24-787: Machine Learning and Artificial Intelligence for Engineers
Ryan Wu
ID: weihuanw
Homework 4
Due: Feb 17 2024

Concept Questions:

Problem 1
Model 2 represents a better discriminator because the margin's width is maximized.

Problem 2
G should be formulated as option 4 ($G = -y*[x_1,x_2,1]$, $h = -1$).

Problem 3
Model 1 will be faster to evaluate a set of 1000 test points.

Problem 4
The RBF SVM model classifies the data best.

Problem 5
One-versus-one classifier would be faster to train.

Problem 6
For the given model, 4 data points contribute to the loss L.

Problem 1 (20 points)

Problem Description

As a lecture activity, you performed support vector classification on a linearly separable dataset by solving the quadratic programming optimization problem to create a large margin classifier.

Now, you will use a similar approach to create a soft margin classifier on a dataset that is not cleanly separable.

Fill out the notebook as instructed, making the requested plots and printing necessary values.

You are welcome to use any of the code provided in the lecture activities.

Summary of deliverables:

Functions (described later):

- `soft_margin_svm(X,y,C)`

Results:

- Print the values of w1, w2, and b for the C=0.05 case

Plots:

- Plot the data with the optimized margin and decision boundary for the case C=0.05
- Make 4 such plots for the requested C values

Discussion:

- Respond to the prompt asked at the end of the notebook

Imports and Utility Functions:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

from cvxopt import matrix, solvers
solvers.options['show_progress'] = False

def plot_boundary(x, y, w1, w2, b, e=0.1):
    x1min, x1max = min(x[:,0]), max(x[:,0])
    x2min, x2max = min(x[:,1]), max(x[:,1])

    xb = np.linspace(x1min,x1max)
    y_0 = 1/w2*(-b-w1*xb)
    y_1 = 1/w2*(1-b-w1*xb)
    y_m1 = 1/w2*(-1-b-w1*xb)
```

```
cmap = ListedColormap(["purple","orange"])

plt.scatter(x[:,0],x[:,1],c=y,cmap=cmap)
plt.plot(xb,y_0,'-',c='blue')
plt.plot(xb,y_1,'--',c='green')
plt.plot(xb,y_m1,'--',c='green')
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.axis((x1min-e,x1max+e,x2min-e,x2max+e))
```

Load data

Data is loaded as follows:

- X: input features, Nx2 array

- y: output class, length N array

```
data = np.load("data/w4-hw1-data.npy")
X = data[:, 0:2]
y = data[:, 2]
```

Soft Margin SVM Optimization Problem

For soft-margin SVM, we introduce N slack variables ξ_i (one for each point), and reformulate the optimization problem as:

$$\min_{w,b} \frac{1}{2}||w||^2 + C\sum_i \xi_i$$

$$\text{subject to: } y_i\left(w^T x_i + b\right) \geq 1 - \xi_i ; \xi_i \geq 0$$

To put this into a form compatible with `cvxopt`, we will need to assemble large matrices as described in the next section.

Soft Margin SVM function

Define a function `soft_margin_svm(X, y, C)` with inputs:

- `X`: (Nx2) array of input features
- `y`: Length N array of output classes, -1 or 1
- `C`: Regularization parameter

In this function, do the following steps:

1. Create the P, q, G, and h arrays for this problem (each comprised of multiple sub-matrices you need to combine into one)

- `P`: (3+N) x (3+N)

- Upper left: Identity matrix, but with 0 instead of 1 for the bias (third) row/column
- Upper right (3xN): Zeros
- Lower left (Nx3): Zeros
- Lower right: (NxN): Zeros

- `q`: (3+N) x (1)
  - Top (3x1): Vector of zeros
  - Bottom (Nx1): Vector filled with 'C'
- `G`: (N+N) x (N+3):
  - Upper left (Nx3): Negative y multiplied element-wise by [`x1`, `x2`, `1`]
  - Upper right (NxN): Negative identity matrix
  - Lower left (Nx3): Zeros
  - Lower right (NxN): Negative identity matrix
- `h`: (N+N) x (1)
  - Top: Vector of -1
  - Bottom: Vector of zeros

You can use `np.block()` to combine multiple submatrices into one.

1. Convert each of these into cvxopt matrices (Provided)
2. Solve the problem using `cvxopt.solvers.qp` (Provided)
3. Extract the `w1`, `w2`, and `b` values from the solution, and return them (Provided)

```
def soft_margin_svm(X, y, C):
    N = np.shape(X)[0]

    # YOUR CODE GOES HERE
    # Define P, q, G, h
    P = np.zeros((N+3,N+3))
    P[:2,:2] = np.eye(2)

    q = np.zeros(N+3)
    q[3:] = C

    G = np.zeros((N+N,N+3))
    G[:N,:2] = -y[:,None]*X
    G[:N,2] = -y
    G[:N,3:] = -np.eye(N)
    G[N:,:N] = -np.eye(N)
    G[N:,3:] = -np.eye(N)

    h = np.zeros(N+N).reshape(-1,1)
    h[:N] = -1

    z = solvers.qp(matrix(P),matrix(q),matrix(G),matrix(h))
    w1 = z['x'][0]
    w2 = z['x'][1]
```

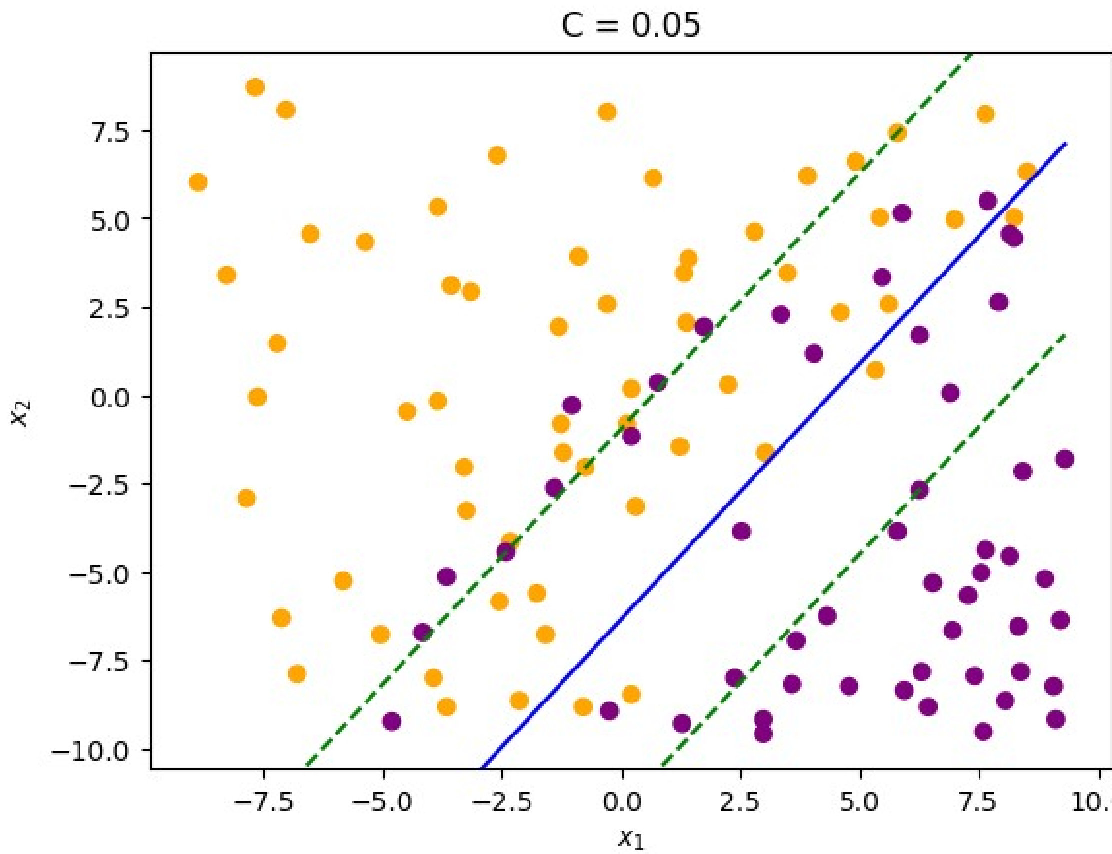
```
    b  = z['x'][2]

    return w1, w2, b
```

Demo: C = 0.05

Run the cell below to create the plot for the N = 0.05 case

```
C = 0.05
w1, w2, b = soft_margin_svm(X, y, C)
print(f"\nSolution\n--------\nw1: {w1:8.4f}\nw2: {w2:8.4f}\n b:
{b:8.4f}")

plt.figure()
plot_boundary(X,y,w1,w2,b,e=1)
plt.title(f"C = {C}")
plt.show()
```

```
Solution
--------
w1:  -0.2685
w2:   0.1857
 b:   1.1785
```



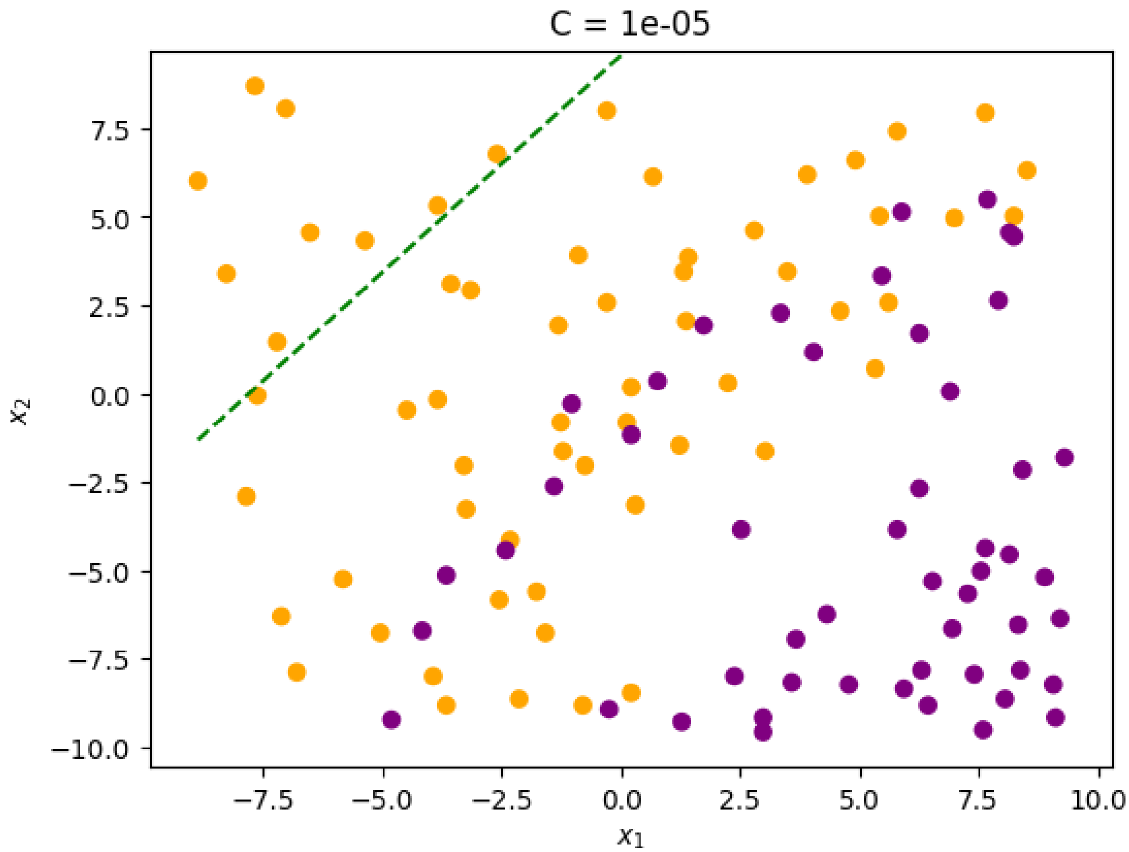
Varying C

Now loop over the C values [1e-5, 1e-3, 1e-2, 1] and generate soft margin decision boundary plots like the one above for each case.

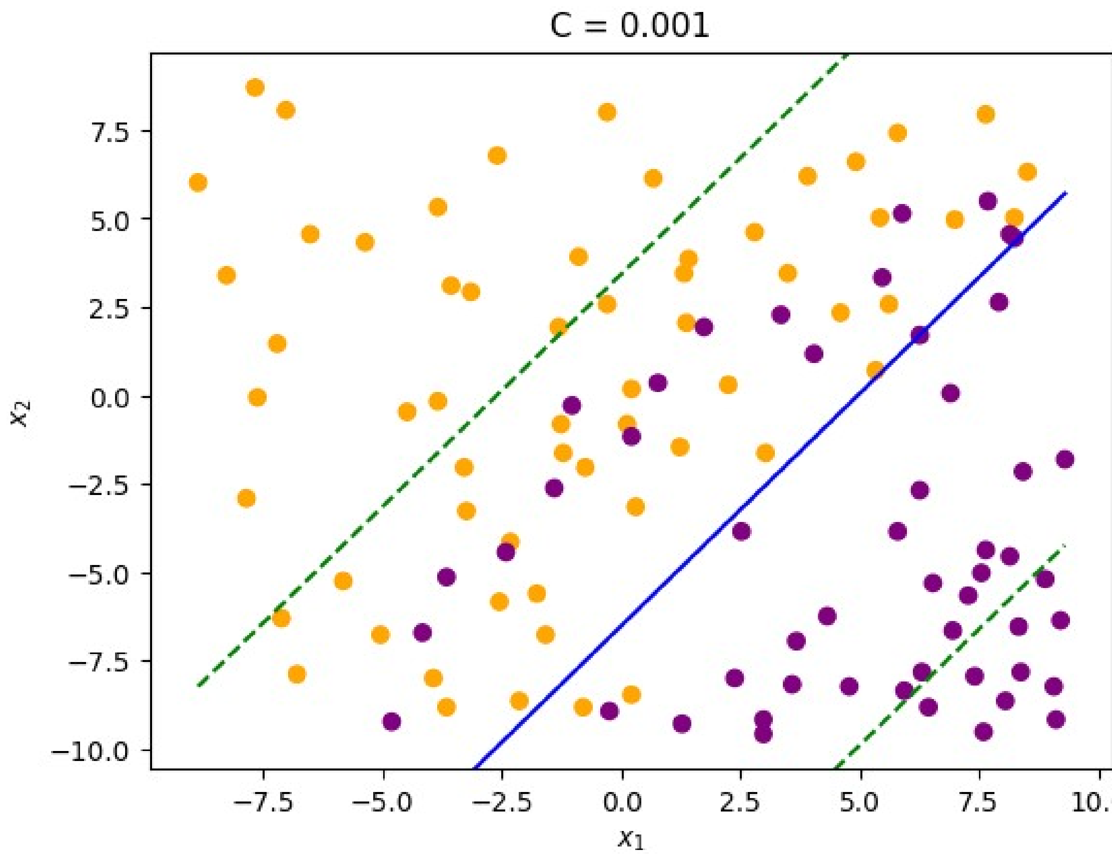
```
# YOUR CODE GOES HERE
# loop over different values of C [1e-5, 1e-3, 1e-2, 1] and plot the
decision boundary
C_values = [1e-5, 1e-3, 1e-2, 1]
for C in C_values:
    w1, w2, b = soft_margin_svm(X, y, C)
    print(f"\nSolution\n--------\nw1: {w1:8.4f}\nw2: {w2:8.4f}\n b:
{b:8.4f}")
    plt.figure()
    plot_boundary(X,y,w1,w2,b,e=1)
    plt.title(f"C = {C}")
    plt.show()
```

```
Solution
--------
w1:  -0.0025
```

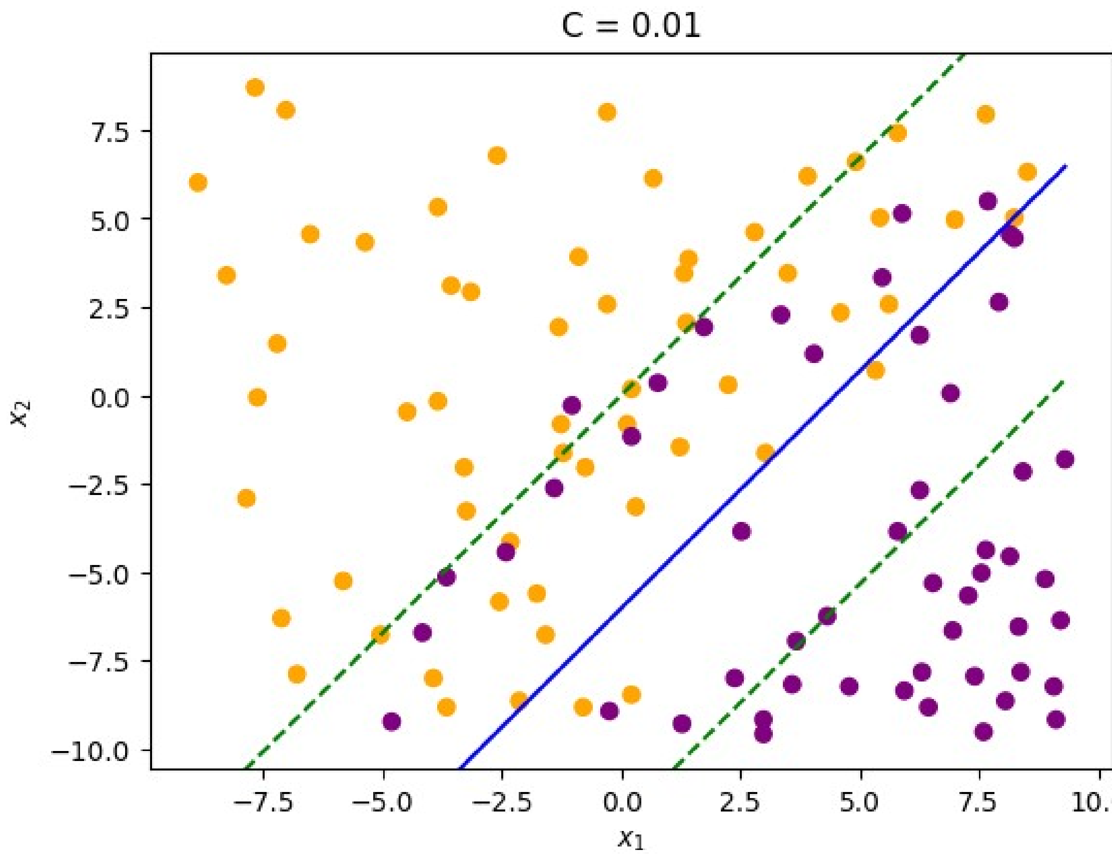
```
w2:   0.0020
 b:   0.9808
```



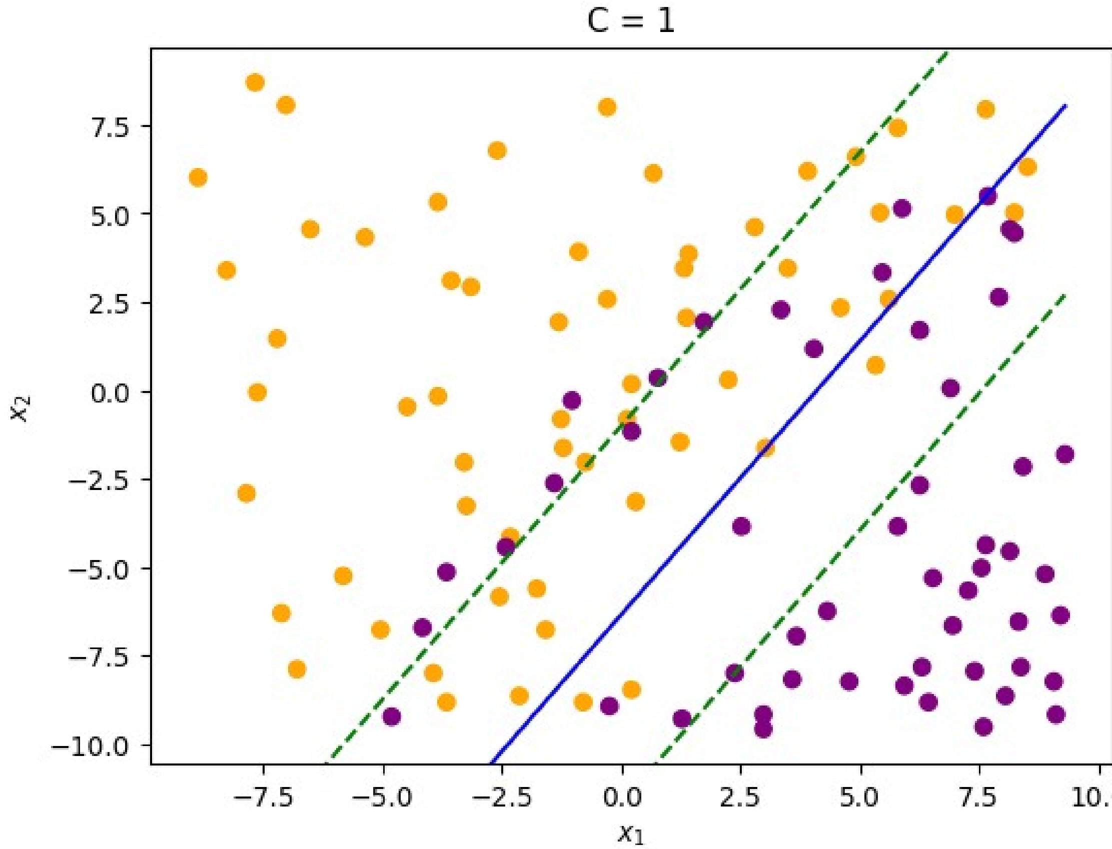

