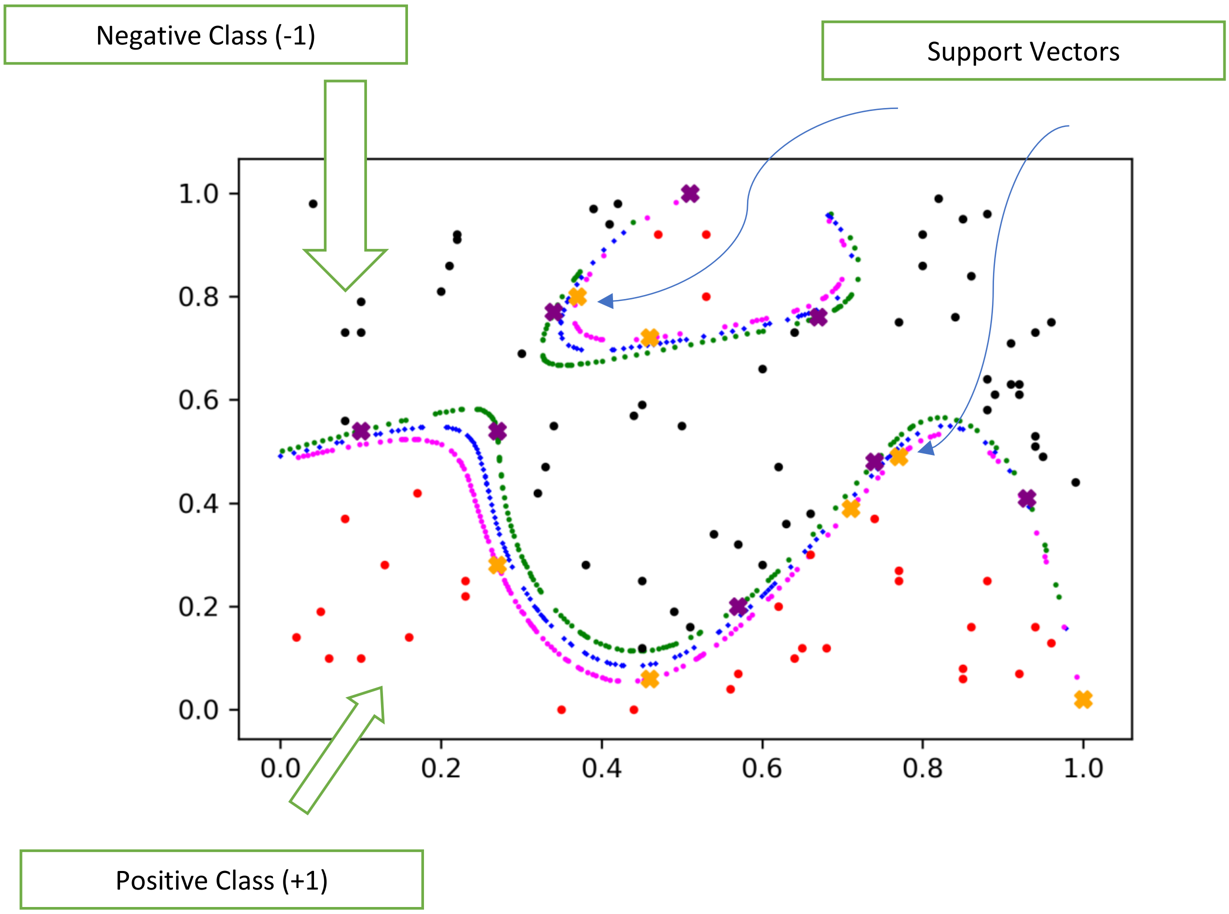
SUPPORT VECTOR MACHINES



The Three Hyperplanes

1.Blue $\mathcal{H} \triangleq \{\mathbf{x} : g(\mathbf{x}) = 0\}$

2.Green $\mathcal{H}^{+} \triangleq \{\mathbf{x} : g(\mathbf{x}) = 1\}$

3.Magentia $\mathcal{H}^{-} \triangleq \{\mathbf{x} : g(\mathbf{x}) = -1\}.$

Code

```
# -*- coding: utf-8 -*-
"""
Created on Fri Apr 17 22:27:30 2020

@author: Varun
"""
#############LIBRARIES#######################
import numpy as np
import matplotlib.pyplot as plt
from cvxopt import matrix as cvxopt_matrix
from cvxopt import solvers as cvxopt_solvers
import cvxopt
###########INPUT#############################
NumSamples=100
n_samples=NumSamples
x = np.round(np.random.uniform(0,1,(NumSamples ,2)),2)
def conditioncheck(x1,x2):
    d=0
    if x2<((1/5)*np.sin(10*x1))+0.3 or (x2-0.8)**2+(x1-0.5)**2< (0.15**2):
        d=1
    else:
        d=-1
    return(d)
D=[conditioncheck(i[0],i[1]) for i in x]
DX=list(zip(D,x))
Xbak=x
ybak=D
#############separate classes############################
C=60000
D1X=[]
D2X=[]
for i in range(0,len(DX)):
    if DX[i][0]==1:
        D1X.append(DX[i])
    else:
        D2X.append(DX[i])
D1X=[list(i[1]) for i in D1X]
D2X=[list(i[1]) for i in D2X]
y1=[1 for i in range(0,len(D1X))]
y2=[-1 for i in range(0,len(D2X))]

X=D1X+D2X
y=y1+y2

Xbak=X
ybak=y

#######kernel definition################
def kernel(x, y, p=10):
    return (1 + np.matmul(x, y)) ** p
###########################################################
```

```
x1=[x[0] for x in D1X]
y1=[x[1] for x in D1X]

x2=[x[0] for x in D2X]
y2=[x[1] for x in D2X]

plt.scatter(x1, y1,color='blue')
plt.scatter(x2, y2,color='green')
plt.title('Data Points')

##################################################
X=np.matrix(X)
y = np.asarray(y).astype(float)
n_samples, n_features = X.shape
K = np.zeros((n_samples, n_samples))
for i in range(n_samples):
    for j in range(n_samples):
        K[i,j] = kernel(X[i], X[j].transpose())

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'])
abak=a
svx = a > 1e-5
ind = np.arange(len(a))[svx]
len(ind)

a = a[svx]
sv_x = X[svx]
sv_y = y[svx]
print("%d support vectors out of %d points" % (len(a), n_samples))

meantheta=[]
for j in range(0,len(sv_x)):
