CS559 – Neural Networks
Assignment 8
By Chandrasekhara Ganesh Jagadeesan
UIN: 660647011

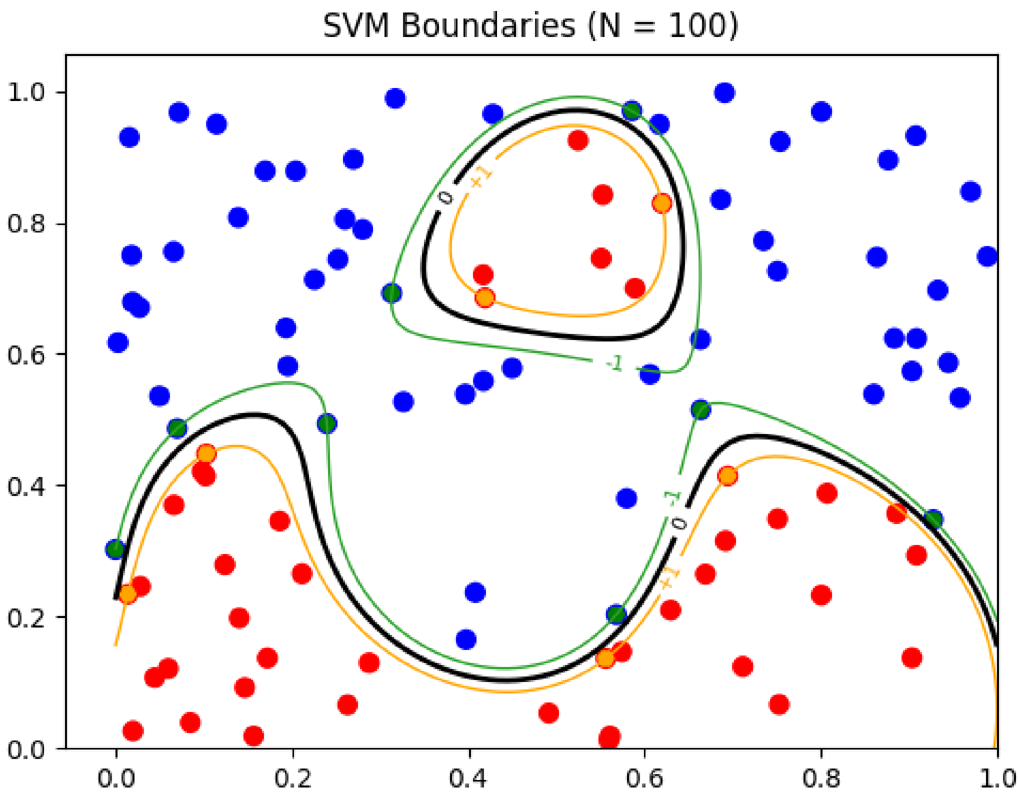
Description of the data set randomly chosen based on given target function:

- **In-set accuracy by maximum class classifier:** 57.0%
- **In-set accuracy using SVM classifier:** 100%

Number of SVMs computed: 14
Number of SVMs in –ve class: 8
Number of SVMs in +ve class: 6

Kernel Used: Radial Basis Function, $K(x, x') = \exp(-\gamma||x - x'||^2)$

Value of Gamma used: 3 (Higher value of Gammas will lead to more SVMs being generated and hence would lead to overfitting). I chose gamma by putting in many values and checking which gave the least support vectors and still completely classified the training data properly.