```
Solution
--------
w1:  -0.1323
w2:   0.1006
 b:   0.6563
```




```
Solution
--------
w1:  -0.2231
w2:   0.1661
 b:   1.0017
```




```
Solution
--------
w1:  -0.2899
w2:   0.1873
 b:   1.1899
```



Discussion

Please write a sentence or two discussing what happens to the decision boundary and margin as you vary C, and try to provide some rationale for why.

As the C value varies, the decision boundary and margin also change. In the above case, with a smaller C value, the decision boundary becomes less accurate and the margins between support vectors also get larger. With a larger C value, the decision boundary becomes more accurate and the margins get smaller between support vectors. The above observation is due to higher C values allowing a more complex decision boundary but can lead to overfitting and misclassifications. Lower C values lead to larger margins and better generalization to unseen data but may result in underfitting.

Problem 2 (20 points)

Problem Description

In this problem you will use `sklearn.svm.SVC` to classify thermal imaging data of a CPU die. We are interested in classifying points on the die as critical or non-critical, to inform where thermal paste should be applied to the die. The thermal imaging data is noisy, so your boss has asked you to develop a model that can produce a smoother profile of where the die is expected to be at, or above critical temperature.

The thermal imaging data is contained in `cputemp.npy`, where the first two columns correspond to the x and y position on the die, and the third column corresponds to the temperature at that point in degrees Celsius.

Fill out the notebook as instructed, making the requested plots and printing necessary values.

You are welcome to use any of the code provided in the lecture activities.

Summary of deliverables:

Functions:

- `accuracy(model, X, y)`

Results:

- Print the accuracy of the two models requested on classifying the training set points as critical or non-critical temperature

Plots:

- Plot the decision boundary of each trained model with the provided plotting functions

Discussion:

- Compare the plots and accuracy of the two models, and reason which model is the better of the two

Imports and Utility Functions:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.svm import SVC

def plot_svc_decision_function(model, ax=None):
    """Plot the decision function for a 2D SVC"""
    if ax is None:
        ax = plt.gca()
    xlim = ax.get_xlim()
```

```
    ylim = ax.get_ylim()

    # create grid to evaluate model
    x = np.linspace(xlim[0], xlim[1], 50)
    y = np.linspace(ylim[0], ylim[1], 50)
    Y, X = np.meshgrid(y, x)
    xy = np.vstack([X.ravel(), Y.ravel()]).T
    P = model.decision_function(xy).reshape(X.shape)

    # plot decision boundary and margins
    ax.contour(X, Y, P, colors='k',
               levels=[-1, 0, 1],
               linestyles=['--', '-', '--'],
               linewidths = [2,4,2])

    ax.set_xlim(xlim)
    ax.set_ylim(ylim)
    plt.show()

def plot_temp_profile(X, T, ax = None):
    if ax == None:
        ax = plt.gca()
    # Plot points colored by temperature
    sc = ax.scatter(X[:,0],X[:,1],c = T)
    # Add colorbar to plot
    cbar = plt.colorbar(sc)
    # Add labels
    cbar.set_label('Temperature ($\degree C$)')
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    plt.show()

def plot_temp_critical(X, y, ax = None):
    if ax is None:
        ax = plt.gca()
        showflag = True
    else:
        showflag = False
    ax.scatter(X[:,0],X[:,1], c = y, cmap = ListedColormap(['blue',
'red']))
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_aspect(0.8)
    if showflag:
        plt.show()
    else:
        return ax

def plot_model(model, X, y):
    # Wrapper function to generate plot and decision boundary
```

```
    ax = plt.gca()
    ax = plot_temp_critical(X,y,ax)
    plot_svc_decision_function(model, ax)
```

Load and visualize the data

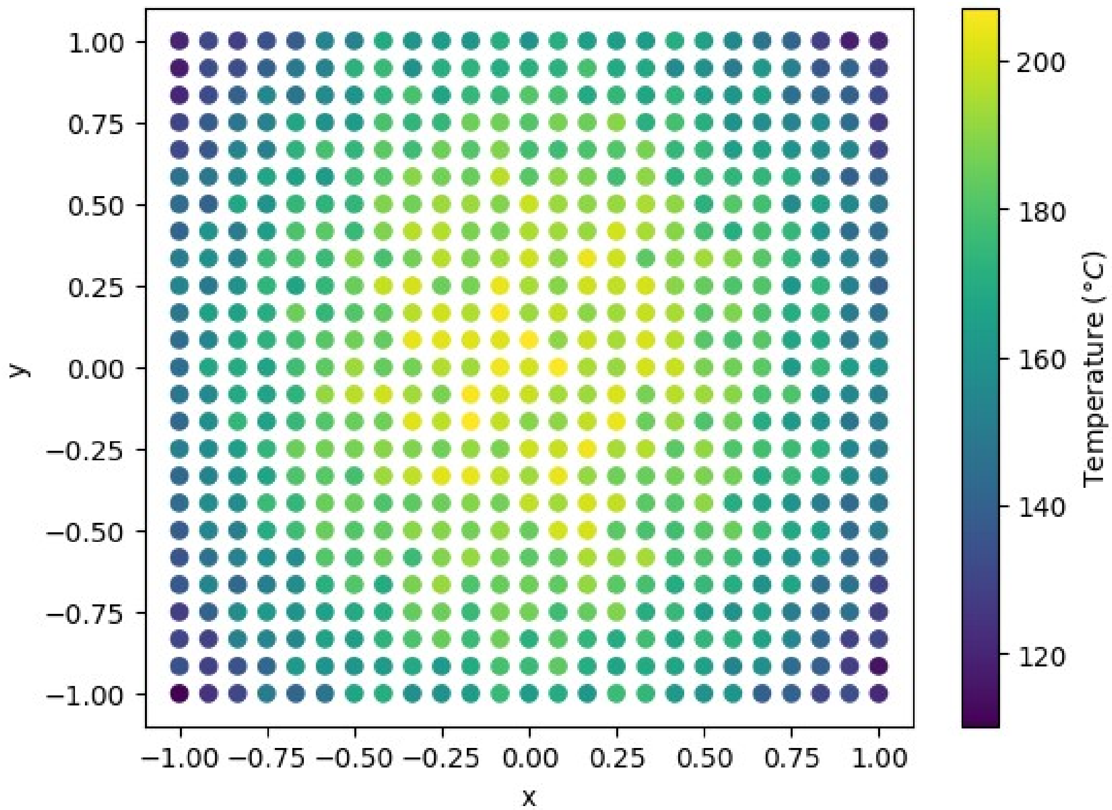
Data is contained in `cputemp.npy` and can be loaded with `np.load()`. The first two columns of the file correspond to the x and y position on the die, and the third columns corresponds to the temperature at that position in degrees Celsius.

Store the data as:

- `X` (Nx2) array of position data
- `T` (Nx1) array of temperature data

Then visualize the data with `plot_temp_profile(X,T)`

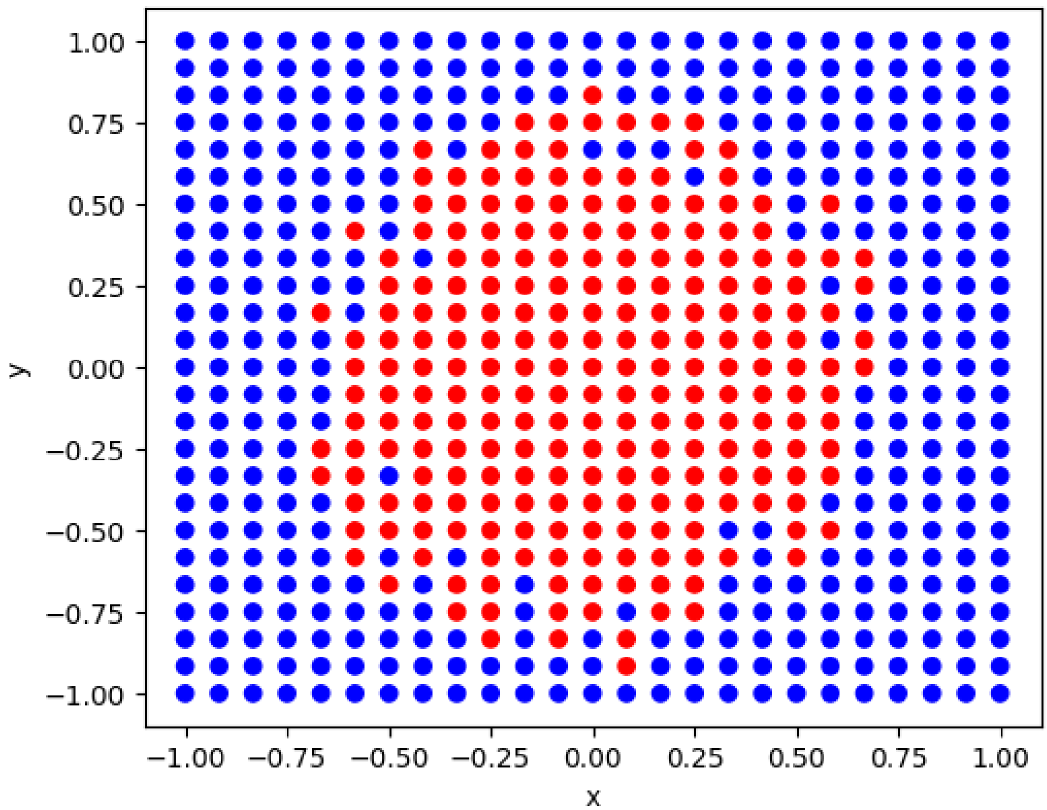
```
# YOUR CODE GOES HERE
# load the cpu temperature data
CPU_data = np.load('data/cputemp.npy')
X = CPU_data[:,:2]
T = CPU_data[:,2]

# plot the temperature profile
plot_temp_profile(X, T)
```



Assign labels to data

Now we need to assign labels to the data for the support vector machine to be able to classify points as critical or non-critical. Generate a boolean vector `y` that is True for points at or above $180 \text{\textdegree}C$, and False otherwise. Then use `plot_temp_critical(X,y)` to plot the points on the die that are critical and non-critical.

```
# YOUR CODE GOES HERE
# create the boolean vector y
y = np.zeros(T.shape[0])
if np.any(T > 180):
    y[T > 180] = 1
else:
    y[T <= 180] = 0

# plot the critical temperature
plot_temp_critical(X, y)
```



Train Support Vector Classifiers

Now you can train a SVC to classify the region on the die that you expect to be at or above the critical temperature. Using sklearn.svm.SVC train the following two models:

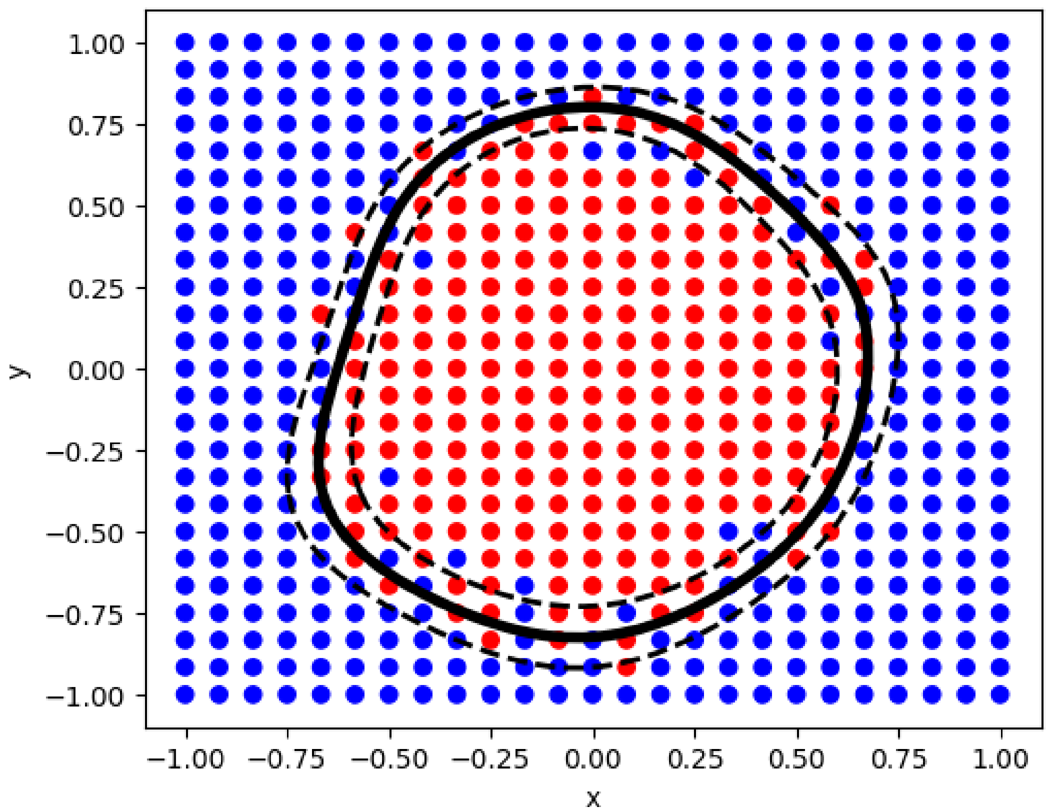
- RBF Kernel with C = 100
- 8th order polynomial Kernel with C = 100

Write a function `accuracy(model, X, y)` that takes in the model, evaluates the points in X, and computes an accuracy between the predictions and ground truth labels in `y`. Accuracy is defined as the number of correctly classified points, divided by the total number of points. For a more in depth discussion of accuracy please see: Accuracy - Wikipedia. We will cover this topic more later in the course.

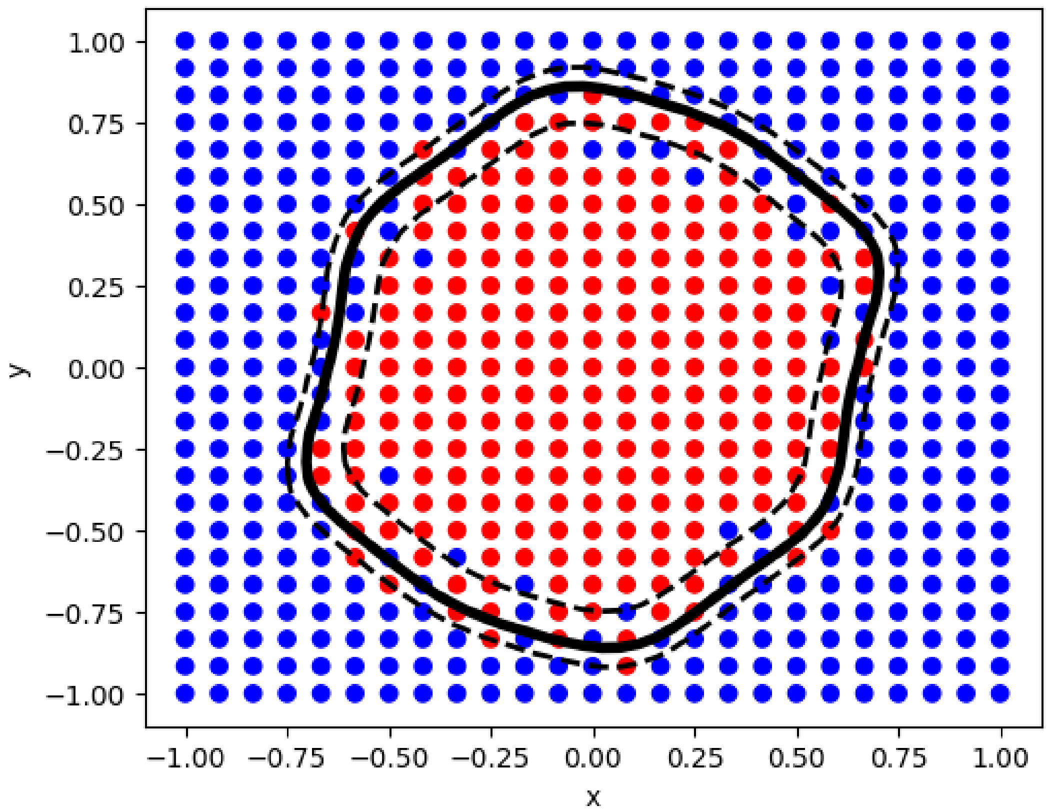
For each model, report the accuracy on the training data and use `plot_model(model, X, y)` to visualize the decision boundary.

```
# YOUR CODE GOES HERE
# Define accuracy function
def accuracy(model, X, y):
    predictions = model.predict(X)
    accuracy = np.mean(predictions == y)
    return accuracy
```

```python
# YOUR CODE GOES HERE
# Train and plot SVC models
# [Model 1] RBF Kernel with C=100
model1 = SVC(kernel='rbf', C=100)
model1.fit(X, y)
plot_model(model1, X, y)
print ("Model 1 Accuracy: ", accuracy(model1, X, y))

# [Model 2] 8th order polynomial kernel with C=100
model2 = SVC(kernel='poly', degree=8, C=100)
model2.fit(X, y)
plot_model(model2, X, y)
print ("Model 2 Accuracy: ", accuracy(model2, X, y))
```




```
Model 1 Accuracy:  0.9328
```




```
Model 2 Accuracy:  0.9232
```

Discussion

Briefly discuss the performance of the two models, both with regard to their accuracy and the appearance of the decision boundary. Which model would you submit to your boss?

The following observation can be seen when comparing two models: Model 1 (RBF) has an accuracy of 93.28% and a smoother decision boundary. Model 2 (8th-order polynomial) has an accuracy of 92.32% and has a rougher decision boundary. I would submit Model 1 (RBF) results to my boss since the prediction accuracy is better. Furthermore, the ideal decision boundary should be in the middle of the support vectors, which is the case in Model 1 (RBF).

Problem 3 (20 points)

Problem Description

In this problem you will use `sklearn.svm.SVR` to train a support vector machine for a regression problem. Your model will predict G forces experienced by a sports car as it travels through a chicane in the Nurburgring.

Fill out the notebook as instructed, making the requested plots and printing necessary values.

You are welcome to use any of the code provided in the lecture activities.

