.0415e+04 -1.0415e+04  3e-141  2e-12  2e-06
93: -1.0415e+04 -1.0415e+04  3e-143  2e-12  1e-06
94: -1.0415e+04 -1.0415e+04  3e-145  1e-12  1e-06
95: -1.0415e+04 -1.0415e+04  3e-147  2e-12  2e-06
96: -1.0415e+04 -1.0415e+04  3e-149  1e-12  2e-06
97: -1.0415e+04 -1.0415e+04  3e-151  1e-12  2e-06
98: -1.0415e+04 -1.0415e+04  3e-153  3e-12  1e-06
```

```
99: -1.0415e+04 -1.0415e+04  3e-155  1e-12  1e-06
100: -1.0415e+04 -1.0415e+04  3e-157  2e-12  1e-06
Terminated (maximum number of iterations reached).
18 support vectors from 140 points
              precision    recall  f1-score   support

        -1.0       1.00      0.88      0.94        34
         1.0       0.87      1.00      0.93        26

    accuracy                           0.93        60
   macro avg       0.93      0.94      0.93        60
weighted avg       0.94      0.93      0.93        60
```

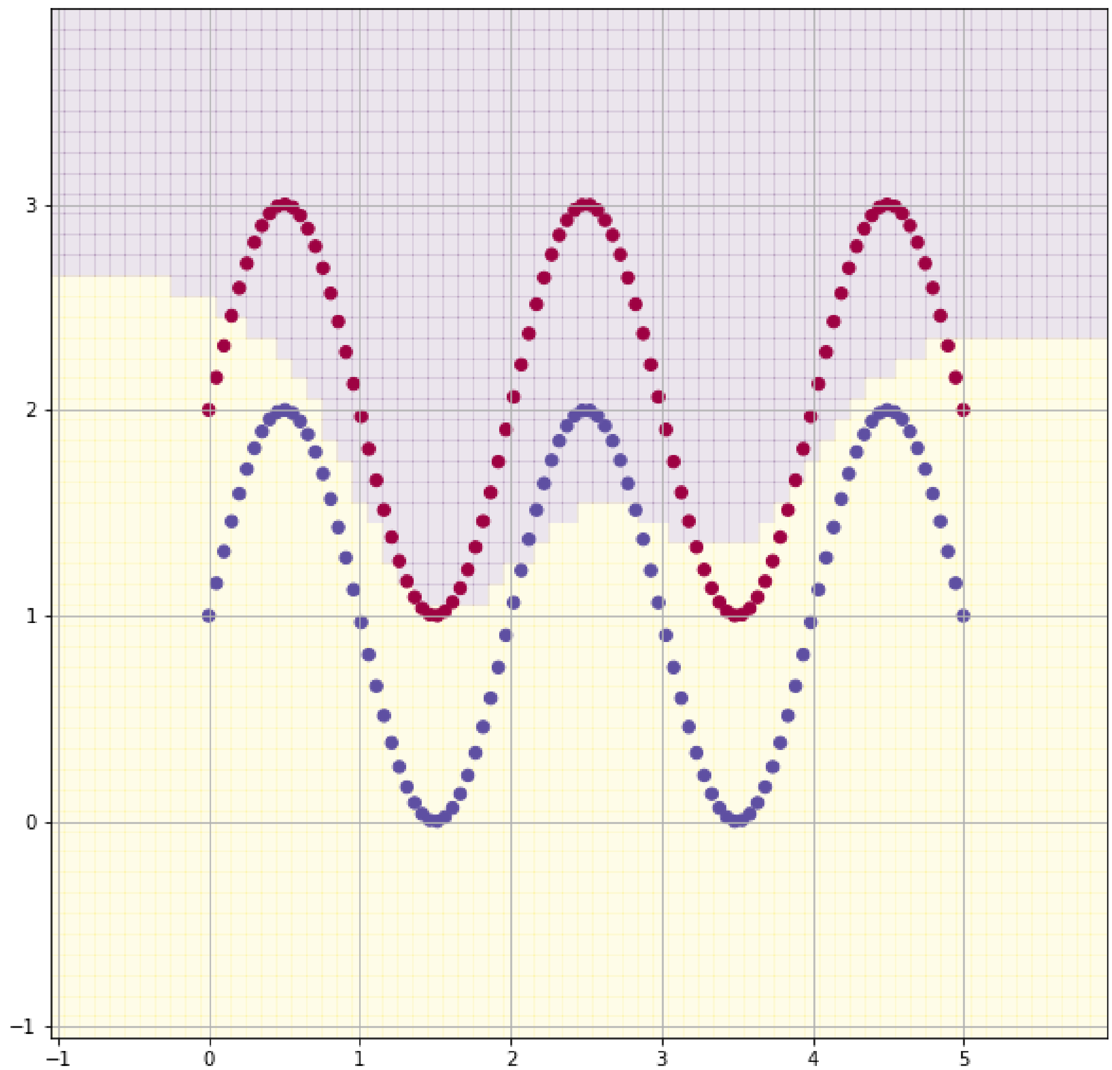


The result may show better even similar to soft margin one.

Additionally, in first try, we did classify with **hard margin of gaussian kernel**. This try is to be **soft margin with C=1** activated to see how different the result with previous one.

[16]:
```
model_gs = SVM(kernel='gs', C=1)
model_gs.fit(X_train, y_train)
y_pred_gs = model_gs.predict(X_test)
print(classification_report(y_test, y_pred_gs))
model_gs.plot_contour(X, y)
```

```
     pcost       dcost       gap    pres   dres
 0: -6.6844e+01 -2.8493e+02  9e+02  2e+00  1e-15
 1: -5.3932e+01 -1.7309e+02  1e+02  7e-16  7e-16
 2: -6.3732e+01 -8.2544e+01  2e+01  4e-16  6e-16
 3: -6.9286e+01 -7.4528e+01  5e+00  2e-15  7e-16
 4: -7.0811e+01 -7.2298e+01  1e+00  4e-15  6e-16
 5: -7.1368e+01 -7.1597e+01  2e-01  3e-15  7e-16
 6: -7.1450e+01 -7.1502e+01  5e-02  9e-16  6e-16
 7: -7.1472e+01 -7.1477e+01  5e-03  1e-15  7e-16
 8: -7.1474e+01 -7.1474e+01  9e-04  4e-15  7e-16
 9: -7.1474e+01 -7.1474e+01  2e-05  2e-15  8e-16
Optimal solution found.
93 support vectors from 140 points
              precision    recall  f1-score   support

        -1.0       0.88      0.68      0.77        34
         1.0       0.68      0.88      0.77        26

    accuracy                           0.77        60
   macro avg       0.78      0.78      0.77        60
weighted avg       0.79      0.77      0.77        60
```



The result show boundary line seems to be more **smooth rather than hard margin**.

1.5 Conclusion

The SVM classifier behaves classification in different way than other model works. The pronciple of it is to construct the line that saparate binary classes data from each other. The training and result labeled target class as positive or negative {+1, -1}. This indicate that the sample is in which side of the line.

Moreover, straight line is inaduquate to saparate in term of non-linear data form. it needs kernel tricks to transform the X in to some thing like K(X,X') to project it in different dimension form berfore putting to train model.

Furthermore, forcing samples belong to a class may cause outliers being wrong position, and decision boundary will be wrong. So that, soft margin technique a part of hard margin approach will be the one to help defining decision line allowing some sample in another side.

To summarized, SVM is another supervised classifier to make binary classification that contains dramatic mathematic behind. However, to use this model, we should consider suitability for each dataset that will be availability for implmenting this model.

```
[ ]:
```