# SVM - Lab

September 4, 2021

Your work: - Load this dataset to numpy, 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]:

[13]:
```
import numpy as np
import matplotlib.pyplot as plt
import sys
from numpy import linalg
import cvxopt
import cvxopt.solvers
from sklearn.model_selection import train_test_split

dataset_numpy = np.array(dataset)
dataset_numpy.shape
```

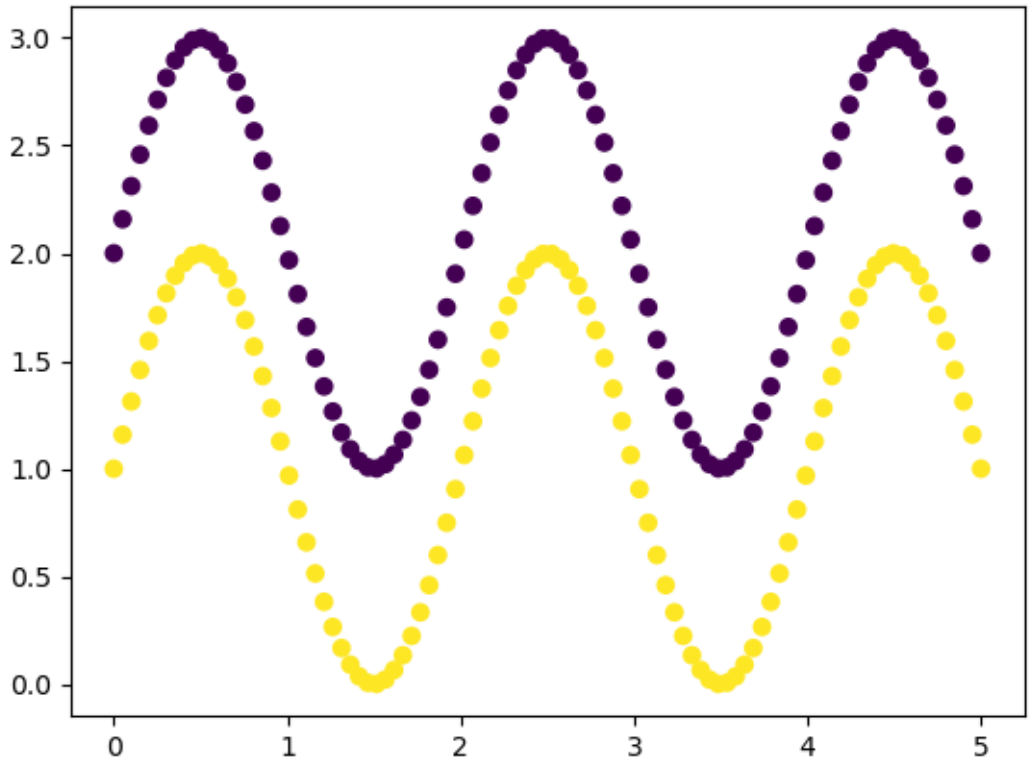
[13]: (200, 3)

## 1 Load this dataset to numpy, with first two columns as features and last as target

[15]:
```
X_train, X_test, y_train, y_test = train_test_split(dataset_numpy[:,[0,1]],␣
↪dataset_numpy[:,2], test_size=0.20, random_state=42)

y_train[y_train==0] = -1
y_test[y_test==0] = -1
```

## 2 Plot the data using a scatter plot

[14]:
```
plt.scatter(dataset_numpy[:,0], dataset_numpy[:,1], c=dataset_numpy[:,2])
plt.show()
```