Summary of deliverables:

Results:

- Plot the fitted SVR function for three different epsilon values
- Compute the R2 score for each of the fitted functions

Discussion:

- Discuss the performance of the models and the effect of epsilon

Imports and Utility Functions:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVR

def plot_data(X, y, ax = None):
    if ax is None:
        ax = plt.gca()
        showflag = True
    else:
        showflag = False
    ax.scatter(X,y, c = 'blue')
    ax.set_xlabel('Normalized Position')
    ax.set_ylabel('G Force')
    if showflag:
        plt.show()
    else:
        return ax

def plot_svr(model, X, y):
    ax = plt.gca()
    ax = plot_data(X, y, ax)
    xs = np.linspace(min(X), max(X), 1000).reshape(-1,1)
    ys = model.predict(xs)
    ax.plot(xs,ys,'r-')
```

```
    plt.legend(['Data', 'Fitted Function'])
    plt.show()
```

Load and visualize the data

The data is contained in `nurburgring.npy` and can be loaded with `np.load()`. The first column corresponds to the normalized position of the car in the chicane, and the second column corresponds to the measured G force experienced at that point in the chicane.

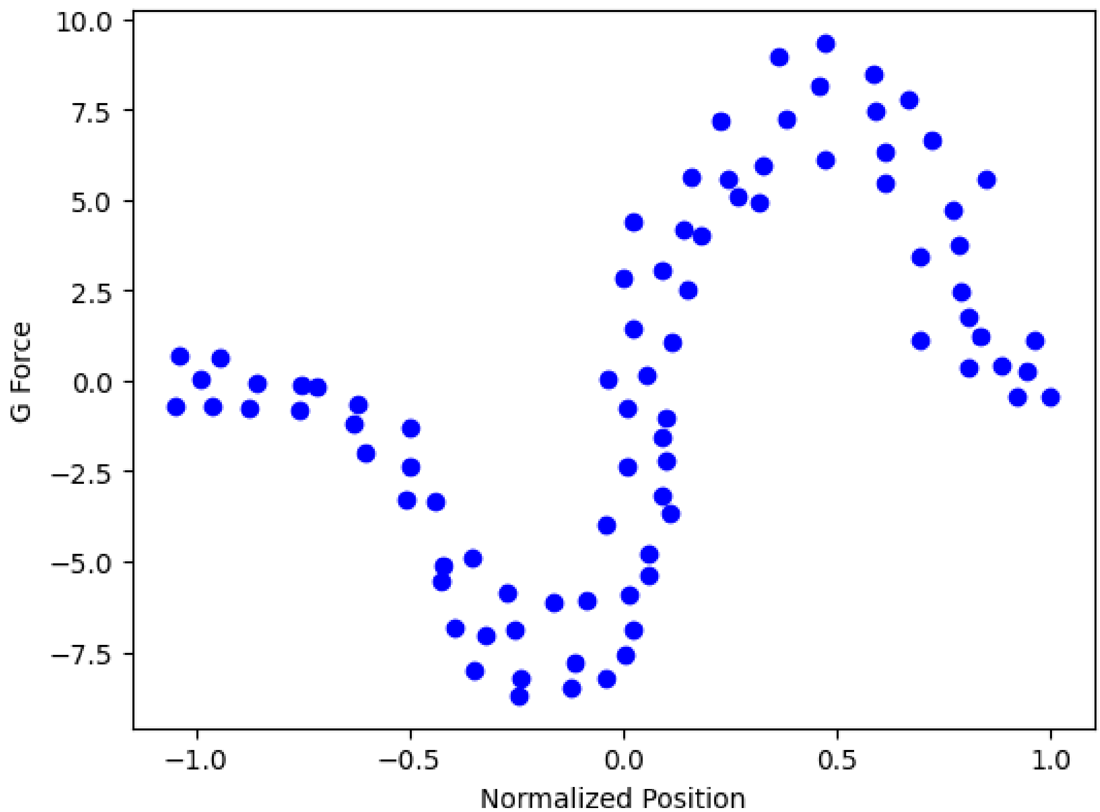
Store the data as:

- `X` (Nx1) array of position data
- `y` N-dimensional vector of G force data

Then visualize the data with `plot_data(X,y)`

Note: use X.reshape(-1,1) to make the `X` array two dimensional as required by 'SVR.fit(X,y)'

```
# YOUR CODE GOES HERE
# load the G force data
G_data = np.load('data/nurburgring.npy')
X = G_data[:,0].reshape(-1,1)
y = G_data[:,1]

# plot the data
plot_data(X,y)
```

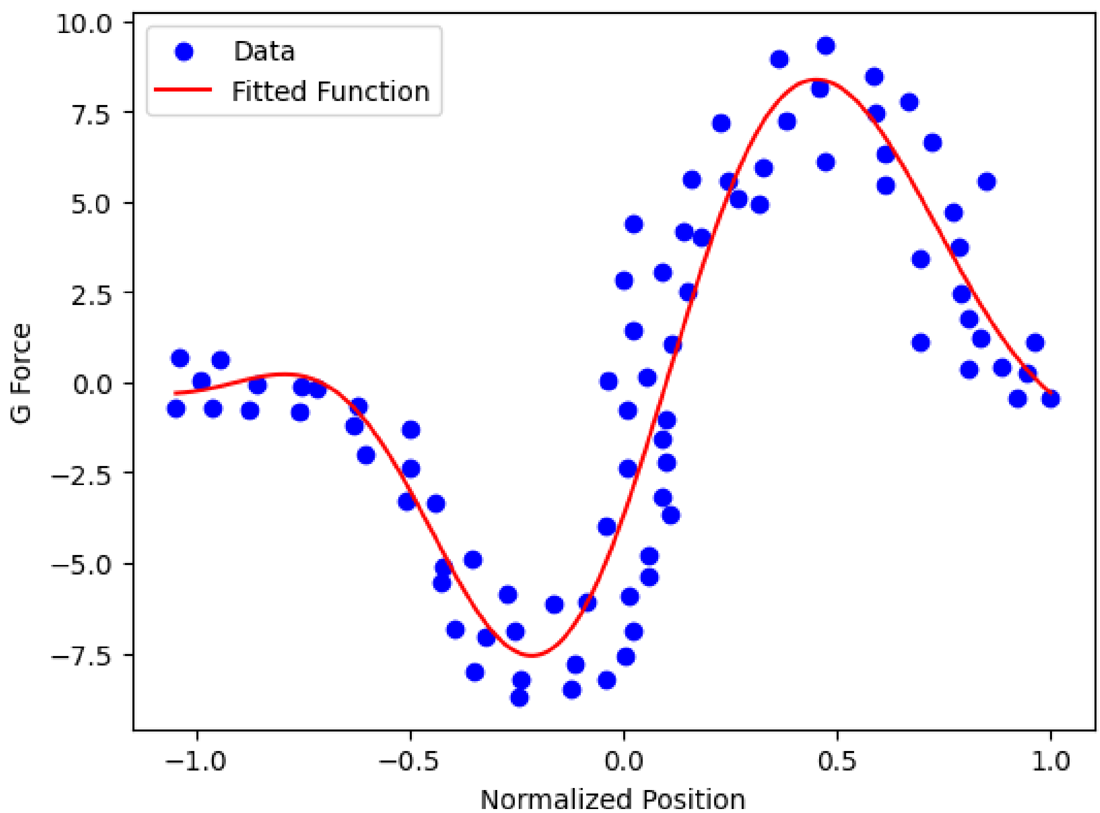


Train Support Vector Regressors

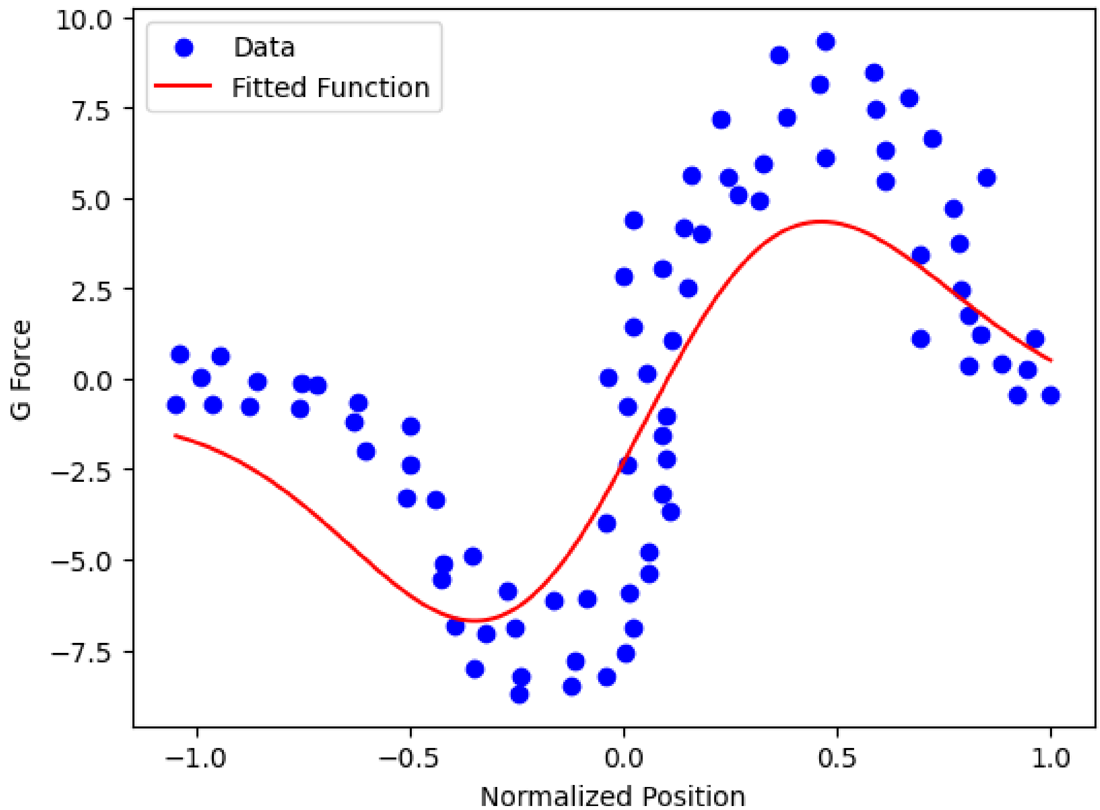
Train three different support vector regressors using the RBF Kernel, C = 100, and epsilon = [1, 5, 10]. For each model, report the coefficient of determination (R^2) for the fitted model using the builtin sklearn function} `model.score(X,y)`, and plot the fitted function against the data using `plot_svr(model, X, y)`

```
# YOUR CODE GOES HERE
# Model paramters: RBF Kernel, C =100, epsilon = [1, 5, 10]
def svr_model(X, y, epsilon):
    model = SVR(kernel= 'rbf', C = 100, epsilon = epsilon)
    model.fit(X,y)
    model.score(X,y)
    print(f'Model R^2 [RBF, C=100, epsilon:{epsilon}]:',
model.score(X,y))
    plot_svr(model, X, y)
    return model

# epsilon = 1
model1 = svr_model(X, y, 1)
# epsilon = 5
model2 = svr_model(X, y, 5)
```

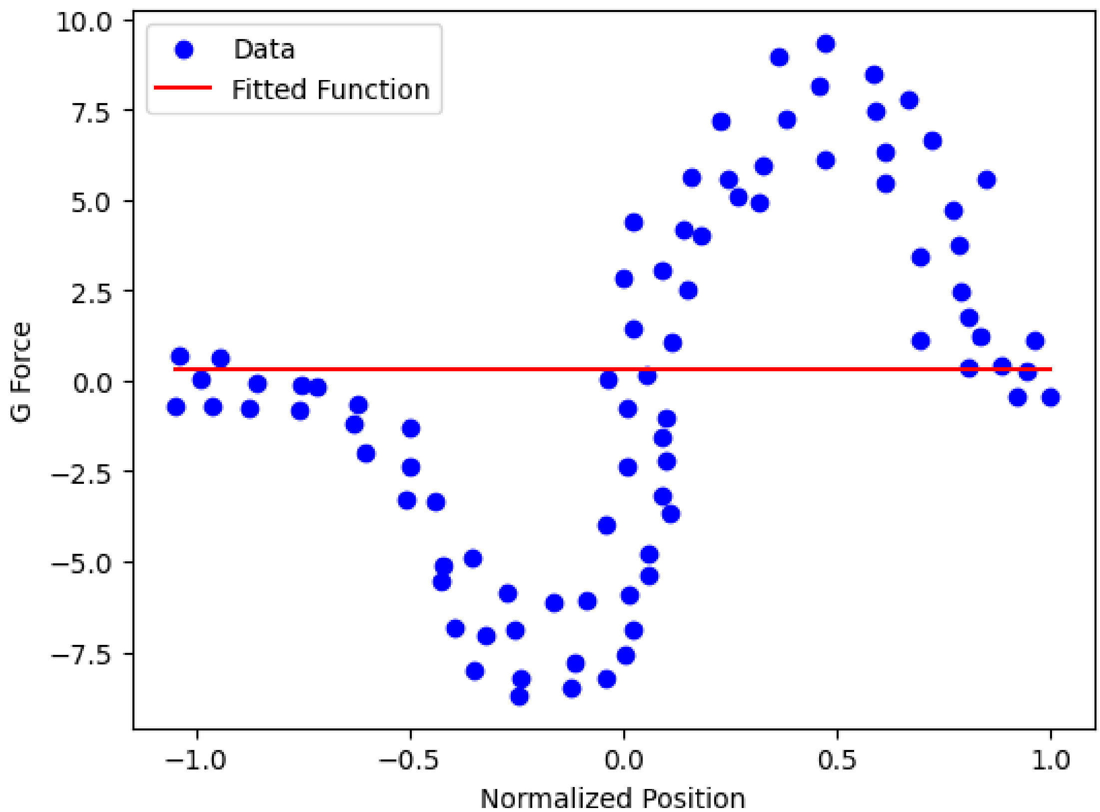
```
# epsilon = 10
model3 = svr_model(X, y, 10)

Model R^2 [RBF, C=100, epsilon:1]: 0.803896252951426
```




```
Model R^2 [RBF, C=100, epsilon:5]: 0.6412302705136446
```




```
Model R^2 [RBF, C=100, epsilon:10]: -0.005585141758953638
```



Discussion

Briefly discuss the performance of the three models, and explain how the value of epsilon influences the fitted model within the context of epsilon insenstive loss introduced in lecture.

In the context of a properly fitted function with the given data, Model 1 (epsilon = 1) performed the best, and Model 3 (epsilon = 10) performed the worst. Furthermore, the epsilon values determine the margin size within which no penalty is incurred for errors. The above observation and reasoning suggested that with a smaller epsilon value, leads to better performing models in our use case.

M4-L1 Problem 1 (5 points)

In this problem, you will perform support vector classification on a linearly separable dataset. You will do so without using an SVM package

That is, you will be solving the large margin linear classifier optimization problem:

$$\min_{w,b} \frac{1}{2} \| w \|^2$$

$$\text{subject to: } y_i \left(w^T x_i + b \right) \geq 1$$

As described in lecture, you will convert the problem into a form compatible with the quadratic programming solver in the `cvxopt` package in Python:

$$\min \frac{1}{2} x^T P x + q^T x$$

$$\text{subject to: } G x \preccurlyeq h ; A x = b$$

Your job in this notebook is to define `P`, `q`, `G`, and `h` from above.

Please install the `cvxopt` package. (You can do that in the notebook directly with `!pip install cvxopt`) Then run the next cell to make the necessary imports.

```python
# Import modules
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

from cvxopt import matrix, solvers
solvers.options['show_progress'] = False

def plot_boundary(x, y, w1, w2, b, e=0.1):
    x1min, x1max = min(x[:,0]), max(x[:,0])
    x2min, x2max = min(x[:,1]), max(x[:,1])

    xb = np.linspace(x1min,x1max)
    y_0 = 1/w2*(-b-w1*xb)
    y_1 = 1/w2*(1-b-w1*xb)
    y_m1 = 1/w2*(-1-b-w1*xb)

    cmap = ListedColormap(["purple","orange"])

    plt.scatter(x[:,0],x[:,1],c=y,cmap=cmap)
    plt.plot(xb,y_0,'-',c='blue')
    plt.plot(xb,y_1,'--',c='green')
    plt.plot(xb,y_m1,'--',c='green')
```

```
    plt.xlabel('$x_1$')
    plt.ylabel('$x_2$')
    plt.axis((x1min-e,x1max+e,x2min-e,x2max+e))
```

Load the data

```
x1 = np.array([0.0478, 1.4237, 0.2514, 0.2549, 0.3378, 0.5349, 0.7319,
0.7768,
       0.6593, 0.9807, 0.877 , 0.8321, 0.6524, 1.4231, 1.2814, 1.3021,
       1.1915, 1.0913, 1.4438, 0.0959, 0.0752, 0.1789, 0.2549, 0.324 ,
       0.4934, 0.5971, 0.6005, 0.718 , 0.5452, 0.2272, 0.7802, 0.9565,
       0.1028, 0.0579, 0.1927, 0.3862])

x2 = np.array([ 0.9555, -0.396 ,  0.8968,  0.7987,  0.7251,  0.5453,
0.5371,
        0.7088,  0.8028,  0.766 ,  0.439 ,  0.1733,  0.3082,  0.9213,
        0.6515,  0.3777,  0.1896, -0.1374,  0.112 ,  0.3368,  0.1569,
        0.2101,  0.3368,  0.2509,  0.1651,  0.0343, -0.1169, -0.2355,
       -0.3009, -0.3091, -0.3418, -0.3377, -0.3091, -0.0188,  0.0547,
-0.3091])

y = np.array([ 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])

X = np.vstack([x1,x2]).T
```

Quadratic Programming

Create the P, q, G, and h matrices as described in the lecture:

- `P` (3x3): Identity matrix, but with 0 instead of 1 for the bias (third) row/column
- `q` (3x1): Vector of zeros
- `G` (Nx3): Negative y multiplied element-wise by [`x1`, `x2`, `1`]
- `h` (Nx1): Vector of -1

Make sure the sizes of your matrices match the above. Use numpy arrays. These will be converted into `cvxopt` matrices later.

```
# YOUR CODE GOES HERE
# Define P, q, G, h

# P matrix
P = np.zeros((3, 3))
P[0, 0], P[1, 1] = 1, 1

# q matrix
q = np.zeros((3, 1))
```

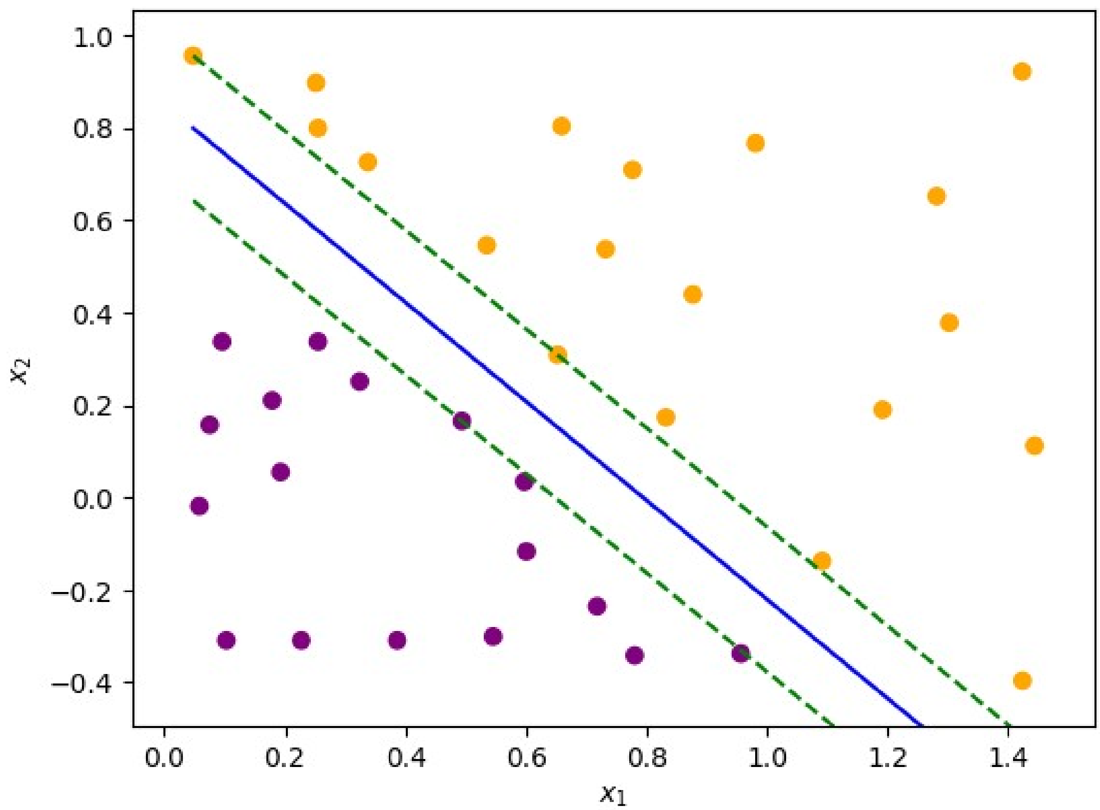
```
# G matrix
G = np.zeros((X.shape[0], 3))
G[:, 0] = -y * X[:, 0]
G[:, 1] = -y * X[:, 1]
G[:, 2] = -y * 1