    sumofalli=0
    for i in range(n_samples):
            sumofalli=sumofalli+(abak[i]*y[i]*kernel(X[i], sv_x[j].transpose()))
```

```
    theta=sv_y[j]-sumofalli
    meantheta.append(theta)

theta=np.mean(meantheta)
print(theta)
##############found theta##################
##someplots
x1=[x[0] for x in D1X]
y1=[x[1] for x in D1X]
x2=[x[0] for x in D2X]
y2=[x[1] for x in D2X]
plt.scatter(x1, y1,color='red')
plt.scatter(x2, y2,color='yellow')

i=0
newl=[]
for i in range(0,len(sv_y)):
    if np.array(sv_y)[i]==1:
        newl.append(np.array(sv_x[i]))

SV1=[i[0] for i in np.array(newl)]
sv1=[i.item(0) for i in SV1]
sv2=[i.item(1) for i in SV1]
plt.scatter(sv1,sv2, c = 'blue')

i=0
newl=[]
for i in range(0,len(sv_y)):
    if np.array(sv_y)[i]==-1:
        newl.append(np.array(sv_x[i]))

SV1=[i[0] for i in np.array(newl)]
sv3=[i.item(0) for i in SV1]
sv4=[i.item(1) for i in SV1]
plt.scatter(sv3,sv4, c = 'blue')




################hyperplane plotter####################################
x_coord = np.linspace(0.0, 1.0, num=500)
y_coord = np.linspace(0.0, 1.0, num=500)

hmain=[]
hminus=[]
hplus=[]

for i in range(len(x_coord)):
    for j in range(len(y_coord)):
        descriminant = theta
        for k in range(len(sv_x)):
            descriminant += a[k]*sv_y[k]*kernel(sv_x[k], np.asarray([x_coord[i], y_coord[j]]))
```

```
        if -0.01 <descriminant<0.01:
            hmain.append([x_coord[i], y_coord[j]])
        if 0.99 <descriminant<1.01:
            hplus.append([x_coord[i], y_coord[j]])
        if -1.01 <descriminant<-0.99:
            hminus.append([x_coord[i], y_coord[j]])

xx1=[x[0] for x in hminus]
yy1=[x[1] for x in hminus]
plt.scatter(xx1,yy1, c = 'green',marker='o',s=1)

xx3=[x[0] for x in hmain]
yy3=[x[1] for x in hmain]
plt.scatter(xx3,yy3, c = 'blue',marker='x',s=1)

xx2=[x[0] for x in hplus]
yy2=[x[1] for x in hplus]
plt.scatter(xx2,yy2, c = 'magenta',marker='o',s=1)


plt.scatter(x1, y1,color='red',s=5)
plt.scatter(x2, y2,color='black',s=5)

plt.scatter(sv3,sv4, c = 'purple',marker='X')
plt.scatter(sv1,sv2, c = 'orange',marker='X')
plt.savefig('filename.png', dpi=300)

import os
path=r'C:\Users\Varun\Desktop\Studies'
os.chdir(path)
len(sv3)
len(sv1)
```