## 3 Perform the SVM classification using our scratch code

[33]:
```
class SVM:
    def linear(self,x1, x2):
        return np.dot(x1, x2)

    def polynomial(self,x, y, p=2):
        return (1 + np.dot(x, y)) ** p

    def gaussian(self,x, y, sigma=0.9999):
        return np.exp(-linalg.norm(x-y)**2 / (2 * (sigma ** 2)))

    def fit(self, X, y, kernel, C):

        n_samples, n_features = X.shape

        # Gram matrix
        # initialize kernel matrix
        K = np.zeros((n_samples, n_samples))
        # Kernel matrix
```

```
        for i in range(n_samples):
            for j in range(n_samples):
                if kernel == 'linear':
                    K[i,j] = self.linear(X[i], X[j])
                elif kernel == 'gaussian':
                    K[i,j] = self.gaussian(X[i], X[j])
                elif kernel == 'polynomial':
                    K[i,j] = self.polynomial(X[i], X[j])
                else:
                    print('kernel must be one of ["linear", "gaussian",␣
↪"polynomial"]')
                    sys.exit()

        P = cvxopt.matrix(np.outer(y, y) * K)
        q = cvxopt.matrix(np.ones(n_samples) * -1)
        A = cvxopt.matrix(y, (1,n_samples))
        b = cvxopt.matrix(0.0)

        if C is None:
            G = cvxopt.matrix(np.diag(np.ones(n_samples) * -1))
            h = cvxopt.matrix(np.zeros(n_samples))
        else:
            tmp1 = np.diag(np.ones(n_samples) * -1)
            tmp2 = np.identity(n_samples)
            G = cvxopt.matrix(np.vstack((tmp1, tmp2)))
            tmp1 = np.zeros(n_samples)
            tmp2 = np.ones(n_samples) * C
            h = cvxopt.matrix(np.hstack((tmp1, tmp2)))

        solution = cvxopt.solvers.qp(P, q, G, h, A, b)

        a = np.ravel(solution['x'])

        sv_idx = a > 1e-5
        ind = np.arange(len(a))[sv_idx]
        a = a[sv_idx]
        sv = X[sv_idx]
        sv_y = y[sv_idx]
        print("%d support vectors out of %d points" % (len(a), n_samples))

        b = 0
        for n in range(len(a)):
            b += sv_y[n]
            b -= np.sum(a * sv_y * K[ind[n],sv_idx])
        b /= len(a)

        if kernel == 'linear':
```

```
            w = np.zeros(n_features)
            for n in range(len(a)):
                w += a[n] * sv_y[n] * sv[n]
        else:
            w = None
        return sv, sv_y, a, w, b


    def project(self,X, kernel,sv, sv_y, a, w, b):
        if w is not None:
            return np.dot(X, w) + b
        else:
            y_predict = np.zeros(len(X))
            for i in range(len(X)):
                s = 0
                for a_val, sv_y_val, sv_val in zip(a, sv_y, sv):
                    if kernel == 'polynomial':
                        s += a_val * sv_y_val * self.polynomial(X[i], sv_val)
                    else:
                        s += a_val * sv_y_val * self.gaussian(X[i], sv_val)
                y_predict[i] = s
            return y_predict + b


    def predict(self,X, kernel, sv, sv_y, a, w, b):
        return np.sign(self.project(X, kernel,sv, sv_y, a, w, b))

    def plot_contour(self,X1_train, X2_train, kernel, sv, sv_y, a, w, b):
        pl.plot(X1_train[:,0], X1_train[:,1], "ro")
        pl.plot(X2_train[:,0], X2_train[:,1], "bo")
        pl.scatter(sv[:,0], sv[:,1], s=100, c="g")
        # here we choose the range between -7 and 7 as we have choosen
        # the mean to be between -4 and 4 while generating data with the␣
↪variance of 0.8
        X1, X2 = np.meshgrid(np.linspace(-7,7,50), np.linspace(-7,7,50))
        X = np.array([[x1, x2] for x1, x2 in zip(np.ravel(X1), np.ravel(X2))])
        Z = self.project(X, kernel,sv, sv_y, a, w, b).reshape(X1.shape)
        pl.contour(X1, X2, Z, [0.0], colors='k', linewidths=1, origin='lower')
        pl.contour(X1, X2, Z + 1, [0.0], colors='grey', linewidths=1,␣
↪origin='lower')
        pl.contour(X1, X2, Z - 1, [0.0], colors='grey', linewidths=1,␣
↪origin='lower')

        pl.axis("tight")
        pl.show()
```

[34]:
```
cls = SVM()
kernel = 'gaussian'
sv, sv_y, a, w, b = cls.fit(X_train,y_train,kernel=kernel,C=None)
```

```
     pcost       dcost       gap    pres   dres
 0: -6.9496e+01 -1.9281e+02  5e+02  1e+01  2e+00
 1: -1.7109e+02 -2.9223e+02  2e+02  6e+00  9e-01
 2: -2.7725e+02 -3.8134e+02  1e+02  2e+00  4e-01
 3: -3.1719e+02 -3.9262e+02  8e+01  5e-01  9e-02
 4: -3.3379e+02 -3.5527e+02  2e+01  1e-01  2e-02
 5: -3.4160e+02 -3.5062e+02  9e+00  4e-03  6e-04
 6: -3.4564e+02 -3.4767e+02  2e+00  7e-04  1e-04
 7: -3.4659e+02 -3.4706e+02  5e-01  2e-05  3e-06
 8: -3.4689e+02 -3.4699e+02  1e-01  4e-06  6e-07
 9: -3.4692e+02 -3.4698e+02  6e-02  2e-06  2e-07
10: -3.4697e+02 -3.4697e+02  2e-03  3e-08  5e-09
11: -3.4697e+02 -3.4697e+02  2e-05  3e-10  5e-11
Optimal solution found.
25 support vectors out of 160 points
```