# h matrix
h = np.ones(x1.shape[0]).reshape(-1,1) * -1

print("P: ",P.shape)
print("q: ",q.shape)
print("G: ",G.shape)
print("h: ",h.shape)
```

```
P:  (3, 3)
q:  (3, 1)
G:  (36, 3)
h:  (36, 1)
```

Using cvxopt for QP

Now we convert these arrays into `cvxopt` matrices and solve the quadratic programming problem. Then we get the weights `w1`, `w2`, and `b` and plot the decision boundary.

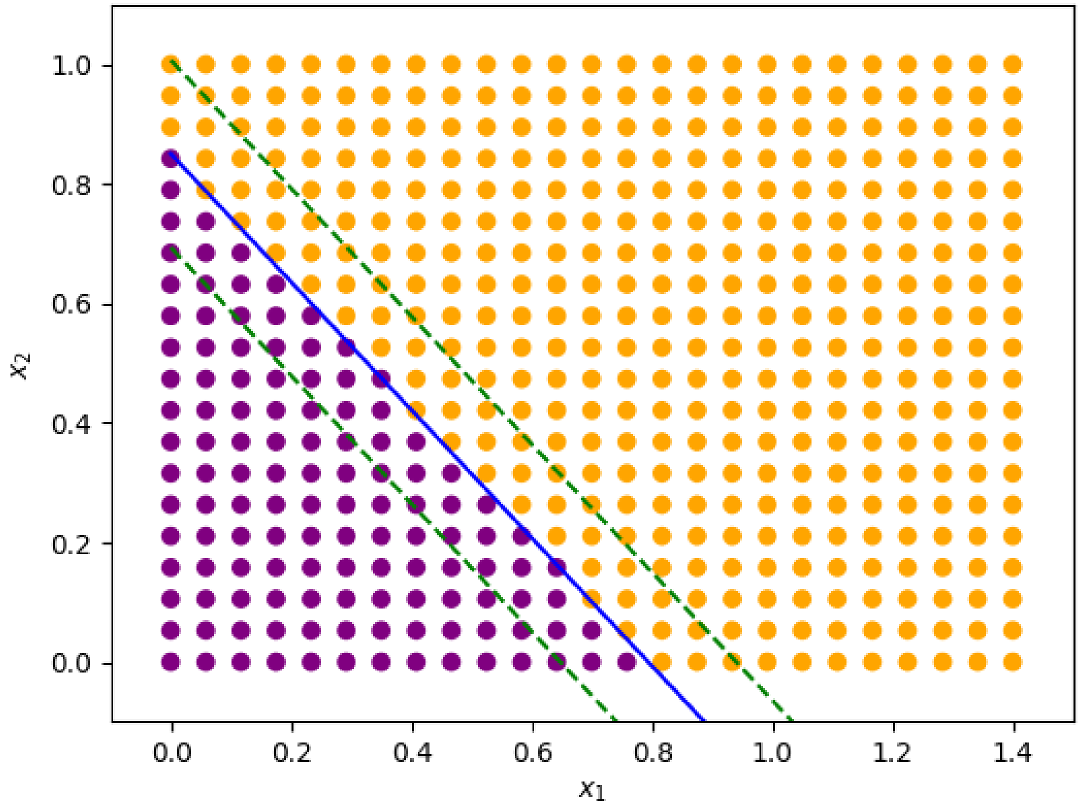
```
z = solvers.qp(matrix(P),matrix(q),matrix(G),matrix(h))
w1 = z['x'][0]
w2 = z['x'][1]
b = z['x'][2]

plot_boundary(X, y, w1, w2, b)
```



Using the SVM

Finally, we will generate a grid of (x1,x2) points and evaluate our support vector classifier on each of these points. Given the array `X_grid`, determine `y_grid`, the class of each point in `X_grid` according to the support vector machine you trained.

```
x1vals = np.linspace(0,1.4,25)
x2vals = np.linspace(0,1,20)
x1s, x2s = np.meshgrid(x1vals, x2vals)
X_grid = np.vstack([x1s.flatten(),x2s.flatten()]).T

# YOUR CODE GOES HERE
# Get y_grid
y_grid = np.dot(X_grid, np.array([w1, w2])) + b
y_grid = np.sign(y_grid)

plot_boundary(X_grid, y_grid, w1, w2, b)
```



1.0
0.8
0.6
0.4
0.2
0.0
x_2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
x_1

M4-L1 Problem 2 (5 points)

The UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/index.php) contains hundreds of public datasets donated by researchers to test machine learning/statistical methods. Here we will look at a curated version of one of these datasets and try to perform classification using SVM.

Tsanas and Xifara, cited below, performed simulations of buildings using a program called Ecotect. They modified 8 building features, and measured energy efficiency with 2 metrics: heating load requirement and cooling load requirement. For the purpose of demonstration, we have truncated the dataset to only look at a subset of the data points and building attributes.

You will be training an SVM (with sklearn) to use "relative compactness" and "wall area" to classify whether "heating load" is high (>20) or low (<=20).

```
Dataset source:
A. Tsanas, A. Xifara: 'Accurate quantitative estimation of energy
performance of residential buildings using statistical machine
learning tools', Energy and Buildings, Vol. 49, pp. 560-567, 2012
```

Run the following cell to perform the necessary imports and load the data:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from matplotlib.colors import ListedColormap

def plot_data(x,y,e=0.1):
    x1min, x1max = min(x[:,0]), max(x[:,0])
    x2min, x2max = min(x[:,1]), max(x[:,1])

    xb = np.linspace(x1min,x1max)

    cmap = ListedColormap(["blue","red"])


    plt.scatter(x[:,0],x[:,1],c=y,cmap=cmap)
    plt.colorbar()

    plt.xlabel('$x_1$')
    plt.ylabel('$x_2$')
    plt.axis((x1min-e,x1max+e,x2min-e,x2max+e))


def plot_SV_decision_boundary(svm, extend=True):
    ax = plt.gca()
```

```
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    xrange = xlim[1] - xlim[0]
    yrange = ylim[1] - ylim[0]

    x = np.linspace(xlim[0] - extend*xrange, xlim[1] + extend*xrange, 
100)
    y = np.linspace(ylim[0] - extend*yrange, ylim[1] + extend*yrange, 
100)

    X,Y = np.meshgrid(x,y)
    xy = np.vstack([X.ravel(), Y.ravel()]).T
    P = svm.decision_function(xy)

    P = P.reshape(X.shape)
    ax.contour(X, Y, P, colors='k',levels=[0],linestyles=['-'])
    ax.contour(X, Y, P, colors='k',levels=[-1, 1],
alpha=0.6,linestyles=['--'])

    plt.xlim(xlim)
    plt.ylim(ylim)

relative_compactness = np.array([0.98, 0.9 , 0.86, 0.82, 0.79, 0.76, 
0.74, 0.71, 0.69, 0.66, 0.64,
       0.62])
wall_area = np.array([294. , 318.5, 294. , 318.5, 343. , 416.5, 245. , 
269.5, 294. ,
       318.5, 343. , 367.5])
heating_load = np.array([24.58, 29.03, 26.28, 23.53, 35.56, 32.96, 
10.36, 10.71, 11.11,
       11.68, 15.41, 12.96])
```

Train an SVM in sklearn

Perform the following steps:

- Combine `relative_compactness` and `wall_area` into one 2-column input feature array
- Transform `heating_load` into an array of classes with -1 where `heating_load` entries are less than 20, and +1 otherwise.
- Create a Support Vector Classification model in sklearn. Make sure to use a "linear" kernel! Also set the argument "C" to a large number, like `1e5`.
- Fit the SVC to your data

```
# YOUR CODE GOES HERE
# combine relative_compactness and wall_area into one 2-column input 
feature array
```

```
combined_compactness_area = np.column_stack((relative_compactness,
wall_area))

# transform heating_load into an array of classes with -1 where
heating_load is less than 20, and +1 otherwise
heating_load_class = np.where(heating_load < 20, -1, +1)

# create a Support Vector Classification model in sklearn, Make sure
to use a "linear" kernel! Also set the arguement "C" to a large number
like 1e5
svc_model = SVC(kernel='linear', C=1e5)

# fit the SVC to your data
svc_model.fit(combined_compactness_area, heating_load_class)
```

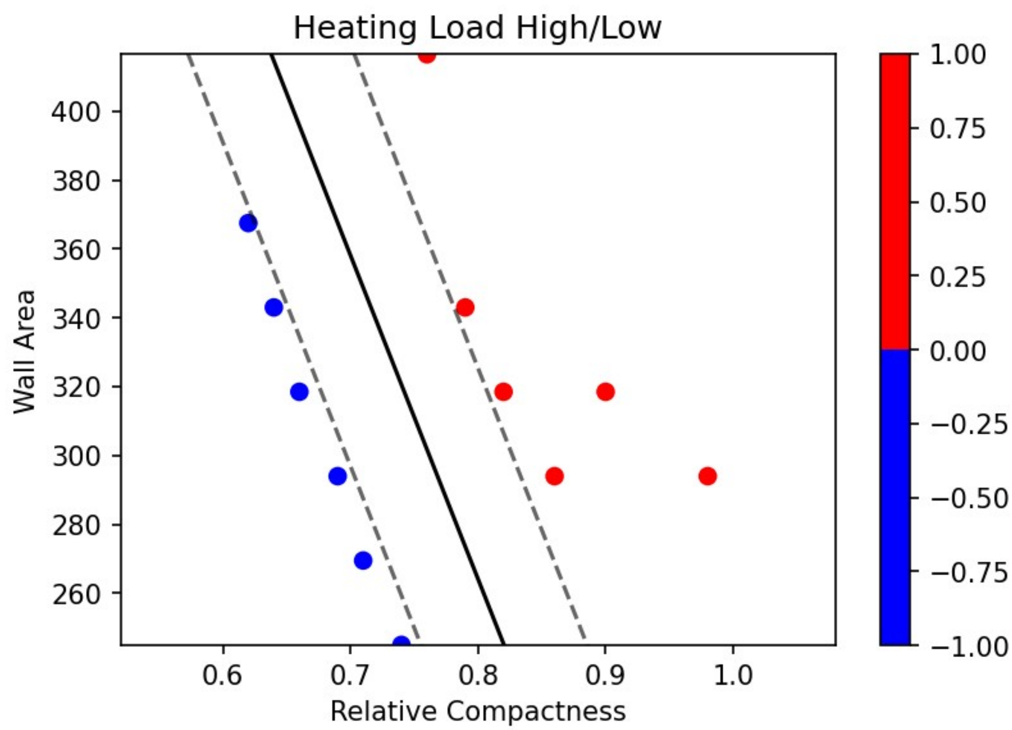
```
SVC(C=100000.0, kernel='linear')
```

Plotting results

You can make predictions on any X data using the `.predict(X)` method of your SVC model. The `.decision_function()` method will return a continuous class evaluation, 0 at the boundary and 1 or -1 at the margin edges.

Now use the provided function `plot_SV_decision_boundary()`, which takes an sklearn model as its input, to plot the decision boundary.

```
plt.figure(figsize=(6,4),dpi=150)
X = combined_compactness_area
y = heating_load_class
plot_data(X,y)

# YOUR CODE GOES HERE
plot_SV_decision_boundary(svc_model)

plt.xlabel("Relative Compactness")
plt.ylabel("Wall Area")
plt.title("Heating Load High/Low")
plt.show()
```


Heating Load High/Low
Wall Area
Relative Compactness
400
380
360
340
320
300
280
260
0.6
0.7
0.8
0.9
1.0
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

M4-L1 Problem 3 (5 points)

In this problem you will use sklearn's support vector classification to study the effect of changing the parameter C, which represents inverse regularization strength.

Run the following cell to import libraries, define functions, and load data:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from matplotlib.colors import ListedColormap

# Plotting functions:
def plot_data(X,c,s=30):
    lims = [0,1]

    markers = [dict(marker="o", color="royalblue"), dict(marker="s", 
color="crimson"), dict(marker="^", color="limegreen")]

    x,y = X[:,0], X[:,1]
    iter = 0
    for i in np.unique(c):
        marker = markers[iter]
        iter += 1
        plt.scatter(x[c==i], y[c==i], s=s, **(marker), 
edgecolor="black", linewidths=0.4, label="y = " + str(i))

def plot_SVs(svm, s=120):
    sv = svm.support_vectors_
    x, y = sv[:,0], sv[:,1]
    plt.scatter(x, y, s=s, edgecolor="black", facecolor="none", 
linewidths=1.5)

def plot_SV_decision_boundary(svm, margin=True,extend=True, 
shade_margins=False, shade_decision=False):
    ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    xrange = xlim[1] - xlim[0]
    yrange = ylim[1] - ylim[0]

    x = np.linspace(xlim[0] - extend*xrange, xlim[1] + extend*xrange, 
200)
    y = np.linspace(ylim[0] - extend*yrange, ylim[1] + extend*yrange, 
200)

    X,Y = np.meshgrid(x,y)
```

```
    xy = np.vstack([X.ravel(), Y.ravel()]).T
    P = svm.decision_function(xy)

    P = P.reshape(X.shape)
    ax.contour(X, Y, P, colors='k',levels=[0],linestyles=['-'])
    if margin:
        ax.contour(X, Y, P, colors='k',levels=[-1, 1],
alpha=0.6,linestyles=['--'])

    if shade_margins:
        cmap = ListedColormap(["white","lightgreen"])

plt.pcolormesh(X,Y,np.abs(P)<1,shading="nearest",cmap=cmap,zorder=-
999999)

    if shade_decision:
        cmap = ListedColormap(["lightblue","lightcoral"])
        pred = (svm.predict(xy).reshape(X.shape) == 1).astype(int)
        plt.pcolormesh(X,Y,pred,shading="nearest",cmap=cmap,zorder=-
1000)

    plt.xlim(xlim)
    plt.ylim(ylim)

def make_plot(title,svm_model,Xdata,ydata):
    plt.figure(figsize=(5,5))
    plot_data(Xdata,ydata)
    plot_SVs(svm_model)

plot_SV_decision_boundary(svm_model,margin=True,shade_decision=True)
    plt.legend()
    plt.xlabel("$x_1$")
    plt.ylabel("$x_2$")
    plt.title(title)
    plt.show()

# Dataset 1:
x1 = np.array([0.48949729, 0.93403431, 0.77318605, 0.99708798,
0.7453347 ,       0.62782192, 0.88728846, 0.71619404, 0.91387844,
0.38568815,
       0.74459769, 0.75305792, 0.79103743, 0.63603483, 0.7035605 ,
0.84037653, 0.47648924, 0.82480262, 0.67128124, 1.00348416,
       0.69268775, 0.74637666, 0.62823845, 0.92394124, 0.52824645,
0.66571952, 0.5772065 , 0.8942154 , 0.84369312, 0.61840017,
       0.68742653, 0.79431218, 0.76105703, 0.729959  , 0.58809188,
0.63920244, 0.75007448, 0.69128972, 0.94851858, 0.88077771,
       0.71621743, 0.68913748, 0.94206083, 0.83811487, 0.52095808,
0.72136467, 0.70606728, 0.65459534, 0.69047433, 0.78913417,
       0.660455  , 0.54130881, 0.99176949, 0.41660508, 0.61517452,
0.76214   , 0.92212188, 0.90712313, 0.61986537, 0.61543379,
```

```
       0.26571114, 0.51712792, 0.17642698, 0.38630807, 0.27326383,
0.4757757 , 0.43221499, 0.29701567, 0.2855336 , 0.36724752,
       0.41828429, 0.55323218, 0.30897445, 0.51987077, 0.25015929,
0.29285768, 0.06361631, 0.32100622, 0.44267413, 0.56155981,
       0.43747171, 0.41560485, 0.40850384, 0.53710681, 0.2458796 ,
0.36389757, 0.34206599, 0.44241723, 0.49718833, 0.41927943,
       0.53785843, 0.56305326, 0.18442455, 0.4783044 , 0.341153  ,
0.59226031, 0.34403529, 0.64020965, 0.5783743 , 0.65201187,
       0.54259663, 0.36260852, 0.28089588, 0.28126787, 0.5046967 ,
0.32032048, 0.25728685, 0.30410956, 0.39587441, 0.53701888,
       0.37573027, 0.43281125, 0.10385945, 0.45855828, 0.12496919,
0.43889099, 0.30972969, 0.32992047, 0.40483719, 0.30036318])
x2 = np.array([0.82692832, 0.64782992, 0.51168806, 0.66255369,
0.80959079,       0.74825032, 0.62810149, 0.77523882, 0.76464772,
0.67861015,
       0.74030383, 0.76234673, 0.57673835, 0.76739864, 0.70551825,
0.76417749, 0.68736246, 0.68255718, 0.6896616 , 0.65142488,
       0.72477217, 0.81890284, 0.75486623, 0.57160741, 0.71961768,
0.69643131, 0.78733278, 0.68253707, 0.74527377, 0.85515197,
       0.6174821 , 0.69385581, 0.72352607, 0.57192729, 0.69906178,
0.85159439, 0.65319918, 0.77788724, 0.73044646, 0.79092217,
       0.81828425, 0.61449583, 0.54882155, 0.61557563, 0.76571808,
0.63905784, 0.82482057, 0.71437531, 0.73098551, 0.69257621,
       0.79516325, 0.71840235, 0.67254172, 0.58651416, 0.5778736 ,
0.8128274 , 0.77131005, 0.83007228, 0.58264091, 0.75917111,
       0.3216439 , 0.43068008, 0.48166151, 0.29743746, 0.45100559,
0.37373449, 0.33908254, 0.47230067, 0.42985384, 0.40687294,
       0.3776663 , 0.39820282, 0.43011064, 0.32873478, 0.35169937,
0.25739568, 0.34931656, 0.2860302 , 0.41440527, 0.33384387,
       0.26646292, 0.44178363, 0.28835415, 0.45468991, 0.19393014,
0.42472115, 0.21083439, 0.3441914 , 0.38892878, 0.44150478,
       0.38262922, 0.36293124, 0.4006077 , 0.34750469, 0.35023348,
0.3905313 , 0.17185166, 0.44013747, 0.34005945, 0.36445769,
       0.40579986, 0.23702401, 0.38844385, 0.29752652, 0.18619147,
0.46662002, 0.33503445, 0.43295842, 0.41922308, 0.46949822,
       0.32186971, 0.37281822, 0.36488808, 0.37194919, 0.30829606,
0.39365028, 0.48855396, 0.40258577, 0.46366417, 0.33758804])
y1 = np.array([-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,  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])
X1 = np.vstack([x1,x2]).T
```

```
# Dataset 2:
z1 = np.array([0.4623709 , 0.68787981, 0.22665386, 0.42140211,
0.30510439,       0.53488987, 0.2040148 , 0.39919817, 0.32411647,
0.32894411,
       0.58131992, 0.21989461, 0.41031163, 0.2825145 , 0.71079507,
0.4301869 , 0.29867119, 0.35561876, 0.35892493, 0.3809551 ,
       0.25007082, 0.40050165, 0.45727726, 0.45009186, 0.3127013 ,
0.24118917, 0.37026561, 0.29343492, 0.30929023, 0.32183529,
       0.62142011, 0.24273132, 0.63236235, 0.39114511, 0.48803606,
0.51600837, 0.26834863, 0.52915085, 0.4940113 , 0.22678134,
       0.779535  , 0.94994687, 0.73010308, 0.61598114, 0.61310177,
0.51381933, 0.34398293, 0.61695795, 0.78951194, 0.62907221,
       0.51162408, 0.62770167, 0.80566504, 0.53683386, 0.48664659,
0.66135962, 0.68646158, 0.53325602, 0.46166815, 0.58555708,
       0.82291395, 0.6414185 , 0.54730993, 0.67858451, 0.53265047,
0.49505561, 0.64200182, 0.36407551, 0.76930752, 0.30522461,
       0.64641634, 0.41411608, 0.64992294, 0.60316402, 0.88008764,
0.75418984, 0.4862578 , 0.66244808, 0.77193682, 0.62495635])
z2 = np.array([0.83290004, 0.66234451, 0.65801115, 0.84029466,
0.70126933,       0.82112621, 0.83142114, 0.80780069, 0.69836278,
0.70415788,
       0.81111503, 0.69181695, 0.81230644, 0.68982279, 0.70037483,
0.79716711, 0.85375938, 0.63633106, 0.61071921, 0.74369119,
       0.87396874, 0.63583241, 0.62337179, 0.71575062, 0.59439517,
0.59527384, 0.57959709, 0.56120683, 0.70760421, 0.68391646,
       0.81318113, 0.74471739, 0.76689873, 0.74142189, 0.58628648,
0.58050036, 0.83946113, 0.51560503, 0.75078613, 0.77018053,
       0.49047076, 0.61580307, 0.46660621, 0.41485462, 0.50601875,
0.55752863, 0.53187983, 0.53825942, 0.57596334, 0.70985225,
       0.37757746, 0.47083258, 0.59490871, 0.4743862 , 0.41337164,
0.30688374, 0.48155856, 0.42810555, 0.66923995, 0.29000443,
       0.41406711, 0.58475545, 0.43525632, 0.61888062, 0.47842385,
0.40661197, 0.71625865, 0.61275964, 0.45230234, 0.55631826,
       0.64427582, 0.37797242, 0.59767007, 0.2815758 , 0.5679225 ,
0.35863786, 0.50579416, 0.3072999 , 0.64316316, 0.47989125])
y2 = np.array([-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,  1,  1,  1,  1,  1,  1,
1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1])
X2 = np.vstack([z1,z2]).T
```

Linearly Separable Dataset

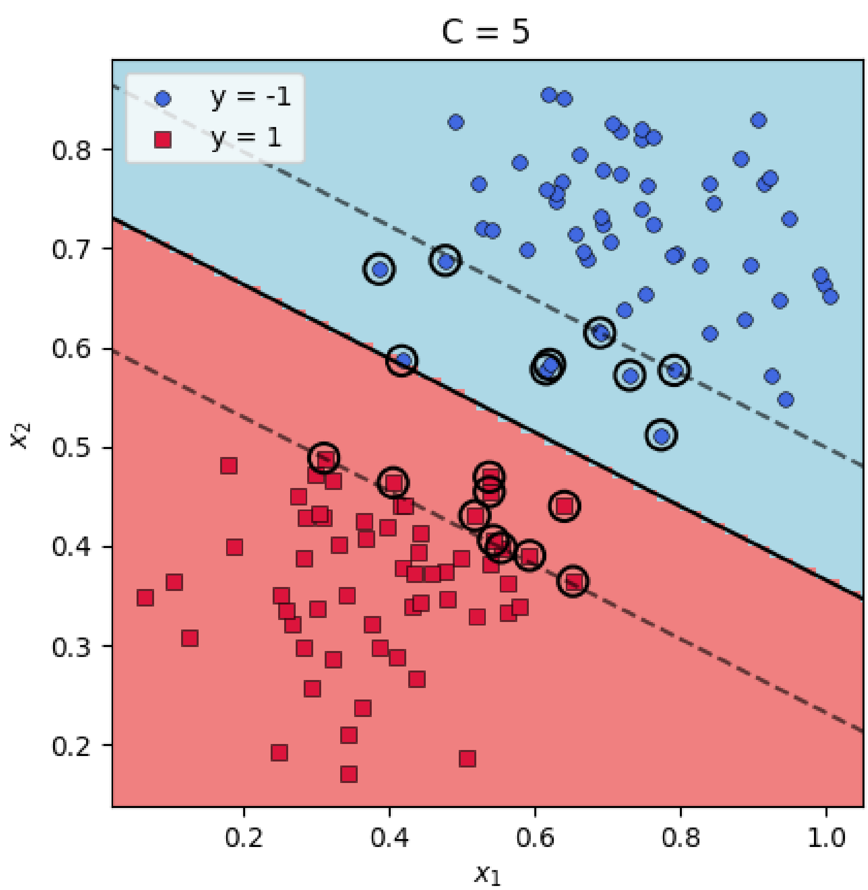
`X1` and `y1` are the features and classes for a linearly separable dataset. Train 4 SVC models on the data. Set `kernel="linear"`, but use four different regularization values:

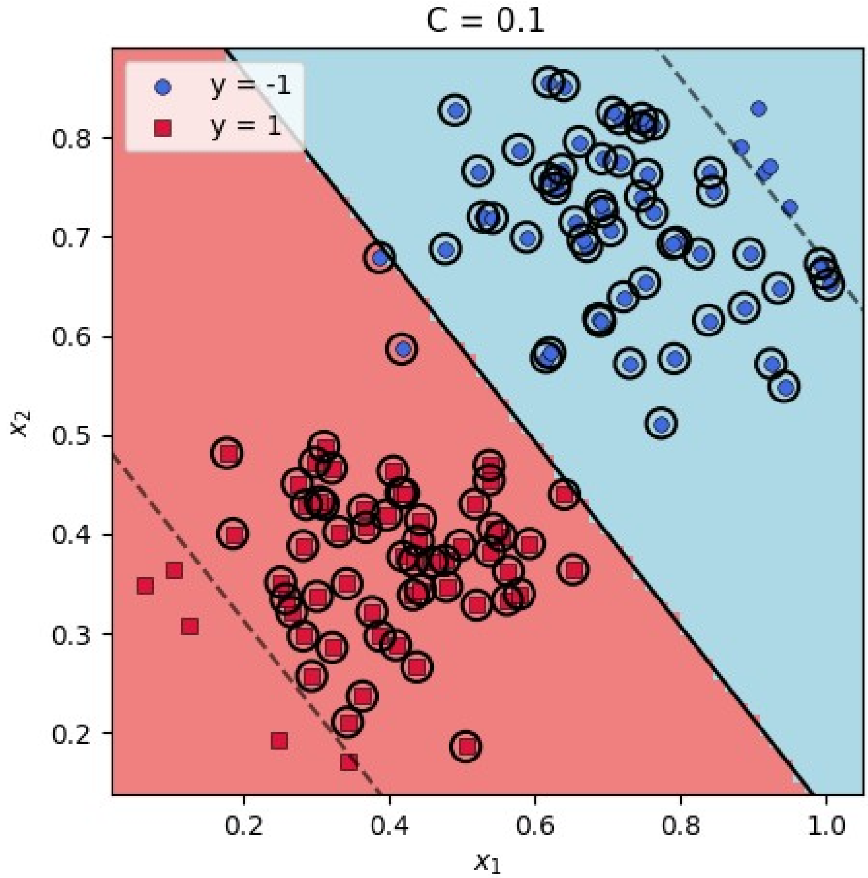
- C = 0.1
- C = 1
- C = 10
- C = 1000

For each of these models, create a plot that shows the data, decision boundary, and support vectors, complete with a title that states the C value.
Use the provided function `make_plot(title,svm_model,Xdata,ydata)`

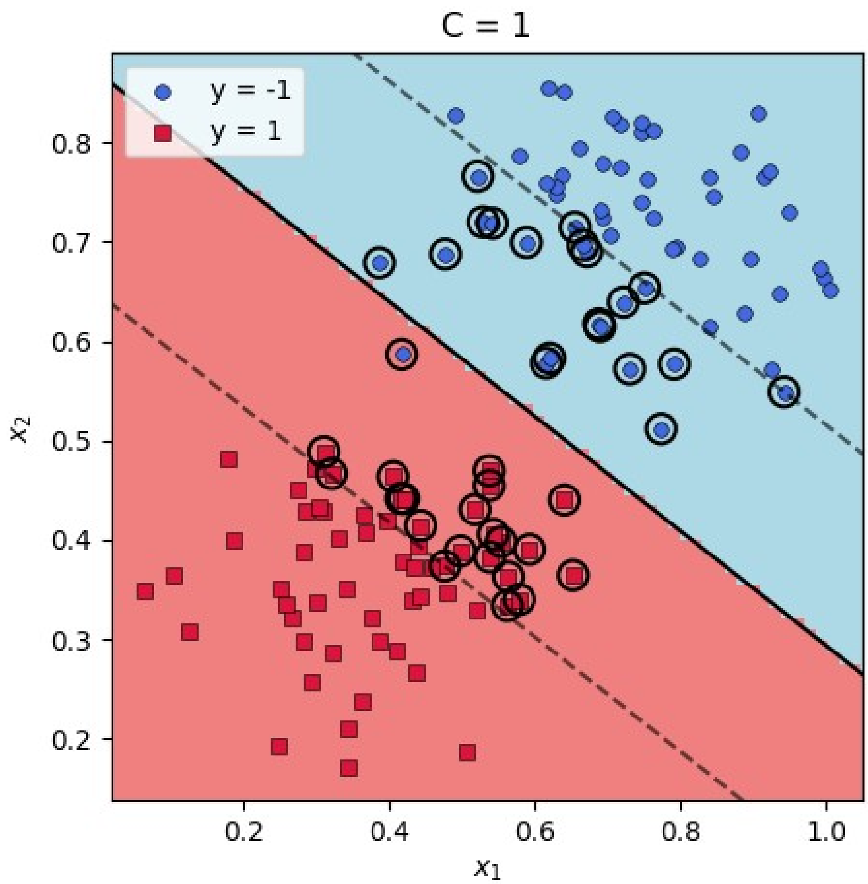
One example has been provided. Please repeat for all of the requested C values:

```
C = 5
svm = SVC(C=C,kernel="linear")
svm.fit(X1,y1)
make_plot(f"C = {C}",svm,X1,y1)


for C in [0.1,1,10,1000]:
    # YOUR CODE GOES HERE
    svm = SVC(C=C,kernel="linear")
    svm.fit(X1,y1)
    make_plot(f"C = {C}",svm,X1,y1)
```

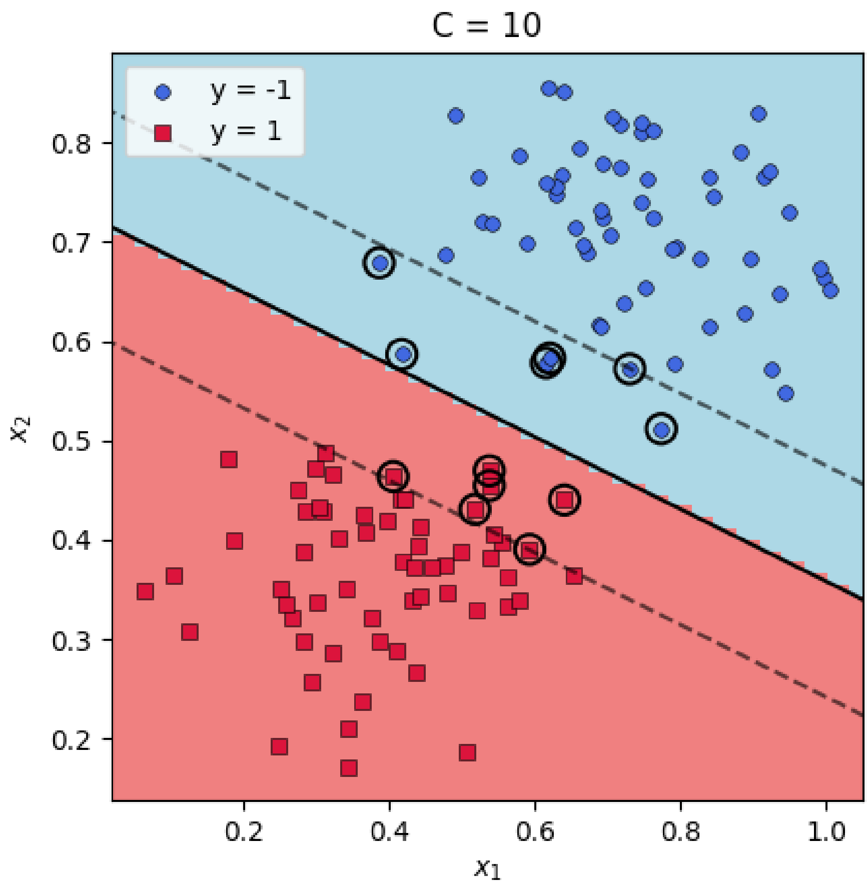




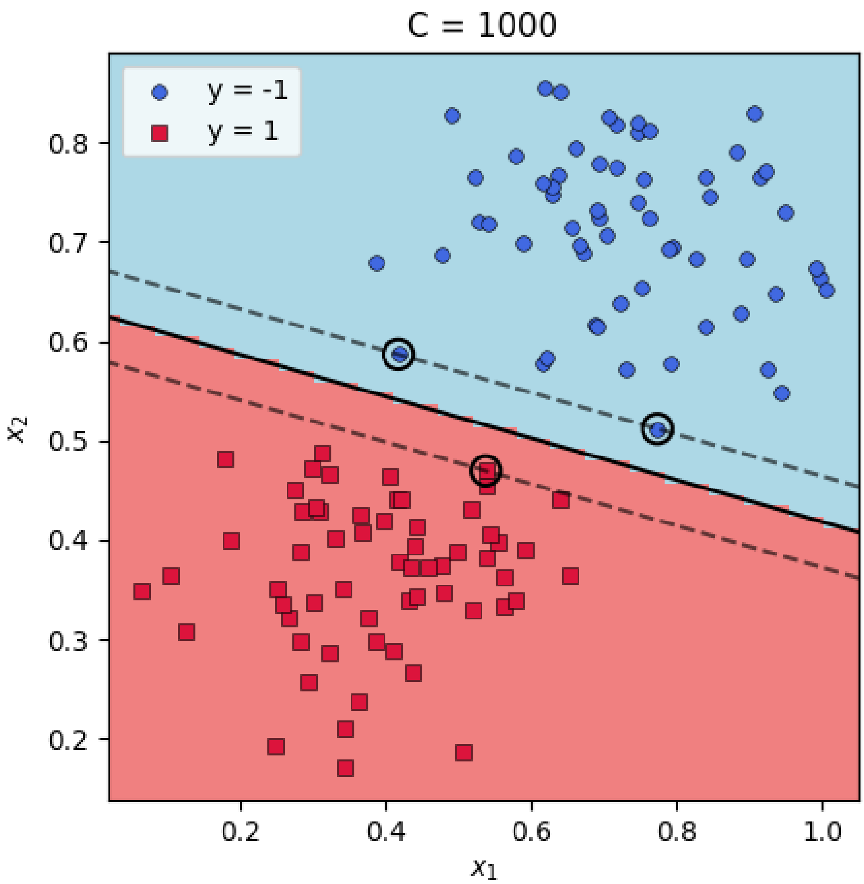
C = 0.1
y = -1
y = 1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.2
0.4
0.6
0.8
1.0
x_1
x_2



C = 1
y = -1
y = 1
0.8
0.7
0.6
0.5
0.4
0.3
0.2
x_2
0.2
0.4
0.6
0.8
1.0
x_1



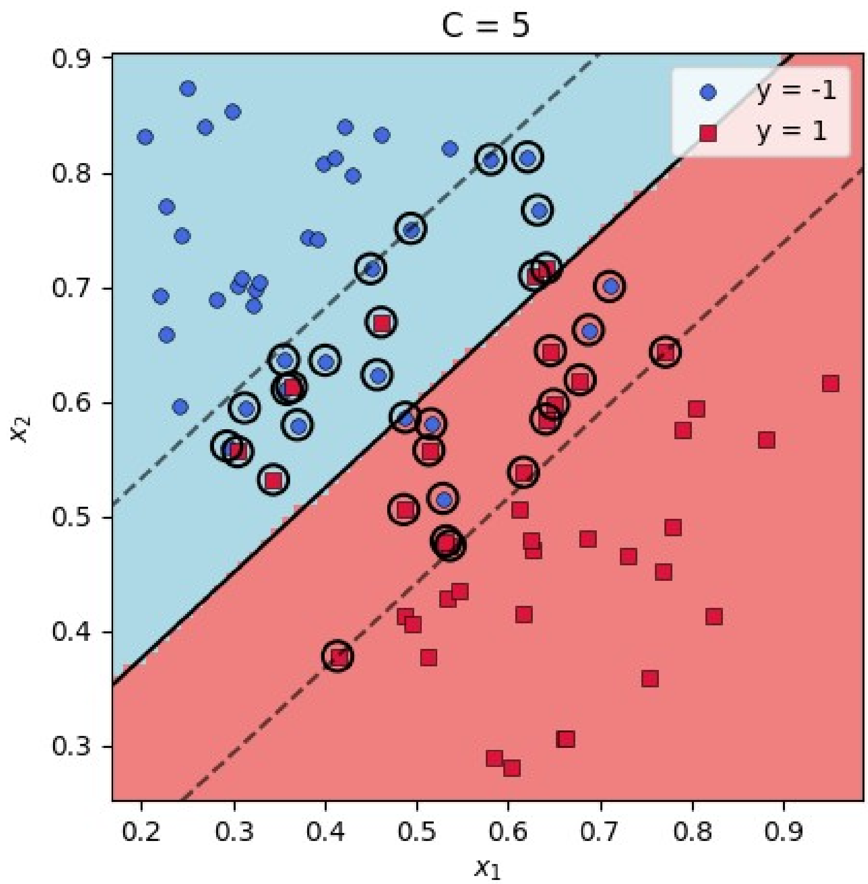
C = 10
y = -1
y = 1
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.2
0.4
0.6
0.8
1.0
x_1
x_2

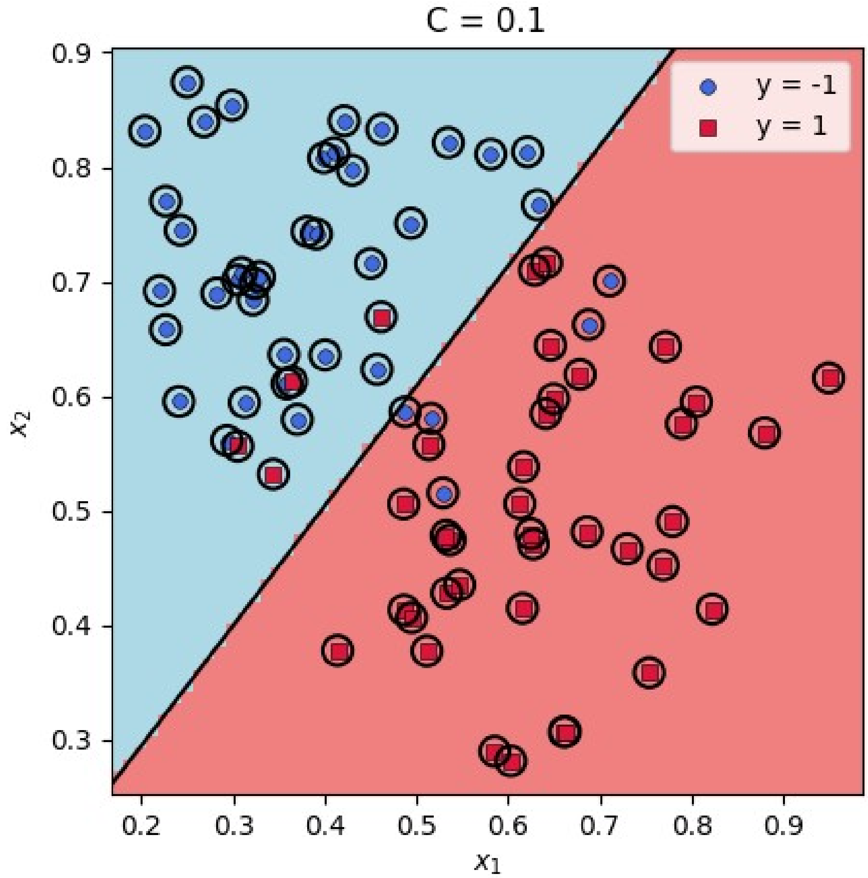


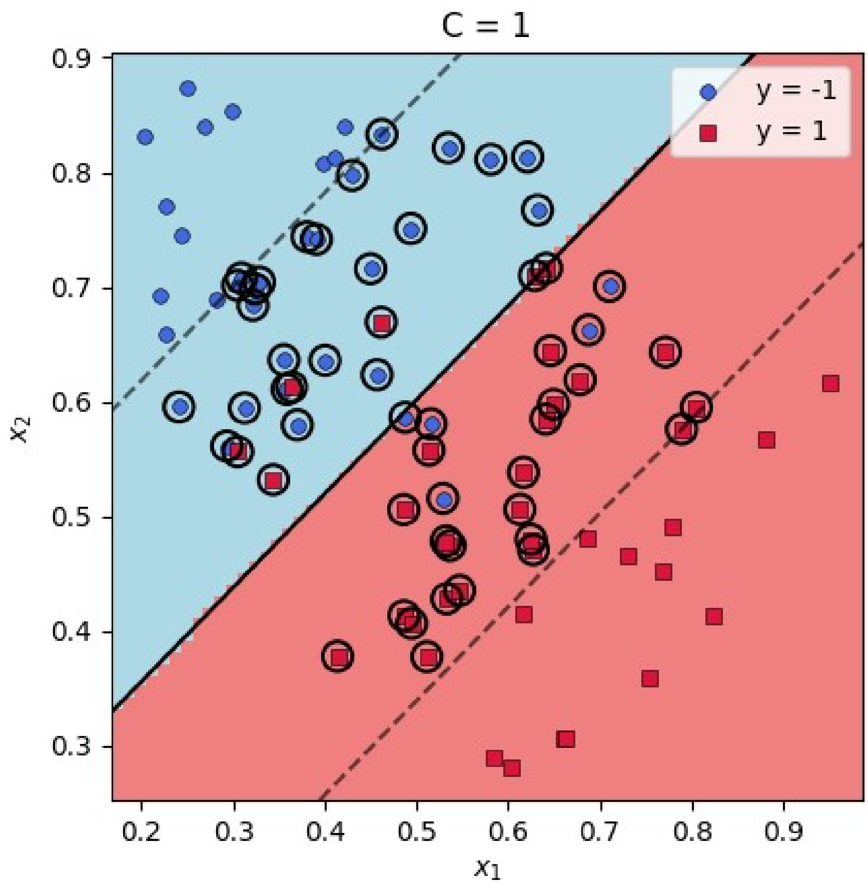
Linearly Non-Separable Dataset

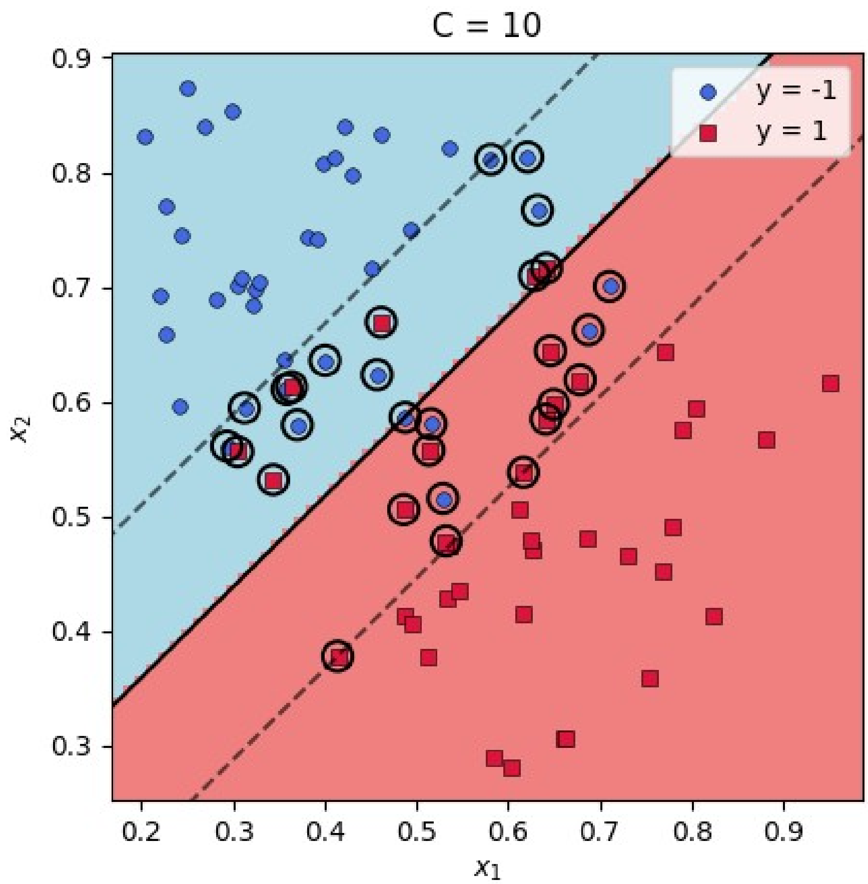
Repeat the above for the linearly non-separable dataset (X2 and y2).

```
C = 5
svm = SVC(C=C,kernel="linear")
svm.fit(X2,y2)
make_plot(f"C = {C}",svm,X2,y2)


for C in [0.1,1,10,1000]:
    svm = SVC(C=C,kernel="linear")
    svm.fit(X2,y2)
    make_plot(f"C = {C}",svm,X2,y2)
```

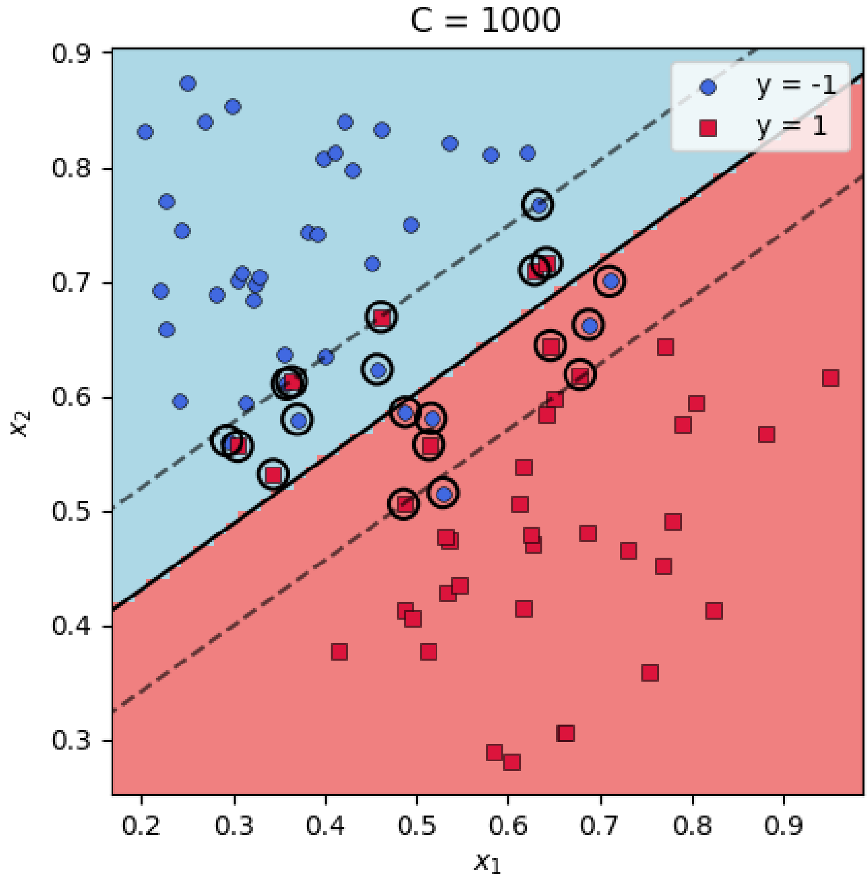

C = 5
y = -1
y = 1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
x_2
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
x_1


C = 0.1
y = -1
y = 1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
x_1
x_2


C = 1
y = -1
y = 1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
x_1
x_2



C = 10
y = -1
y = 1
x_1
x_2
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9



C = 1000
y = -1
y = 1
x_1
x_2
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

M4-L2 Problem 1 (5 points)

Now you will try support vector classification on data with nonlinear decision boundaries. You will use the sklearn SVC tool on four datasets. Your job is to find an appropriate choice of kernel and regularization strength that does a qualitatively good job separating the data.

Run this cell first:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from matplotlib.colors import ListedColormap

# Plotting functions:
def plot_data(X,c,s=30):
    lims = [0,1]

    markers = [dict(marker="o", color="royalblue"), dict(marker="s",
color="red"), dict(marker="^", color="limegreen")]

    x,y = X[:,0], X[:,1]
    iter = 0
    for i in np.unique(c):
        marker = markers[iter]
        iter += 1
        plt.scatter(x[c==i], y[c==i], s=s, **(marker),
edgecolor="black", linewidths=0.4, label="y = " + str(i))

def plot_SVs(svm, s=120):
    sv = svm.support_vectors_
    x, y = sv[:,0], sv[:,1]
    plt.scatter(x, y, s=s, edgecolor="black", facecolor="none",
linewidths=1.5)

def plot_SV_decision_boundary(svm, margin=True,extend=True,
shade_margins=False, shade_decision=False):
    ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    xrange = xlim[1] - xlim[0]
    yrange = ylim[1] - ylim[0]

    x = np.linspace(xlim[0] - extend*xrange, xlim[1] + extend*xrange,
200)
    y = np.linspace(ylim[0] - extend*yrange, ylim[1] + extend*yrange,
200)
```

```
    X,Y = np.meshgrid(x,y)
    xy = np.vstack([X.ravel(), Y.ravel()]).T
    P = svm.decision_function(xy)

    P = P.reshape(X.shape)
    ax.contour(X, Y, P, colors='k',levels=[0],linestyles=['-'])
    if margin:
        ax.contour(X, Y, P, colors='k',levels=[-1, 1],
alpha=0.6,linestyles=['--'])

    if shade_margins:
        cmap = ListedColormap(["white","lightgreen"])

plt.pcolormesh(X,Y,np.abs(P)<1,shading="nearest",cmap=cmap,zorder=-
999999)

    if shade_decision:
        cmap = ListedColormap(["lightblue","lightcoral"])
        pred = (svm.predict(xy).reshape(X.shape) == 1).astype(int)
        plt.pcolormesh(X,Y,pred,shading="nearest",cmap=cmap,zorder=-
1000)

    plt.xlim(xlim)
    plt.ylim(ylim)

def plot(Xdata, ydata, svm_model=None, title=""):
    plt.figure(figsize=(5,5))
    plot_data(Xdata,ydata)
    if svm_model is not None:
        plot_SVs(svm_model)

plot_SV_decision_boundary(svm_model,margin=True,shade_decision=True)
    plt.legend()
    plt.xlabel("$x_1$")
    plt.ylabel("$x_2$")
    plt.title(title)
    plt.show()
```

Loading the data

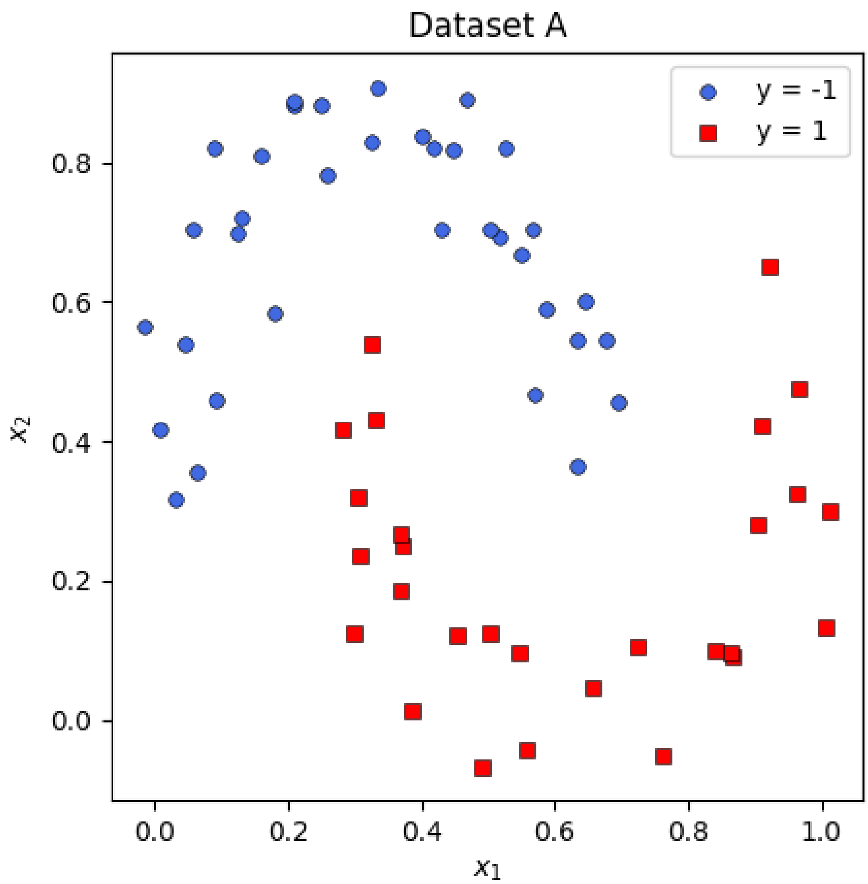
There are four datasets, all 2D and with X and y names as follows:

- Xa, ya
- Xb, yb
- Xc, yc
- Xd, yd

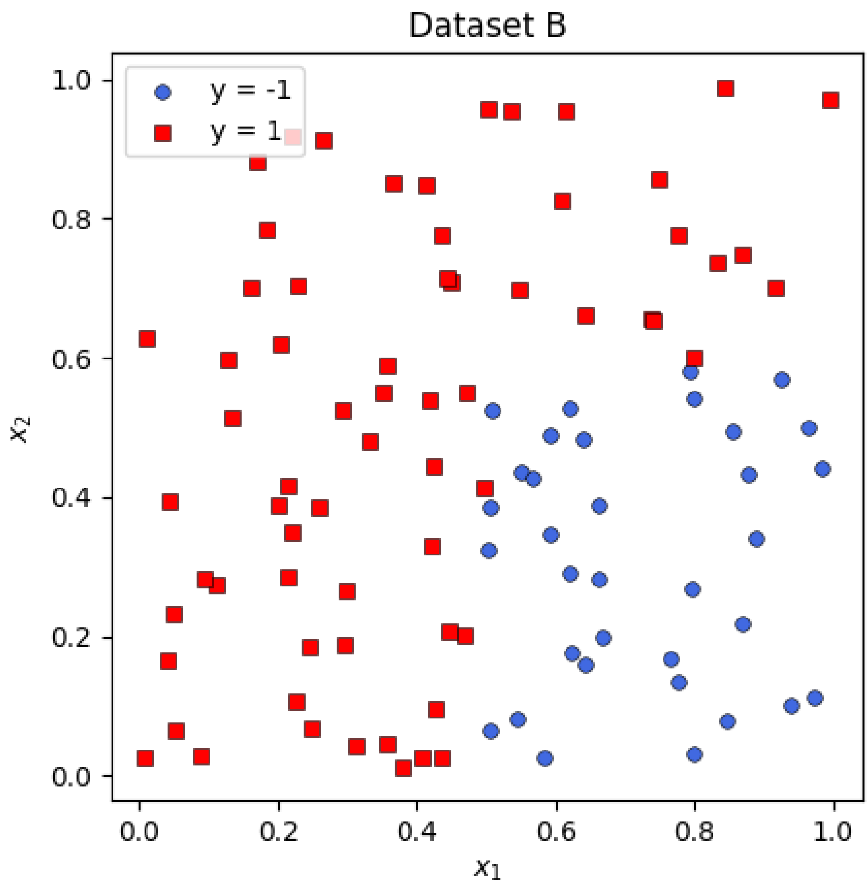
Run this cell to load and plot the data:

```
xa0 = np.array([0.00806452, 0.0467742, -0.0145161, 0.0564516,
0.130645, 0.0887097, 0.208065, 0.25, 0.324194, 0.259677, 0.469355,
0.333871, 0.417742, 0.430645, 0.566129, 0.58871, 0.646774, 0.695161,
0.633871, 0.569355, 0.55, 0.446774, 0.527419, 0.517742, 0.679032,
0.633871, 0.501613, 0.401613, 0.208065, 0.159677, 0.124194, 0.179032,
0.0919355, 0.0629032, 0.0306452])
ya0 = np.array([0.418367, 0.540816, 0.565306, 0.704082, 0.720408,
0.822449, 0.883673, 0.883673, 0.830612, 0.781633, 0.891837, 0.908163,
0.822449, 0.704082, 0.704082, 0.589796, 0.602041, 0.455102, 0.365306,
0.467347, 0.667347, 0.818367, 0.822449, 0.691837, 0.544898, 0.544898,
0.704082, 0.838776, 0.887755, 0.810204, 0.7, 0.585714, 0.459184,
0.357143, 0.316327])
xa1 = np.array([0.324194, 0.304839, 0.372581, 0.369355, 0.453226,
0.546774, 0.724194, 0.866129, 1.00806, 1.01129, 0.91129, 0.904839,
0.840323, 0.656452, 0.501613, 0.369355, 0.330645, 0.282258, 0.308065,
0.298387, 0.385484, 0.491935, 0.762903, 0.559677, 0.862903, 0.962903,
0.966129, 0.920968])
ya1 = np.array([0.540816, 0.320408, 0.25102, 0.185714, 0.120408,
0.0959184, 0.104082, 0.0918367, 0.132653, 0.3, 0.422449, 0.279592,
0.1, 0.0469388, 0.12449, 0.267347, 0.430612, 0.418367, 0.234694,
0.12449, 0.0142857, -0.0673469, -0.0510204, -0.0428571, 0.0959184,
0.32449, 0.47551, 0.65102])
Xa = np.concatenate([np.vstack([xa0,ya0]).T,np.vstack([xa1,ya1]).T],0)
ya = np.array([-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,-
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,])
plot(Xa,ya,title="Dataset A")

xb1 =
np.array([0.43599,0.54966,0.42037,0.20465,0.29965,0.62113,0.13458,0.18
444,0.85398,0.84656,0.50525,0.42812,0.12716,0.22601,0.22031,0.46779,0.
64041,0.50524,0.79364,0.1623,0.96455,0.88952,0.56714,0.43675,0.5356,0.
54421,0.36634,0.40628,0.24718,0.99385,0.80026,0.76496,0.29302,0.35662,
0.98315,0.504,0.25974,0.83202,0.37921,0.7974,0.58268,0.6622,0.49707,0.
35087,0.97291,0.31326,0.7384,0.21464,0.64384,0.17048,0.77801,0.86892,0
.79859,0.22084,0.59208,0.26378,0.41974,0.60844,0.62356,0.59126,0.54791
,0.24581,0.11058,0.01025,0.29517,0.095288,0.21492,0.47141,0.84511,0.04
8868,0.64331,0.87015,0.74176,0.79889,0.22957,0.087563,0.35713,0.052223
,0.043501,0.66843,0.87627,0.61964,0.61525,0.44801,0.42537,0.50762,0.04
2429,0.45023,0.77759,0.50278,0.33187,0.74837,0.41446,0.44457,0.0084484
,0.94045,0.66197,0.20084,0.9259,0.91671,])
xb2 =
np.array([0.025926,0.43532,0.33033,0.61927,0.26683,0.52914,0.51358,0.7
8534,0.49424,0.079645,0.065287,0.096531,0.59675,0.10695,0.34983,0.2017
4,0.48307,0.38689,0.58,0.70075,0.50001,0.34161,0.42755,0.77656,0.95374
,0.082095,0.85085,0.027202,0.067144,0.97058,0.60182,0.16923,0.52407,0.
045679,0.44135,0.32354,0.38689,0.73675,0.013017,0.26939,0.025551,0.387
52,0.41491,0.55098,0.11278,0.041798,0.65751,0.41675,0.66148,0.88165,0.
13395,0.74878,0.54335,0.91846,0.34624,0.91392,0.54019,0.82625,0.17671,
0.48927,0.69952,0.18663,0.27406,0.62936,0.18729,0.28376,0.2856,0.5495,
```

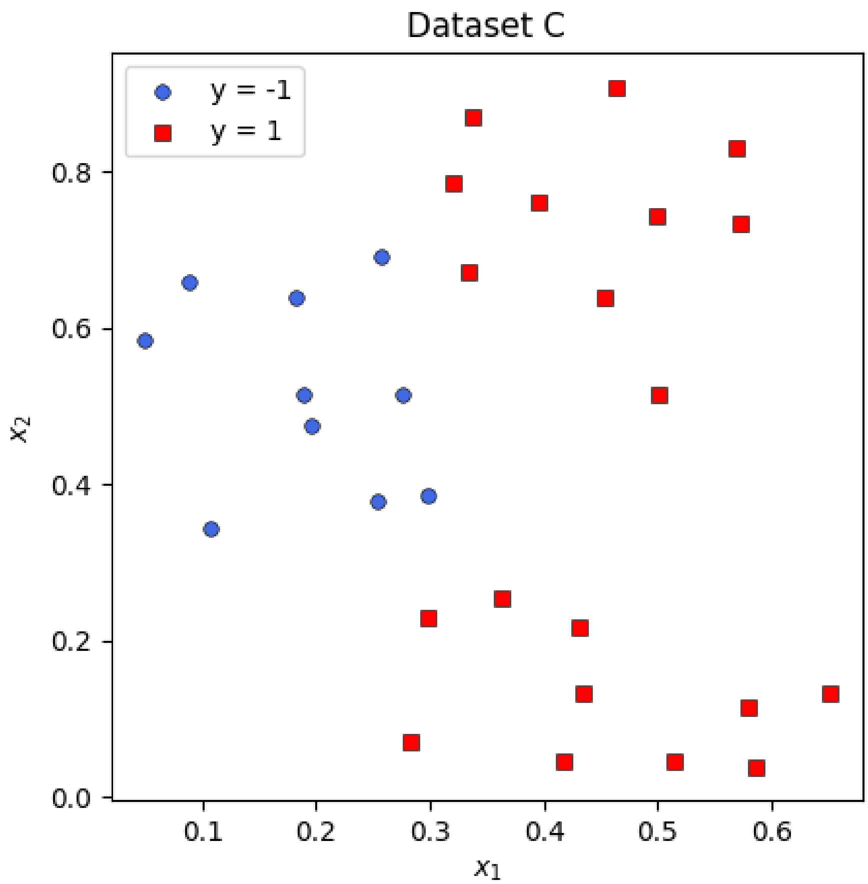
```
0.98851,0.23212,0.16147,0.2174,0.65302,0.031248,0.70463,0.030589,0.589
78,0.065664,0.39515,0.19803,0.43239,0.29043,0.95366,0.20705,0.44457,0.
52573,0.16442,0.70797,0.7771,0.95675,0.48146,0.85784,0.84868,0.71575,0
.025201,0.10214,0.28326,0.38833,0.57027,0.70226,])
yb = np.array([1,-1,1,1,1,-1,1,1,-1,-1,-1,1,1,1,1,1,-1,-1,-1,1,-1,-1,-
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,-1,1,1,1,1,1,1,1,1,1,1,-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,])
Xb = np.vstack([xb1,xb2]).T
plot(Xb,yb,title="Dataset B")

xc1 =
np.array([0.05,0.08871,0.18226,0.18871,0.27581,0.25323,0.10806,0.19516
,0.25645,0.29839,0.3371,0.4629,0.49839,0.33387,0.39516,0.32097,0.45323
,0.56936,0.50161,0.57258,0.41774,0.58548,0.43387,0.28226,0.29839,0.362
9,0.43065,0.57903,0.51452,0.65,])
xc2 =
np.array([0.58571,0.65918,0.63878,0.51633,0.51633,0.37755,0.3449,0.475
51,0.69184,0.38571,0.87143,0.90816,0.7449,0.67143,0.76122,0.78571,0.63
878,0.83061,0.51633,0.73265,0.046939,0.038775,0.13265,0.071429,0.23061
,0.2551,0.21837,0.11633,0.046939,0.13265,])
Xc = np.vstack([xc1,xc2]).T
yc = np.array([-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,])
plot(Xc,yc,title="Dataset C")

xd1 =
np.array([0.062903,0.08871,0.18548,0.33065,0.54032,0.68226,0.81129,0.9
1129,0.92742,0.88548,0.80807,0.6371,0.38226,0.21452,0.13387,0.098387,0
.12097,0.21452,0.30806,0.49839,0.66613,0.74032,0.83387,0.86935,0.81452
,0.69839,0.54032,0.38871,0.25,0.17258,0.12742,0.43065,0.79839,0.6371,0
.27903,0.19516,0.40484,0.48871,0.61129,0.71129,0.7371,0.72097,0.61774,
0.56936,0.47903,0.36935,0.25323,0.21452,0.26936,0.34355,0.41129,0.5661
3,0.60161,0.53387,0.36935,0.29839,])
xd2 =
np.array([0.52041,0.67143,0.77347,0.88367,0.92857,0.83469,0.64286,0.54
898,0.3898,0.17347,0.079592,0.022449,0.026531,0.079592,0.20204,0.36122
,0.58571,0.72041,0.78571,0.84286,0.82653,0.67959,0.50816,0.37347,0.197
96,0.07551,0.071429,0.12041,0.13673,0.27551,0.49592,0.85918,0.56122,0.
083673,0.20204,0.5898,0.72041,0.72041,0.66326,0.55306,0.43061,0.29184,
0.19796,0.25102,0.31224,0.30408,0.31224,0.43061,0.56122,0.67551,0.6265
3,0.56122,0.40204,0.37755,0.40612,0.44694,])
Xd = np.vstack([xd1,xd2]).T
yd =
np.array([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,])
plot(Xd,yd,title="Dataset D")
```

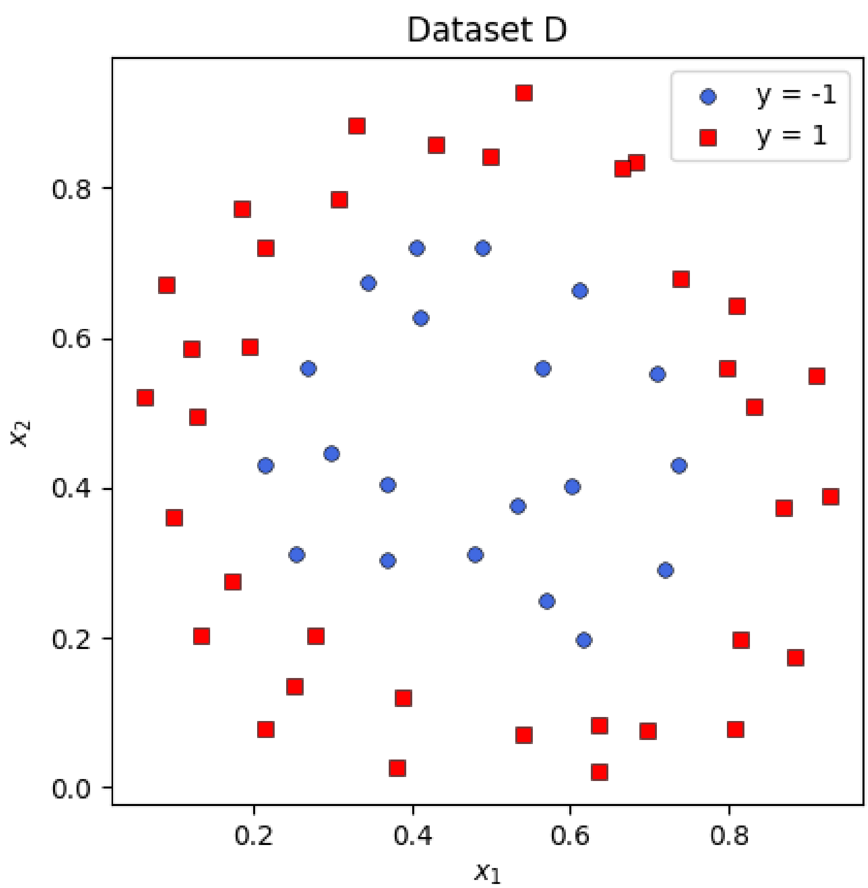


Dataset A
y = -1
y = 1
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
x_1
x_2



Dataset B
y = -1
y = 1
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
x_2
x_1


Dataset C
y = -1
y = 1
0.0
0.2
0.4
0.6
0.8
0.1
0.2
0.3
0.4
0.5
0.6
x_1
x_2



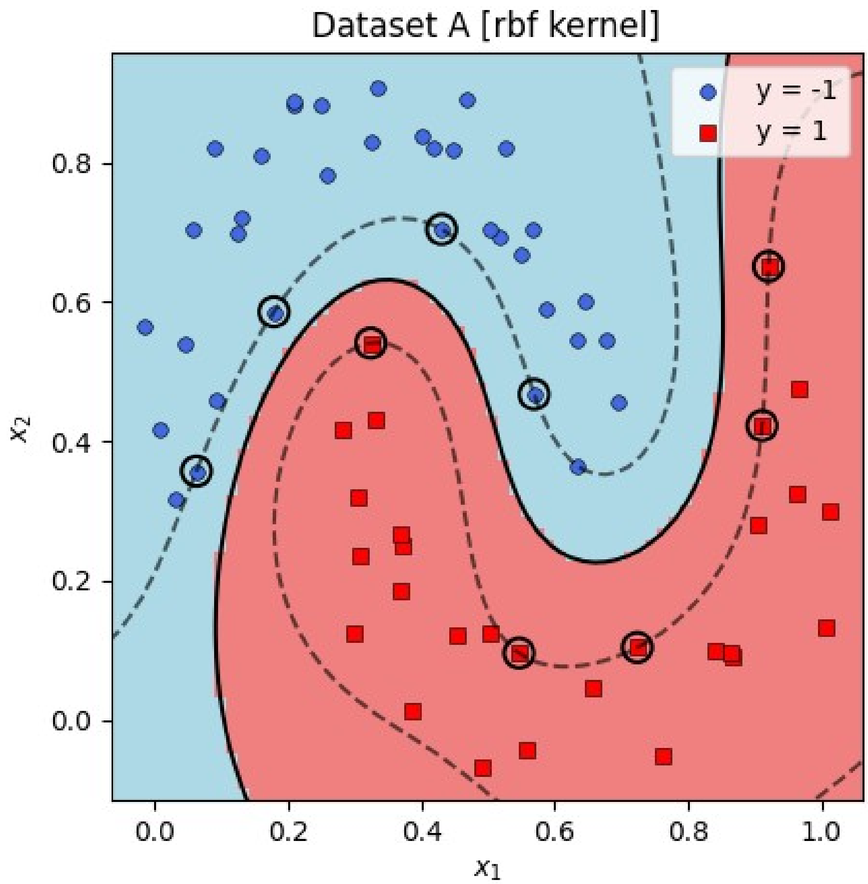
Using the Kernel Trick

Now, train four SVC models, one for each dataset. Try out different combinations of 'kernel' and 'C', until you find a satisfactory classifier in each case.

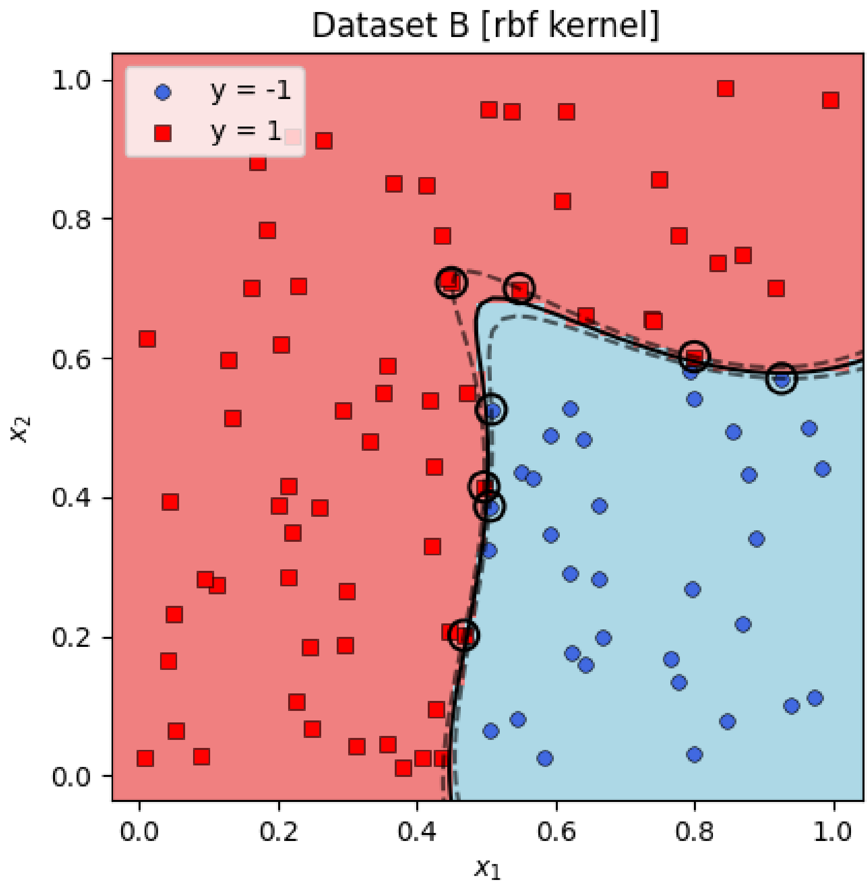
Please generate a plot for each dataset showing the results of a trained support vector classifier, using the provided function:

`plot(Xdata, ydata, svm_model, title)`

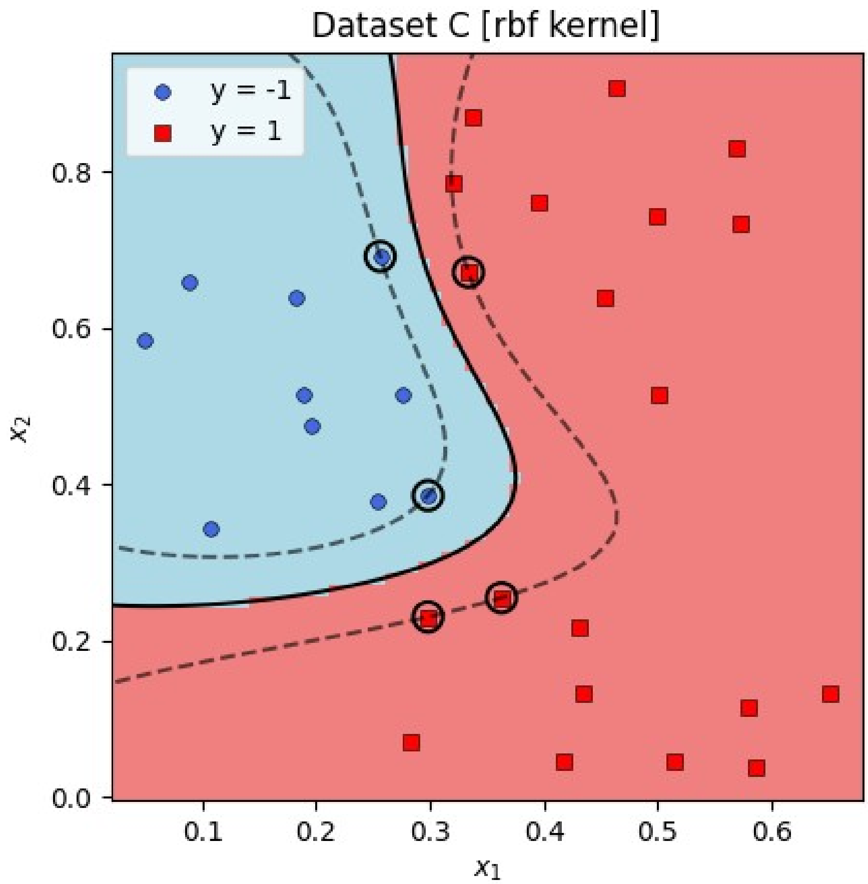
```
# YOUR CODE GOES HERE
# different kernels: linear, poly, rbf, sigmoid
# svm = SVC(kernel="linear", C=1e10)
# svm = SVC(kernel="poly", C=1e10)
# svm = SVC(kernel="rbf", C=1e10)
# svm = SVC(kernel="sigmoid", C=1e10)

# (Dataset A)
svm = SVC(kernel="rbf", C=1e10)
svm.fit(Xa,ya)
plot(Xa,ya,svm,"Dataset A [{kernel}
kernel]".format(kernel=svm.kernel))
```



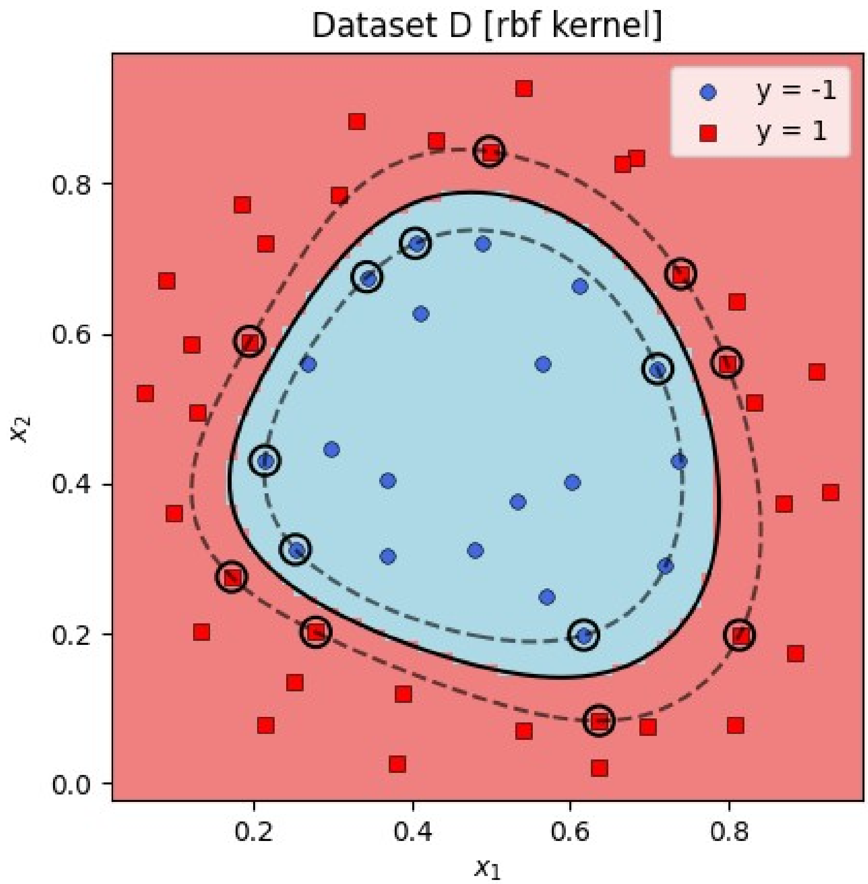

```
# YOUR CODE GOES HERE
# (Dataset B)
svm = SVC(kernel="rbf", C=1e10)
svm.fit(Xb,yb)
plot(Xb,yb,svm,"Dataset B [{kernel}
kernel]".format(kernel=svm.kernel))
```




```
# YOUR CODE GOES HERE
# (Dataset C)
# svm = SVC(kernel="linear", C=1e10)
svm = SVC(kernel="rbf", C=1e10)     # This is better compared to
linear kernel
svm.fit(Xc,yc)
plot(Xc,yc,svm,"Dataset C [{kernel}
kernel]".format(kernel=svm.kernel))
```




```
# YOUR CODE GOES HERE
# (Dataset D)
svm = SVC(kernel="rbf", C=1e10)
svm.fit(Xd,yd)
plot(Xd,yd,svm,"Dataset D [{kernel}
kernel]".format(kernel=svm.kernel))
```


Dataset D [rbf kernel]
y = -1
y = 1
0.0
0.2
0.4
0.6
0.8
x_2
0.2
0.4
0.6
0.8
x_1

M4-L2 Problem 2 (5 points)

Here we will revisit the phase diagram problem from the logistic regression module. Your task will be to code a one-vs-rest support vector classifier.

Work through this notebook, filling in code as requested, to implement the OvR classifier.

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.svm import SVC

x1 = np.array([7.4881350392732475,16.351893663724194,22.427633760716436,29.04883182996897,35.03654799338904,44.45894113066656,6.375872112626925,18.117730007820796,26.036627605010292,27.434415188257777,38.71725038082664,43.28894919752904,7.680445610939323,18.45596638292661,17.110360581978867,24.47129299701541,31.002183974403255,46.32619845547938,9.781567509498505,17.90012148246819,26.186183422327638,31.59158564216724,35.41479362252932,45.805291762864556,3.182744258689332,15.599210213275237,17.833532874090462,33.04668917049584,36.018483217500716,42.146619399905234,4.64555612104627,16.942336894342166,20.961503322165484,29.284339488686488,30.98789800436355,44.17635497075877,])
x2 = np.array([0.11120957227224215,0.1116933996874757,0.14437480785146242,0.11818202991034835,0.0859507900573786,0.09370319537993416,0.2797631195927265,0.216022547162927,0.27667667154456677,0.27706378696181594,0.2310382561073841,0.22289262976548535,0.40154283509241845,0.4063710770942623,0.427019677041788,0.41386015134623205,0.46883738380592266,0.38020448107480287,0.5508876756094834,0.5461309517884996,0.5953108325465398,0.5553291602539782,0.5766310772856306,0.5544425592001603,0.705896958364552,0.7010375141164304,0.7556329589465274,0.7038182951348614,0.7096582361680054,0.7268725170660963,0.9320993229847936,0.8597101275793062,0.9337944907498804,0.8596098407893963,0.9476459465013396,0.8968651201647702,])
X = np.vstack([x1,x2]).T
y = np.array([0,2,2,2,2,2,0,2,2,2,2,2,0,0,2,0,1,2,0,0,1,1,1,2,0,1,0,1,1,1,0,0,1,1,1,1,])


def plot_data(X, y, title="Phase of simulated material", newfig=True):
    xlim = [0,52.5]
    ylim = [0,1.05]
    markers = [dict(marker="o", color="royalblue"), dict(marker="s", color="crimson"), dict(marker="^", color="limegreen")]
    labels = ["Solid", "Liquid", "Vapor"]
```

```
    if newfig:
        plt.figure(dpi=150)

    for i in range(1+max(y)):
        plt.scatter(X[y==i,0], X[y==i,1], s=60, **(markers[i]),
edgecolor="black", linewidths=0.4,label=labels[i])

    plt.title(title)
    plt.legend(loc="upper right")
    plt.xlim(xlim)
    plt.ylim(ylim)
    plt.xlabel("Temperature, K")
    plt.ylabel("Pressure, atm")
    plt.box(True)

def plot_ovr_colors(classifiers, res=40):
    xlim = [0,52.5]
    ylim = [0,1.05]
    xvals = np.linspace(*xlim,res)
    yvals = np.linspace(*ylim,res)
    x,y = np.meshgrid(xvals,yvals)
    XY = np.concatenate((x.reshape(-1,1),y.reshape(-1,1)),axis=1)
    if type(classifiers) == list:
        color = classify_ovr(classifiers,XY).reshape(res,res)
    else:
        color = classifiers(XY).reshape(res,res)
    cmap = ListedColormap(["lightblue","lightcoral","palegreen"])
    plt.pcolor(x, y, color, shading="nearest", zorder=-1,
cmap=cmap,vmin=0,vmax=2)
    return
```

Binomial classification function

You are given a function that performs binomial classification by using sklearn's `SVC` tool: `prob = get_ovr_decision_function(X, y, A, kernel, C)`

To use it, input:

- `X`, an array in which each row contains (x,y) coordinates of data points
- `y`, an array that specifies the class each point in `X` belongs to
- `A`, the class of the group (0, 1, or 2 in this problem) -- classifies into A or "rest"
- `kernel`, the kernel to use for the SVM
- `C`, the inverse regularization strength to use for the SVM

The function outputs a decision function (`decision()` in this case), which can be used to evaluate each `X`, giving positive values for class A, and negative values for [not A].

```
def get_ovr_decision_function(X, y, A, kernel="linear",C=1000):
    y_new = -1 + 2*(y == A).astype(int)
```

```
    model = SVC(kernel=kernel,C=C)
    model.fit(X, y_new)

    def decision(X):
        pred = model.decision_function(X)
        return pred.flatten()

    return decision
```

Coding an OvR classifier

Now you will create a one-vs-rest classifier to do multinomial classification. This will generate a binomial classifier for each class in the dataset, when compared against the rest of the classes. Then to predict the class of a new point, classify it using each of the binomial classifiers, and select the class whose binomial classifier decision function returns the highest value.

Complete the two functions we have started:

- `generate_ovr_decision_functions(X, y)` which returns a list of binary classifier probability functions for all possible classes (0, 1, and 2 in this problem)
- `classify_ovr(decisions, X)` which loops through a list of ovr classifiers and gets the decision function evaluation for each point in `X`. Then taking the highest decision function value for each, return the overall class predictions for each point.

```
def generate_ovr_decision_functions(X, y, kernel="linear", C=1000):
    # YOUR CODE GOES HERE
    decisions = []
    for i in range(3):
        decision = get_ovr_decision_function(X, y, i, kernel, C)
        decisions.append(decision)
    return decisions

def classify_ovr(decisions, X):
    # YOUR CODE GOES HERE
    pred = np.zeros((X.shape[0],3))
    for i in range(3):
        pred[:,i] = decisions[i](X)
    pred = np.argmax(pred,axis=1)

    return pred
```

Testing the classifier

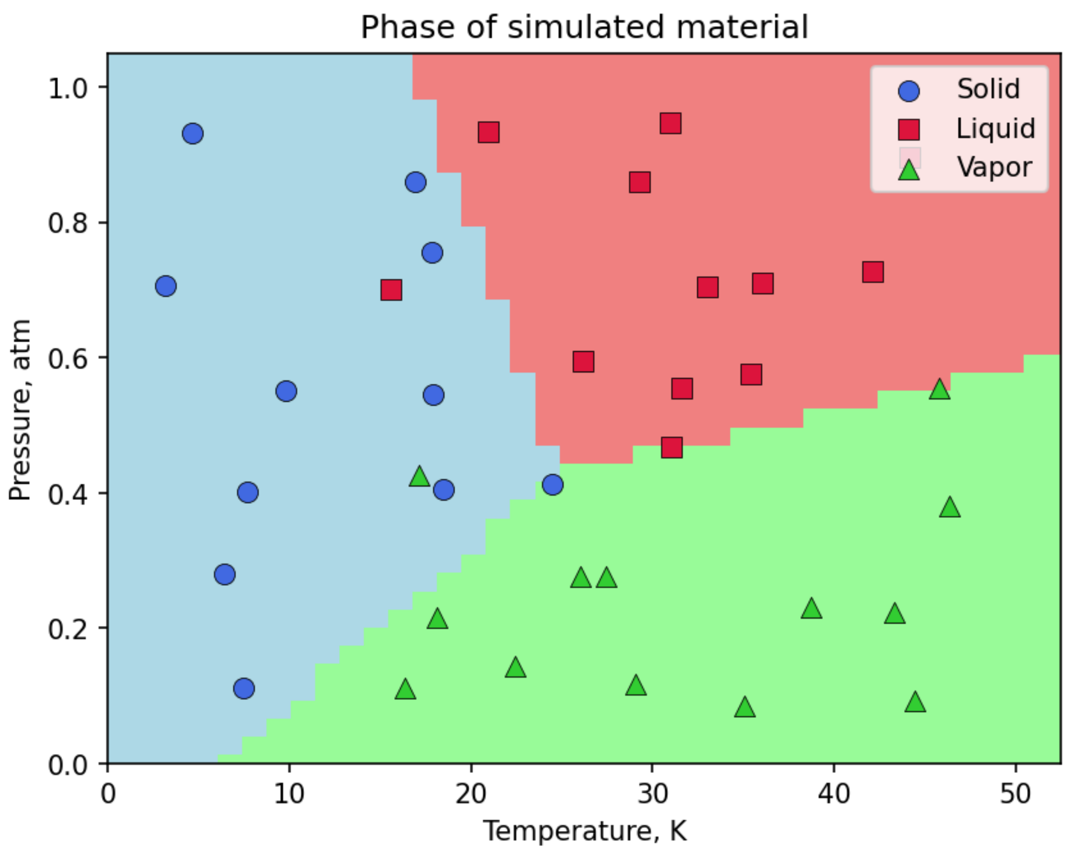
```
kernel = "linear"
C = 1000

decisions = generate_ovr_decision_functions(X, y, kernel, C)
pred = classify_ovr(decisions, X)
```

```
accuracy = np.sum(pred == y) / len(y) * 100
print("True Classes:", y)
print(" Predictions:", pred)
print("    Accuracy:", accuracy, r"%")
```

```
True Classes: [0 2 2 2 2 2 0 2 2 2 2 2 0 0 2 0 1 2 0 0 1 1 1 2 0 1 0 1
1 1 0 0 1 1 1 1]
 Predictions: [0 2 2 2 2 2 0 2 2 2 2 2 0 0 0 2 1 2 0 0 1 1 1 2 0 0 0 1
1 1 0 0 1 1 1 1]
    Accuracy: 91.66666666666666 %
```

Plotting results

```
plot_data(X,y)
plot_ovr_colors(decisions)
plt.show()
```



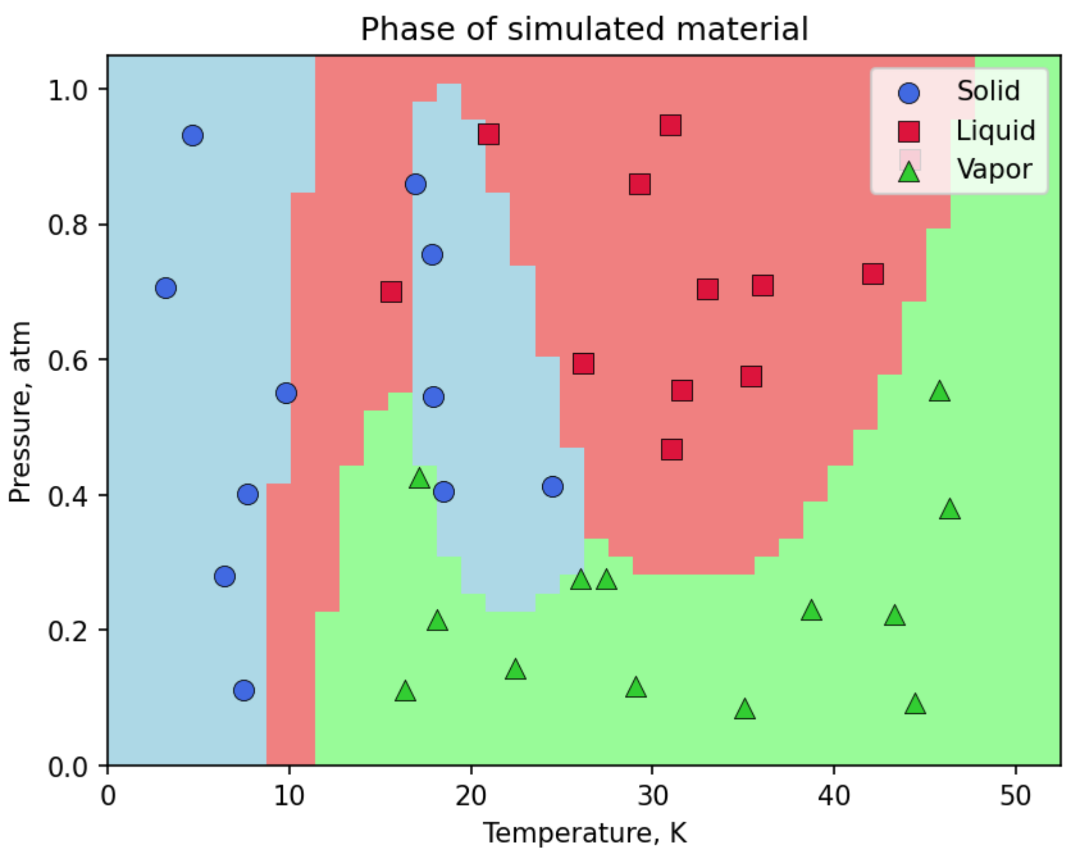
Modifying the SVC

Now go back and change the kernel and C value; observe how the results change.

```
kernel = "rbf"  # CHANGE THIS
C = 1e10    # CHANGE THIS
```

```python
# different kernels: linear, poly, rbf, sigmoid
# svm = SVC(kernel="linear", C=1e10)
# svm = SVC(kernel="poly", C=1e10)
# svm = SVC(kernel="rbf", C=1e10)
# svm = SVC(kernel="sigmoid", C=1e10)

decisions = generate_ovr_decision_functions(X, y, kernel, C)
preds = classify_ovr(decisions, X)
accuracy = np.sum(preds == y) / len(y) * 100
print("True Classes:", y)
print(" Predictions:", preds)
print("    Accuracy:", accuracy, r"%")

plot_data(X,y)
plot_ovr_colors(decisions)
plt.show()
```

```
True Classes: [0 2 2 2 2 2 0 2 2 2 2 2 0 0 2 0 1 2 0 0 1 1 1 2 0 1 0 1
1 1 0 0 1 1 1 1]
 Predictions: [0 2 2 2 2 2 0 2 2 2 2 2 0 0 2 0 1 2 0 0 1 1 1 2 0 1 0 1
1 1 0 0 1 1 1 1]
    Accuracy: 100.0 %
```



M4-L2 Problem 3 (5 points)

In this problem, we will investigate kernel selection and regularization strength in support vector regression for a 1-D problem.

Run each cell below, then try out the interactive plot to answer the questions.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVR

xs =
np.array([0.094195,0.10475,0.12329,0.12767,0.1343,0.11321,0.16134,0.16
622,0.15704,0.16892,0.1707,0.19564,0.18697,0.20818,0.22071,0.21833,0.2
3029,0.23398,0.25217,0.25168,0.2538,0.25143,0.27121,0.27319,0.28675,0.
29971,0.30451,0.32319,0.32141,0.33977,0.35378,0.37053,0.35916,0.36534,
0.3807,0.38696,0.41073,0.41095,0.41302,0.42177,0.42517,0.43633,0.42191
,0.45198,0.4606,0.4838,0.4664,0.48132,0.49296,0.51028,0.51747,0.499,0.
49948,0.53049,0.53986,0.55444,0.54966,0.56389,0.5544,0.56139,0.58974,0
.59864,0.59467,0.6122,0.61911,0.62601,0.63302,0.63993,0.65452,0.64038,
0.67782,0.66911,0.67807,0.68518,0.68705,0.70398,0.72397,0.71793,0.7293
1,0.76366,0.75441,0.73797,0.7741,0.77121,0.77784,0.7816,0.79257,0.8046
9,0.82256,0.82495,0.83913,0.8226,0.84766,0.83838,0.8493,0.89643,0.8678
3,0.89621,0.90823,0.90054,])
ys =
np.array([0.51123,0.50881,0.50546,0.50756,0.51653,0.50797,0.49658,0.50
899,0.50218,0.50242,0.50906,0.50466,0.48063,0.49306,0.48622,0.51558,0.
50493,0.48378,0.518,0.49348,0.51459,0.53657,0.54106,0.54207,0.56463,0.
56601,0.61192,0.61208,0.63699,0.64194,0.67329,0.70949,0.74668,0.77664,
0.82362,0.84736,0.89991,0.91268,0.92689,0.93635,0.94732,0.95202,0.9411
2,0.92713,0.89726,0.88055,0.83289,0.78465,0.75197,0.71588,0.64221,0.58
237,0.52391,0.45466,0.37946,0.31505,0.25479,0.18915,0.14154,0.084572,0
.058735,0.027538,0.013328,0.0098045,0.068816,0.094916,0.10225,0.16912,
0.21646,0.27493,0.33072,0.40278,0.48282,0.53813,0.63165,0.69685,0.7449
4,0.8089,0.8693,0.89515,0.92841,0.94583,0.93489,0.91862,0.92811,0.9004
7,0.86258,0.85054,0.82246,0.83096,0.78313,0.74352,0.71369,0.69591,0.65
134,0.65297,0.61356,0.59983,0.57448,0.56923,])

x_gt =
np.array([0.0,0.010101,0.020202,0.030303,0.040404,0.050505,0.060606,0.
070707,0.080808,0.090909,0.10101,0.11111,0.12121,0.13131,0.14141,0.151
52,0.16162,0.17172,0.18182,0.19192,0.20202,0.21212,0.22222,0.23232,0.2
4242,0.25253,0.26263,0.27273,0.28283,0.29293,0.30303,0.31313,0.32323,0
.33333,0.34343,0.35354,0.36364,0.37374,0.38384,0.39394,0.40404,0.41414
,0.42424,0.43434,0.44444,0.45455,0.46465,0.47475,0.48485,0.49495,0.505
05,0.51515,0.52525,0.53535,0.54545,0.55556,0.56566,0.57576,0.58586,0.5
9596,0.60606,0.61616,0.62626,0.63636,0.64646,0.65657,0.66667,0.67677,0
```

```
.68687,0.69697,0.70707,0.71717,0.72727,0.73737,0.74747,0.75758,0.76768
,0.77778,0.78788,0.79798,0.80808,0.81818,0.82828,0.83838,0.84848,0.858
59,0.86869,0.87879,0.88889,0.89899,0.90909,0.91919,0.92929,0.93939,0.9
4949,0.9596,0.9697,0.9798,0.9899,1.0,])
y_gt =
np.array([0.46193,0.47566,0.48699,0.49609,0.50315,0.50836,0.51189,0.51
393,0.51467,0.51428,0.51294,0.51085,0.50818,0.50512,0.50186,0.49856,0.
49542,0.49263,0.49035,0.48878,0.4881,0.4885,0.49015,0.49323,0.49794,0.
50446,0.51298,0.52376,0.53706,0.55316,0.57231,0.59478,0.62084,0.65075,
0.68477,0.72317,0.76529,0.80864,0.85051,0.88819,0.91898,0.94015,0.9491
7,0.94553,0.93,0.90339,0.86651,0.82017,0.76518,0.70233,0.63243,0.5563,
0.47475,0.38966,0.3049,0.22456,0.15274,0.093526,0.051005,0.028929,0.02
7469,0.044659,0.078502,0.127,0.18816,0.25999,0.34048,0.42761,0.51845,0
.60913,0.69574,0.77438,0.84113,0.89208,0.92416,0.93858,0.93795,0.92487
,0.90197,0.87185,0.83712,0.80039,0.76426,0.73054,0.69893,0.66883,0.639
63,0.61072,0.5815,0.55136,0.51968,0.48587,0.44931,0.40939,0.36551,0.31
706,0.26344,0.20402,0.13821,0.065402,])

%matplotlib inline
from ipywidgets import interact, interactive, fixed, interact_manual,
Layout, FloatSlider, Dropdown


def plotting_function(kernel, log_C, log_epsilon):
    C = np.power(10.,log_C)
    epsilon = np.power(10.,log_epsilon)

    model = SVR(kernel=kernel,C=C,epsilon=epsilon)
    model.fit(xs.reshape(-1,1),ys)

    xfit = np.linspace(0,1,200)
    yfit = model.predict(xfit.reshape(-1,1))

    plt.figure(figsize=(12,7))
    plt.scatter(xs,ys,s=10,c="k",label="Data")
    plt.plot(xfit,yfit,linewidth=3, label="SVR")
    plt.plot(x_gt,y_gt,"--",label="Ground Truth")
    title = f"Kernel: {kernel},  C = {C:.1e}, eps = {epsilon:.1e}"
    plt.legend(loc="lower left")
    plt.xlabel("$x_1$")
    plt.ylabel("$y$")
    plt.title(title)
    plt.show()

slider1 = FloatSlider(
    value=0,
    min=-5,
    max=5,
    step=.5,
    description='C',
```

```
    disabled=False,
    continuous_update=True,
    orientation='horizontal',
    readout=False,
    layout = Layout(width='550px')
)

slider2 = FloatSlider(
    value=-1,
    min=-7,
    max=-1,
    step=.5,
    description='epsilon',
    disabled=False,
    continuous_update=True,
    orientation='horizontal',
    readout=False,
    layout = Layout(width='550px')
)

dropdown = Dropdown(
    options=['linear', 'rbf','sigmoid'],
    value='linear',
    description='kernel',
    disabled=False,
)


interactive_plot = interactive(
    plotting_function,
    kernel = dropdown,
    log_C = slider1,
    log_epsilon = slider2
    )
output = interactive_plot.children[-1]
output.layout.height = '500px'

interactive_plot
```

{"model_id":"c92043177fc9463ca5f0e17b2f90d79e","version_major":2,"version_minor":0}

Questions

1. Which kernel produced the best fit overall? (Assume this kernel for subsequent questions.)

   The "RBF" kernel produced the best fit overall.

2. As 'C' increases, does model performance on in-sample data generally improve or worsen?

   As C increased, the model performance on in-sample data generally improved.

3. As 'C' increases, does model performance on out-of-sample data (on the intervals [0.0, 0.1] and [0.9, 1.0]) generally improve or worsen?

   As C increased, the model performance on out-of-sample data generally worsened.

4. What 'C' value would you recommend for this kernel?

   From the interactive plot, I would recommend the C value to be 3.2e3 for this kernel.

5. What 'epsilon' value would you recommend?

   From the interactive plot, I would recommend the epsilon value to be 3.2e-3 for this kernel.