Source Code:

```
#svm.py

import numpy as np
import cvxopt
import cvxopt.solvers

cvxopt.solvers.options['show_progress'] = False

class SVM():
	def __init__(self,kernel="rbf",polyconst=1,gamma=10,degree=2):
		self.kernel = kernel
		self.polyconst = float(1)
		self.gamma = float(gamma)
		self.degree = degree
		self.kf = {
			"linear":self.linear,
			"rbf":self.rbf,
			"poly":self.polynomial
		}
		self._support_vectors = None
		self._alphas = None
		self.intercept = None
		self._n_support = None
		self.weights = None
		self._support_labels = None
		self._indices = None

	def linear(self,x,y):
		return np.dot(x.T,y)

	def polynomial(self,x,y):
		return (np.dot(x.T,y) + self.polyconst)**self.degree

	def rbf(self,x,y):
		return np.exp(-1.0*self.gamma*np.dot(np.subtract(x,y).T,np.subtract(x,y)))

	def transform(self,X):
		K = np.zeros([X.shape[0],X.shape[0]])
		for i in range(X.shape[0]):
			for j in range(X.shape[0]):
				K[i,j] = self.kf[self.kernel](X[i],X[j])
		return K

	def fit(self,data,labels):
		num_data, num_features = data.shape
		labels = labels.astype(np.double)
```

```
		K = self.transform(data)
		P = cvxopt.matrix(np.outer(labels,labels)*K)
		q = cvxopt.matrix(np.ones(num_data)*-1)
		A = cvxopt.matrix(labels,(1,num_data))
		b = cvxopt.matrix(0.0)
		G = cvxopt.matrix(np.diag(np.ones(num_data) * -1))
		h = cvxopt.matrix(np.zeros(num_data))

		alphas = np.ravel(cvxopt.solvers.qp(P, q, G, h, A, b)['x'])
		is_sv = alphas>1e-5
		self._support_vectors = data[is_sv]
		self._n_support = np.sum(is_sv)
		self._alphas = alphas[is_sv]
		self._support_labels = labels[is_sv]
		self._indices = np.arange(num_data)[is_sv]
		self.intercept = 0
		for i in range(self._alphas.shape[0]):
			self.intercept += self._support_labels[i]
			self.intercept -= np.sum(self._alphas*self._support_labels*K[self._indices[i],is_sv])
		self.intercept /= self._alphas.shape[0]
		self.weights = np.sum(data*labels.reshape(num_data,1)*self._alphas.reshape(num_data,1),axis=0,keepdims=True) if self.kernel == "linear" else None

	def signum(self,X):
		return np.where(X>0,1,-1)

	def project(self,X):
		if self.kernel=="linear":
			score = np.dot(X,self.weights)+self.intercept
		else:
			score = np.zeros(X.shape[0])
			for i in range(X.shape[0]):
				s = 0
				for alpha,label,sv in zip(self._alphas,self._support_labels,self._support_vectors):
					s += alpha*label*self.kf[self.kernel](X[i],sv)
				score[i] = s
			score = score + self.intercept
		return score

	def predict(self,X):
		return self.signum(self.project(X))
```

```
#question1.py
import numpy as np
import matplotlib.pyplot as plt
from svm import SVM
np.random.seed(1)
def get_data(lower,upper,num,num_dims):
    return np.random.uniform(lower,upper,size=(num,num_dims))
def get_labels(X):
    Y = []
    for x1,x2 in X:
        if x2 < np.sin(10*x1)/5 + 0.3 or ((x2 - 0.8)**2 + (x1 - 0.5)**2)<0.15**2:
            Y.append(1)
        else:
            Y.append(-1)
    return np.asarray(Y)

def main():
    N = 100
    data = get_data(0,1,N,2)
    labels = get_labels(data).reshape(-1)
    predictions = np.ones_like(labels)*-1
    print("Max-class classifier training set accuracy:
",np.mean(np.equal(predictions,labels))*100,"%")
    model = SVM(kernel="rbf",gamma=3)
    model.fit(data,labels)
    predictions = model.predict(data)
    print("SVM model Training set accuracy:
",np.mean(np.equal(predictions,labels))*100,"%")
    print("Number of SVMs computed: ",model._n_support)
    color = np.where(model._support_labels==1,"orange","green")
    plt.scatter(data[:, 0], data[:, 1], c=labels, s=50, cmap=plt.cm.bwr)
    plt.scatter(model._support_vectors[:, 0], model._support_vectors[:, 1], s=35,
c=color, marker='H')
    plt.title('SVM Boundaries (N = %d)' % (N))
    X1, X2 = np.meshgrid(np.linspace(0, 1, 100), np.linspace(0, 1, 100))
    X_T = np.array([[x1, x2] for x1, x2 in zip(np.ravel(X1), np.ravel(X2))])
    Z = model.project(X_T).reshape(X1.shape)
    H = plt.contour(X1, X2, Z, [0.0], colors='k', linewidths=2, origin='lower')
    H_1= plt.contour(X1, X2, Z + 1, [0.0],colors='tab:green', linewidths=1, origin='lower')
    H1 = plt.contour(X1, X2, Z - 1, [0.0], colors='orange', linewidths=1, origin='lower')
    plt.clabel(H,inline=True, fmt="0", fontsize=8)
    plt.clabel(H_1,inline=True, fmt="-1", fontsize=8)
    plt.clabel(H1,inline=True, fmt="+1", fontsize=8)
    plt.axis("tight")
    plt.show()
if __name__ == '__main__':
    main()
```