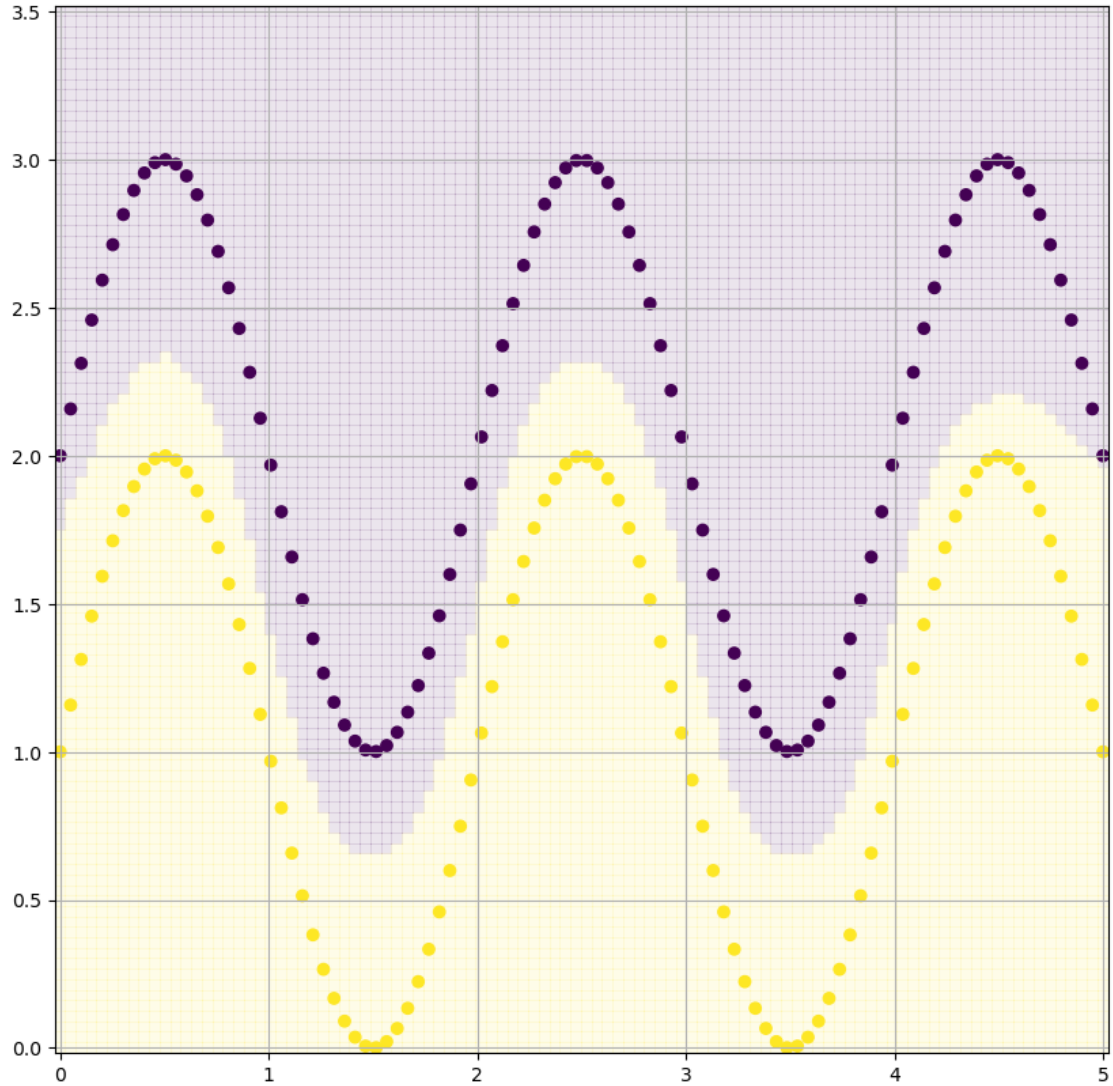
[35]:
```
y_pred = cls.predict(X_test,kernel,sv,sv_y,a,w,b).astype(int)

print("y_test:",y_test.astype(int))
print("y_pred:",y_pred)
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
y_test: [-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]
y_pred: [-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]
              precision    recall  f1-score   support

        -1.0       1.00      1.00      1.00        17
         1.0       1.00      1.00      1.00        23

    accuracy                           1.00        40
   macro avg       1.00      1.00      1.00        40
weighted avg       1.00      1.00      1.00        40
```

[37]:
```
resolution = 100
x_series = np.linspace(0,5,resolution)
y_series = np.linspace(0,3.5,resolution)

x_mesh,y_mesh = np.meshgrid(x_series,y_series)

x_mesh = x_mesh.reshape(-1,1)
```

```
y_mesh = y_mesh.reshape(-1,1)

mesh = np.append(x_mesh,y_mesh,axis=1)
y_pred = cls.predict(mesh,kernel,sv,sv_y,a,w,b).astype(int)

x_mesh = x_mesh.reshape(resolution,resolution)
y_mesh = y_mesh.reshape(resolution,resolution)
y_pred = y_pred.reshape(resolution,resolution)
```

[42]:
```
plt.figure(figsize=(10,10))
plt.scatter(dataset_numpy[:,0],dataset_numpy[:,1],c=dataset_numpy[:,2])
plt.pcolormesh(x_mesh,y_mesh,y_pred,cmap='viridis',shading='auto',alpha=0.1)
plt.grid(True)
plt.show()
```



[ ]: