Problem 1: Support Vector Machines

Instructions:

1. Please use this q1.ipynb file to complete SVM using cvxopt
2. You may create new cells for discussions or visualizations

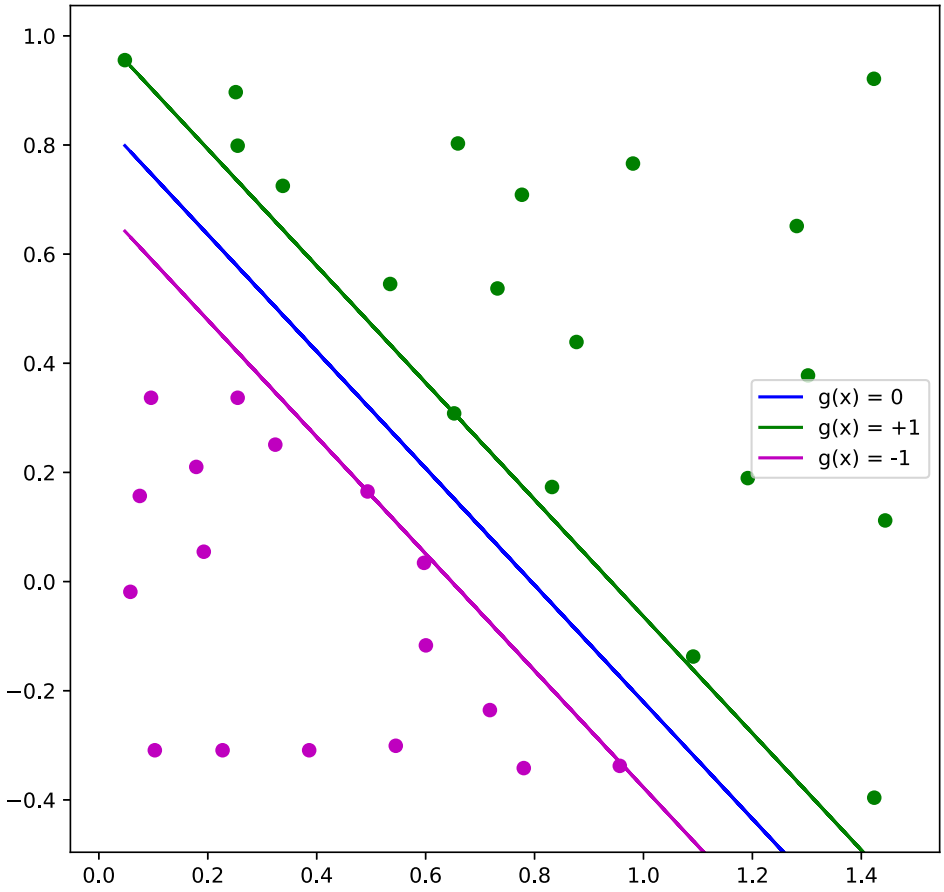
In [1]:
```
# Import modules
import numpy as np
import matplotlib.pyplot as plt
from cvxopt import matrix, solvers
```

a): Linearly Separable Dataset

In [64]:
```
data = np.loadtxt('clean_lin.txt', delimiter='\t')
x = data[:, 0:2]
y = data[:, 2]

""" Assign color per label, then scatter plot dataset """
pred_color = {1:'g', -1:'m'}
colors = [pred_color[i] for i in y]
plt.figure(figsize = (7.5, 7.5))
plt.scatter(x[:, 0], x[:, 1], c=colors)

""" Modifying y to work with NumPy broadcasting,
concatenating a stack of ones onto x for bias term """
y = y[:, np.newaxis]
x = np.hstack((x, np.ones_like(y)))

""" Quadratic programming """
Q = matrix([[0.5, 0, 0], [0, 0.5, 0], [0, 0, 0]], tc='d')
p = matrix([0, 0, 0], tc='d')
G = matrix(-y * x, tc='d')
h = matrix(-1*np.ones_like(y), tc='d')
sol = solvers.qp(Q, p, G, h)
w1, w2, b = sol['x'][0], sol['x'][1], sol['x'][2]

""" Plot boundary and margin boundaries """
plt.plot(x[:, 0], (-w1*x[:, 0]-b)/w2, c='b', label='g(x) = 0')
plt.plot(x[:, 0], (-w1*x[:, 0]-b+1)/w2, c='g', label='g(x) = +1')
plt.plot(x[:, 0], (-w1*x[:, 0]-b-1)/w2, c='m', label='g(x) = -1')
plt.xlim((x[:, 0].min()-0.1, x[:, 0].max()+0.1))
plt.ylim((x[:, 1].min()-0.1, x[:, 1].max()+0.1))
plt.legend()
plt.show()
```

```
     pcost       dcost       gap    pres   dres
0:  8.7483e-01  3.5428e+01  1e+02  2e+00  3e+01
1:  8.4879e+00 -3.2104e+00  4e+01  6e-01  9e+00
2:  1.6650e+01  1.1151e+01  1e+01  1e-01  2e+00
3:  2.1484e+01  1.6526e+01  6e+00  4e-02  5e-01
4:  2.1785e+01  2.0581e+01  2e+00  9e-03  1e-01
5:  2.1870e+01  2.1694e+01  2e-01  2e-04  3e-03
```

```
 6:  2.1867e+01  2.1801e+01  7e-02  5e-05  7e-04
 7:  2.1861e+01  2.1860e+01  1e-03  8e-07  1e-05
 8:  2.1861e+01  2.1861e+01  1e-05  8e-09  1e-07
 9:  2.1861e+01  2.1861e+01  1e-07  8e-11  1e-09
Optimal solution found.
```



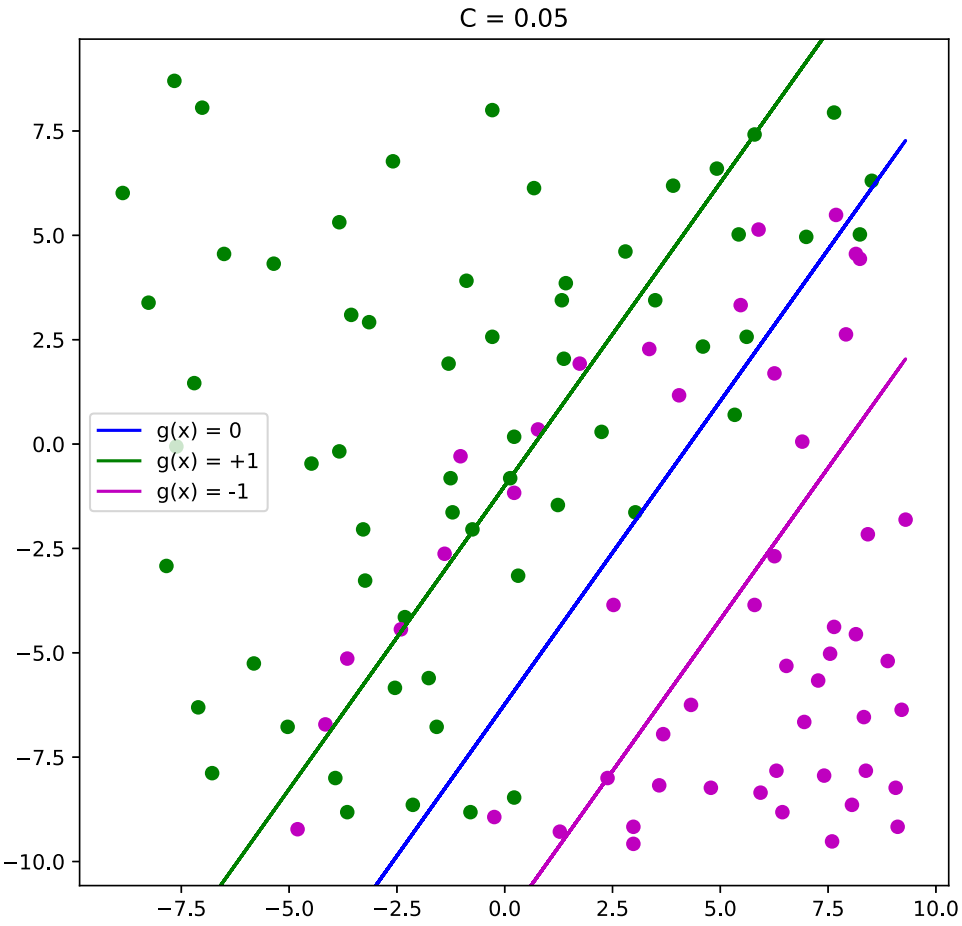
b): Linearly Non-separable Dataset

In [66]:

```
# Load the data set that is not linearly separable
data = np.loadtxt('dirty_nonlin.txt', delimiter='\t')
x = data[:, 0:2]
y = data[:, 2]

""" Assign color per label, then scatter plot dataset """
pred_color = {1:'g', -1:'m'}
colors = [pred_color[i] for i in y]
plt.figure(figsize = (7.5, 7.5))
plt.scatter(x[:, 0], x[:, 1], c=colors)

""" Modifying y to work with NumPy broadcasting,
concatenating a stack of ones onto x for bias term """
y = y[:, np.newaxis]
x = np.hstack((x, np.ones_like(y)))
```

```
""" Quadratic programming """
Q = np.zeros((len(y)+3, len(y)+3))
Q[0:2, 0:2] = 0.5*np.eye(2)
Q = matrix(Q, tc='d')

C = 0.05
p = C*np.ones(len(y))[np.newaxis, :]
p = matrix(np.hstack((np.zeros((1, 3)), p)).T, tc='d')

G = np.block([[-y*x, -1*np.eye(len(y))],
              [np.zeros((len(y), 3)), -1*np.eye(len(y))]])
G = matrix(G, tc='d')

h = matrix(np.vstack((-1*np.ones_like(y), np.zeros_like(y))), tc='d')

sol = solvers.qp(Q, p, G, h)
w1, w2, b = sol['x'][0], sol['x'][1], sol['x'][2]

""" Plot boundary and margin boundaries """
plt.plot(x[:, 0], (-w1*x[:, 0]-b)/w2, c='b', label='g(x) = 0')
plt.plot(x[:, 0], (-w1*x[:, 0]-b+1)/w2, c='g', label='g(x) = +1')
plt.plot(x[:, 0], (-w1*x[:, 0]-b-1)/w2, c='m', label='g(x) = -1')
plt.xlim((x[:, 0].min()-1, x[:, 0].max()+1))
plt.ylim((x[:, 1].min()-1, x[:, 1].max()+1))
plt.title("C = {}".format(C))
plt.legend()
plt.show()
```

```
     pcost       dcost       gap    pres   dres
 0:  1.2369e+00  3.9714e+01  6e+02  2e+00  7e+02
 1:  1.0368e+01 -1.5829e+01  3e+01  8e-02  2e+01
 2:  6.1832e+00  7.5736e-02  6e+00  1e-02  3e+00
 3:  2.6957e+00  1.9060e+00  8e-01  1e-03  3e-01
 4:  2.3622e+00  2.1461e+00  2e-01  3e-04  8e-02
 5:  2.3003e+00  2.2049e+00  1e-01  7e-05  2e-02
 6:  2.2686e+00  2.2290e+00  4e-02  2e-05  6e-03
 7:  2.2515e+00  2.2425e+00  9e-03  3e-06  7e-04
 8:  2.2470e+00  2.2462e+00  8e-04  1e-07  3e-05
 9:  2.2466e+00  2.2466e+00  9e-06  1e-09  3e-07
10:  2.2466e+00  2.2466e+00  9e-08  1e-11  3e-09
Optimal solution found.
```

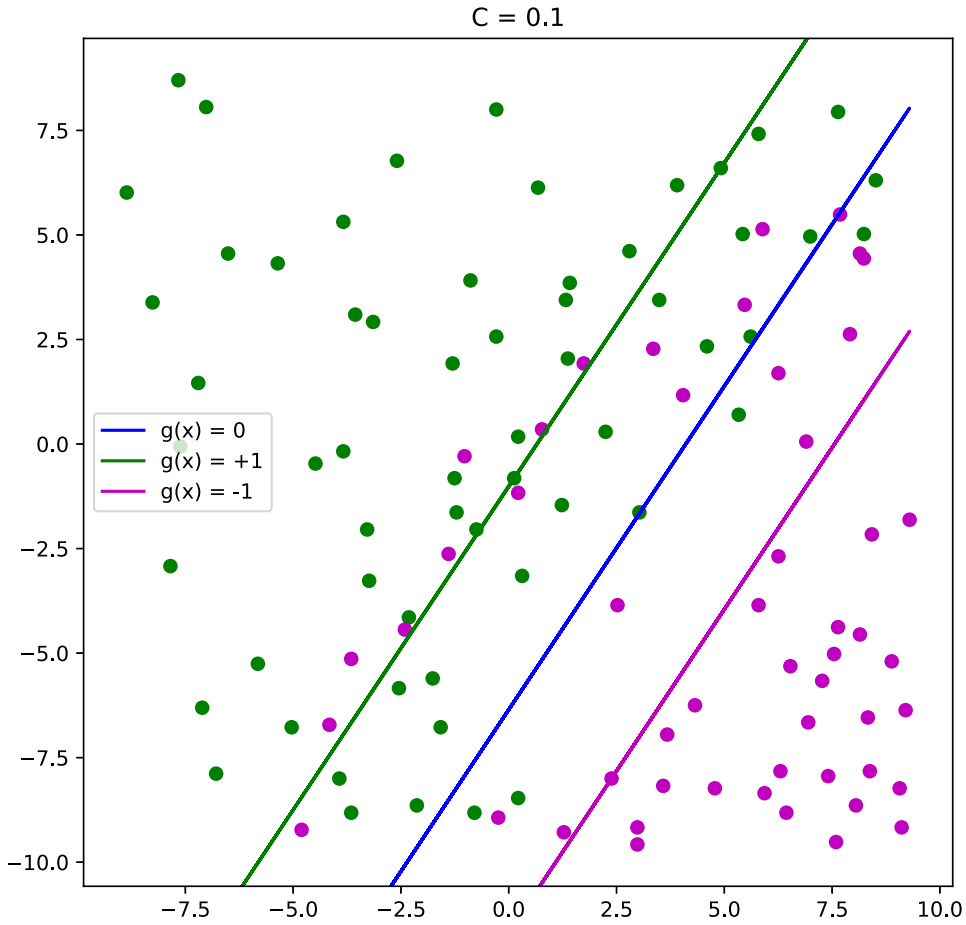


c): Output 4 plots & Explain your observations here:

In [71]:

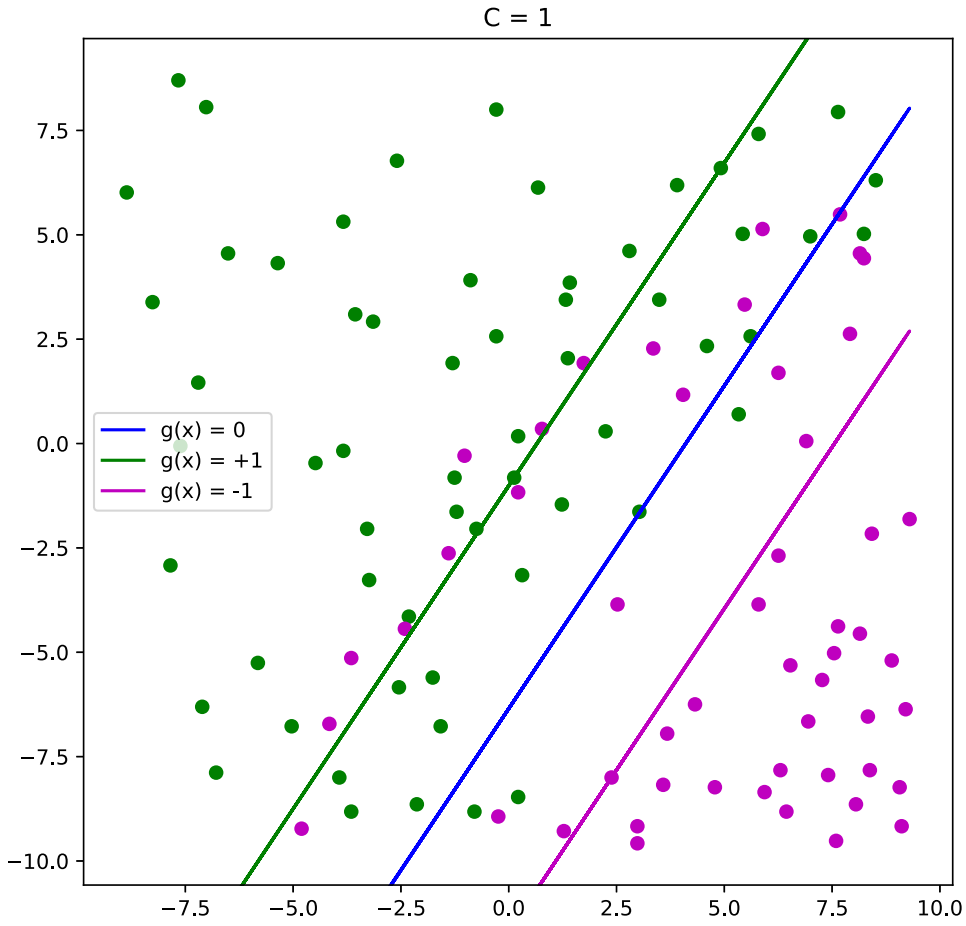
```
C = [0.1, 1, 100, 1000000]
for i in range(len(C)):
    plt.figure(i, figsize = (7.5, 7.5))
    plt.title("C = {}".format(C[i]))
    p = C[i]*np.ones(len(y))[np.newaxis, :]
    p = matrix(np.hstack((np.zeros((1, 3)), p)).T, tc='d')

    sol = solvers.qp(Q, p, G, h)
    w1, w2, b = sol['x'][0], sol['x'][1], sol['x'][2]

    plt.scatter(x[:, 0], x[:, 1], c=colors)
    plt.plot(x[:, 0], (-w1*x[:, 0]-b)/w2, c='b', label='g(x) = 0')
    plt.plot(x[:, 0], (-w1*x[:, 0]-b+1)/w2, c='g', label='g(x) = +1')
    plt.plot(x[:, 0], (-w1*x[:, 0]-b-1)/w2, c='m', label='g(x) = -1')
    plt.xlim((x[:, 0].min()-1, x[:, 0].max()+1))
    plt.ylim((x[:, 1].min()-1, x[:, 1].max()+1))
    plt.legend()
    plt.show()
```

```
     pcost       dcost       gap    pres   dres
 0:  2.0586e+00  5.0520e+01  7e+02  3e+00  6e+02
 1:  1.9677e+01 -2.3176e+01  5e+01  1e-01  3e+01
 2:  1.1129e+01  1.0321e+00  1e+01  2e-02  4e+00
 3:  5.1180e+00  3.8916e+00  1e+00  2e-03  5e-01
 4:  4.6561e+00  4.2742e+00  4e-01  5e-04  1e-01
 5:  4.5701e+00  4.3758e+00  2e-01  2e-04  5e-02
 6:  4.5183e+00  4.4169e+00  1e-01  6e-05  2e-02
 7:  4.4859e+00  4.4444e+00  4e-02  2e-05  6e-03
 8:  4.4679e+00  4.4615e+00  6e-03  8e-07  2e-04
 9:  4.4647e+00  4.4644e+00  3e-04  3e-08  8e-06
10:  4.4645e+00  4.4645e+00  3e-06  3e-10  9e-08
Optimal solution found.
```

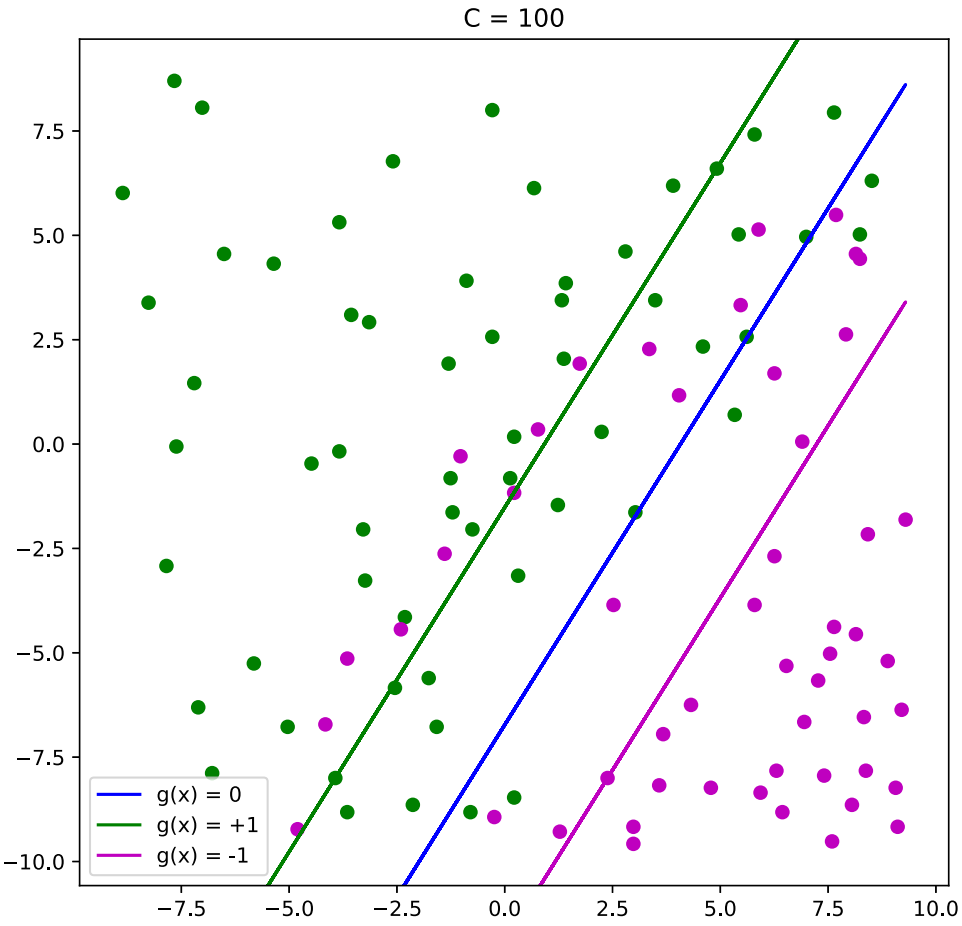



```
     pcost       dcost       gap    pres   dres
 0: -5.3462e+01  2.5872e+02  1e+03  4e+00  7e+01
 1:  1.3078e+02 -3.0840e+01  2e+02  5e-01  8e+00
 2:  5.6284e+01  3.4499e+01  2e+01  3e-02  5e-01
 3:  4.9731e+01  4.0109e+01  1e+01  1e-02  2e-01
 4:  4.6211e+01  4.2662e+01  4e+00  4e-03  7e-02
 5:  4.5260e+01  4.3579e+01  2e+00  1e-03  2e-02
 6:  4.4780e+01  4.4030e+01  8e-01  5e-04  8e-03
 7:  4.4567e+01  4.4205e+01  4e-01  2e-04  3e-03
 8:  4.4427e+01  4.4336e+01  9e-02  4e-05  6e-04
 9:  4.4408e+01  4.4349e+01  6e-02  2e-05  3e-04
```

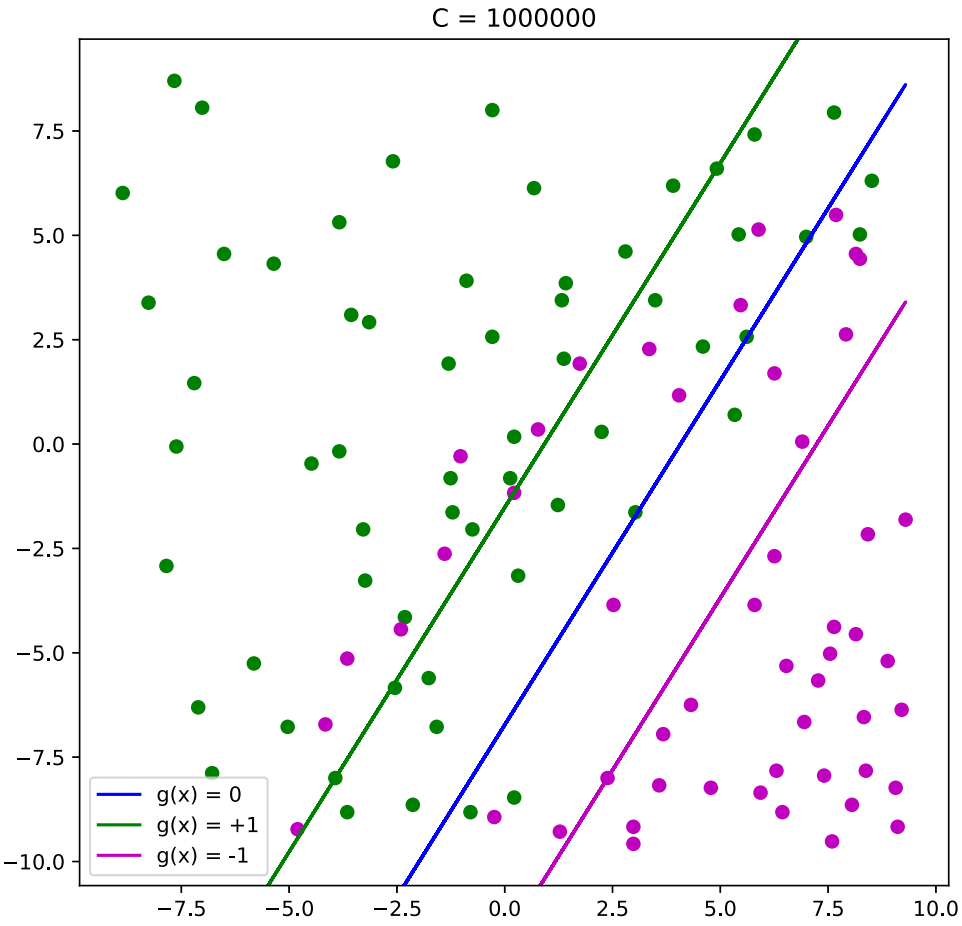
```
10:  4.4385e+01  4.4371e+01  1e-02  4e-16  8e-14
11:  4.4378e+01  4.4377e+01  2e-04  4e-16  1e-12
12:  4.4377e+01  4.4377e+01  2e-06  4e-16  3e-13
Optimal solution found.
```




```
    pcost       dcost       gap    pres   dres
 0: -8.1948e+05  4.6956e+05  2e+06  2e+02  2e+01
 1:  1.4936e+05 -6.1257e+03  2e+05  4e+00  3e-01
 2:  7.9786e+03  2.6759e+03  6e+03  1e-01  8e-03
 3:  5.3173e+03  3.5802e+03  2e+03  3e-02  2e-03
 4:  5.1659e+03  3.8326e+03  1e+03  2e-02  1e-03
 5:  4.7745e+03  4.1018e+03  7e+02  8e-03  6e-04
 6:  4.6479e+03  4.2535e+03  4e+02  4e-03  3e-04
 7:  4.4923e+03  4.3953e+03  1e+02  2e-04  2e-05
 8:  4.4686e+03  4.4071e+03  6e+01  7e-05  5e-06
 9:  4.4355e+03  4.4342e+03  1e+00  9e-07  7e-08
10:  4.4348e+03  4.4347e+03  5e-02  3e-08  3e-09
11:  4.4348e+03  4.4348e+03  8e-04  4e-10  4e-11
Optimal solution found.
```



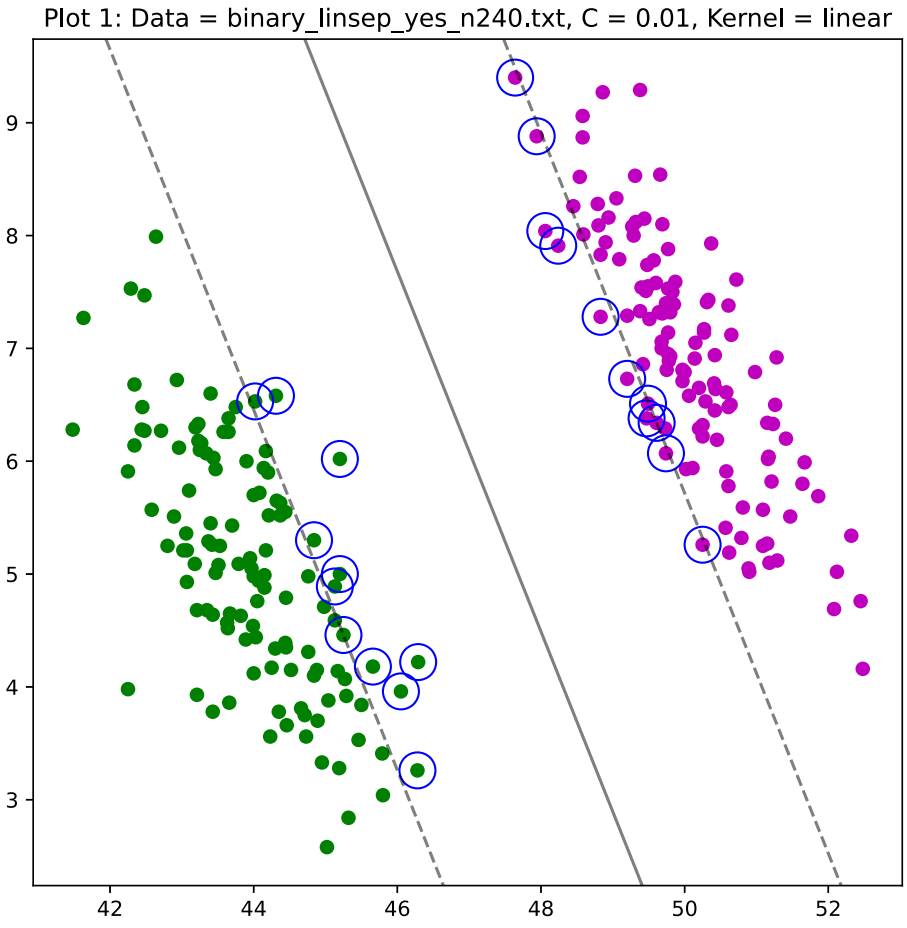

```
     pcost       dcost       gap    pres   dres
 0: -8.2235e+13  4.5105e+13  2e+14  2e+06  2e+01
 1:  1.4250e+13 -6.9400e+11  2e+13  3e+04  2e-01
 2:  2.4964e+11 -1.7815e+08  3e+11  4e+02  3e-03
 3:  2.5672e+09  2.6019e+07  3e+09  4e+00  3e-05
 4:  9.4700e+07  2.6383e+07  8e+07  1e-01  8e-07
 5:  5.5257e+07  3.4069e+07  2e+07  3e-02  2e-07
 6:  5.2212e+07  3.7734e+07  2e+07  2e-02  1e-07
 7:  4.8927e+07  4.0549e+07  9e+06  7e-03  5e-08
 8:  4.6421e+07  4.2681e+07  4e+06  2e-03  2e-08
 9:  4.4985e+07  4.3923e+07  1e+06  7e-05  5e-10
10:  4.4770e+07  4.4021e+07  8e+05  3e-05  2e-10
11:  4.4415e+07  4.4293e+07  1e+05  5e-06  4e-11
12:  4.4349e+07  4.4346e+07  3e+03  1e-07  8e-13
13:  4.4347e+07  4.4347e+07  2e+02  6e-09  2e-12
14:  4.4347e+07  4.4347e+07  2e+00  6e-11  2e-12
Optimal solution found.
```

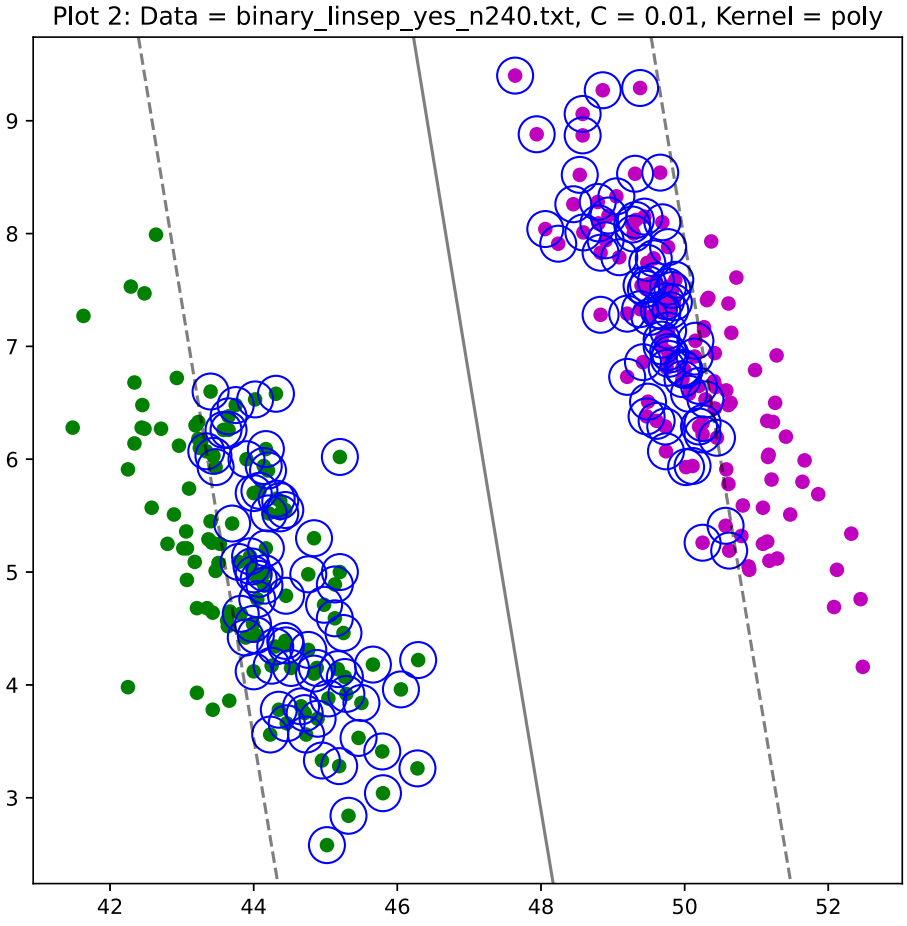

C = 1000000
g(x) = 0
g(x) = +1
g(x) = -1
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

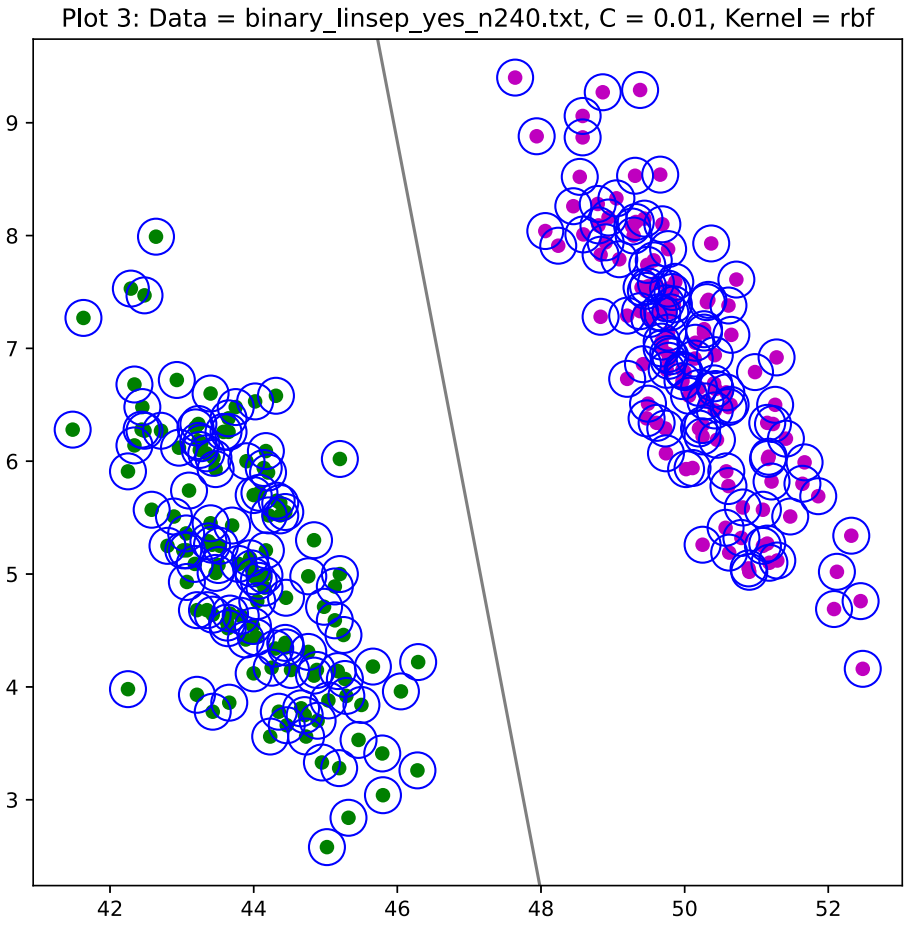
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from plot_svc_decision_function import plot_svc_decision_function
```

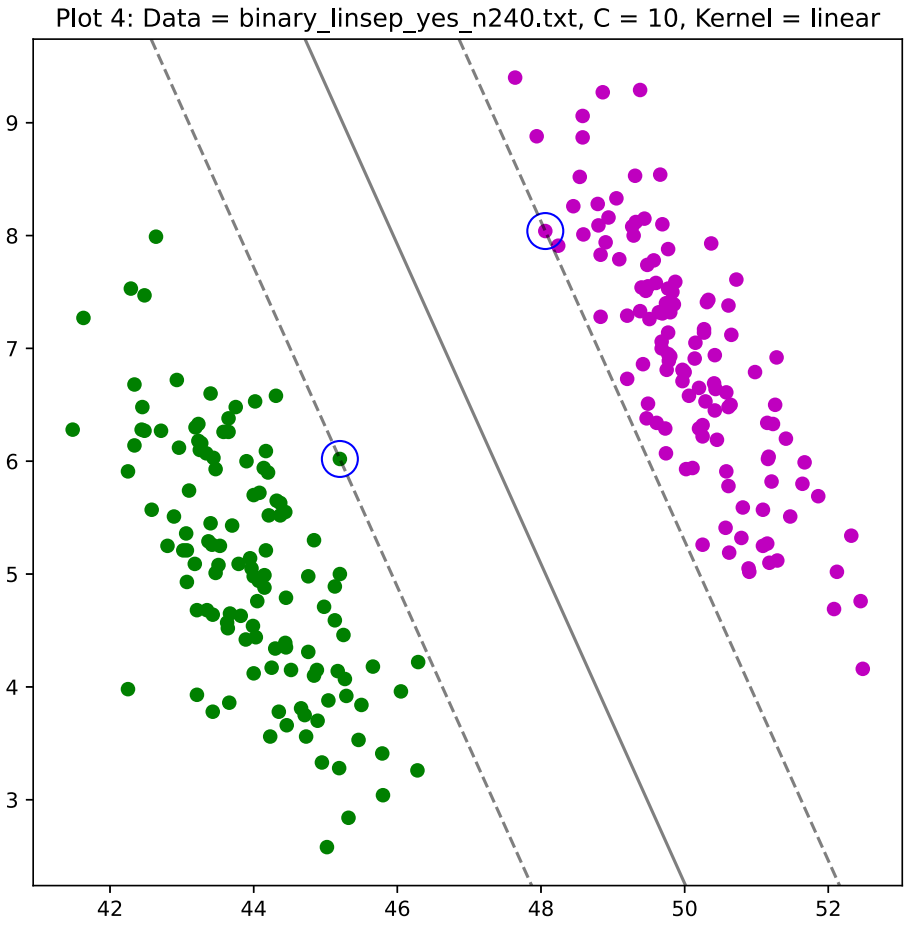
Problem 2a: Support Vector Machine with sklearn

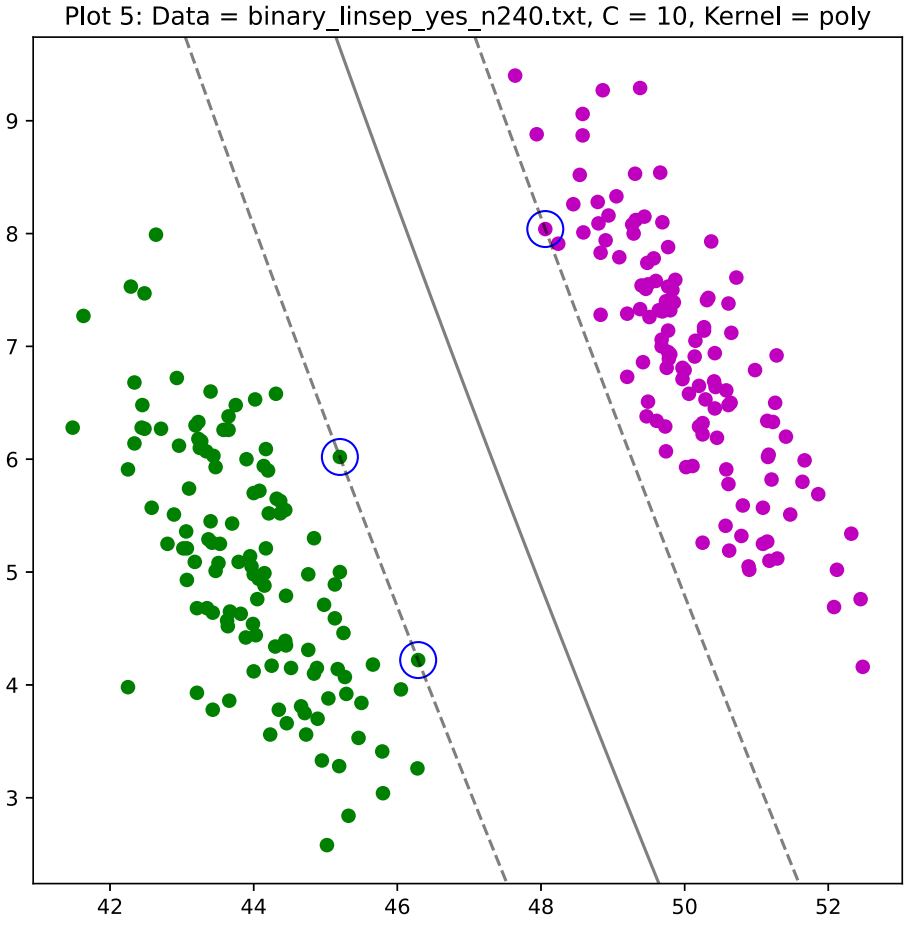
In [2]:
```
file_list = ["binary_linsep_yes_n240.txt", "binary_linsep_no_n240.txt",
             "binary_moons_linsep_no_n100.txt", "concentriccircles_binary.txt"]
kernels = ["linear", "poly", "rbf"]
Cs = [0.01, 10, 1000]
count = 0

for i in range(len(file_list)):
    data = np.genfromtxt(file_list[i], delimiter=' ')
    X = data[:, 0:2]
    y = data[:, 2]
    pred_color = {0:'m', 1:'g'}
    colors = [pred_color[i] for i in y]
    for j in range(len(Cs)):
        for k in range(len(kernels)):
            model = SVC(kernel=kernels[k], C=Cs[j]).fit(X, y)
            plt.figure(count, figsize = (7.5, 7.5))
            plt.scatter(X[:, 0], X[:, 1], c=colors)
            plt.title("Plot {}: Data = {}, C = {}, Kernel = {}".format(count+1, file_li
            plot_svc_decision_function(model)
            if i == 2 and j == 2 and k == 2:
                 SV = model.support_vectors_
            count += 1
print("Support vector values (Shape = {}) for case of interest:\n".format(SV.shape))
print(SV)
```

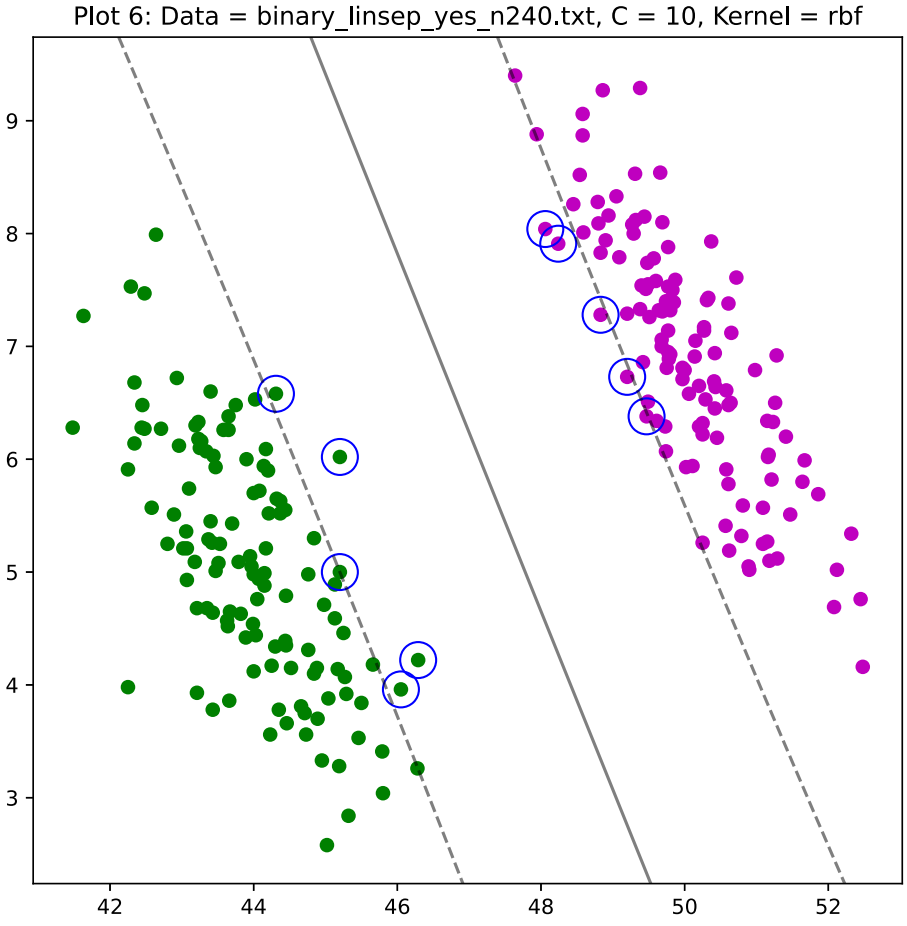

Plot 1: Data = binary_linsep_yes_n240.txt, C = 0.01, Kernel = linear
9
8
7
6
5
4
3
42
44
46
48
50
52

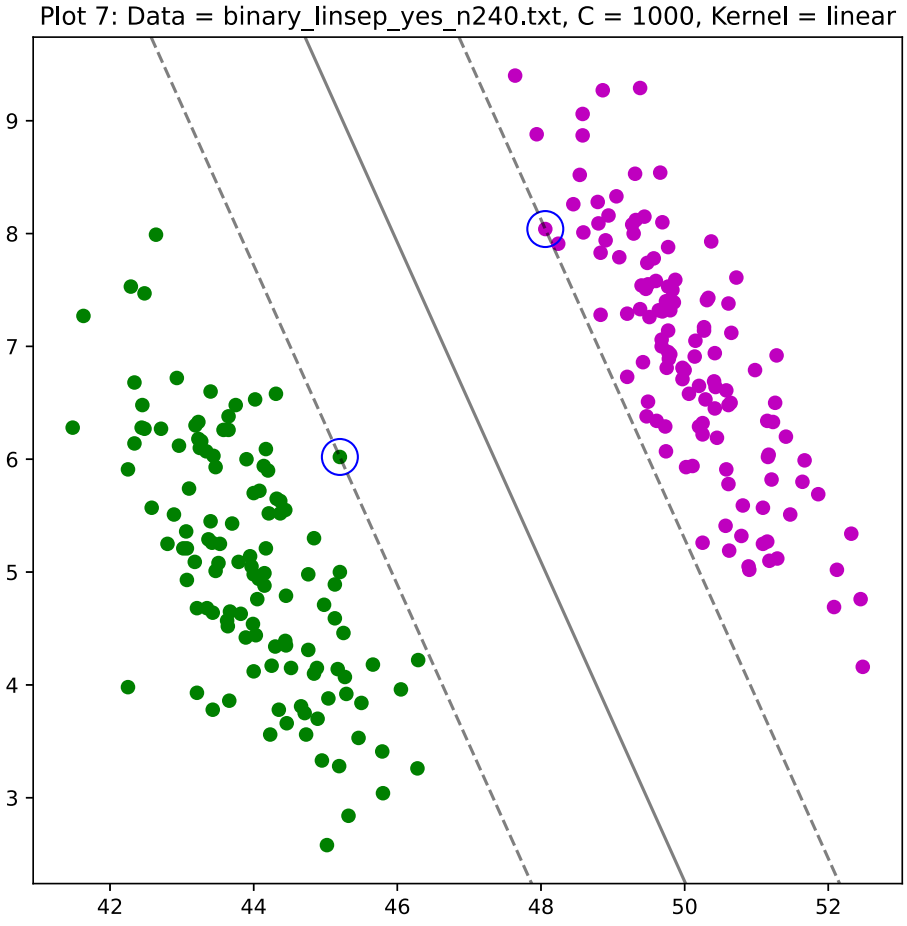

Plot 2: Data = binary_linsep_yes_n240.txt, C = 0.01, Kernel = poly
9
8
7
6
5
4
3
42
44
46
48
50
52

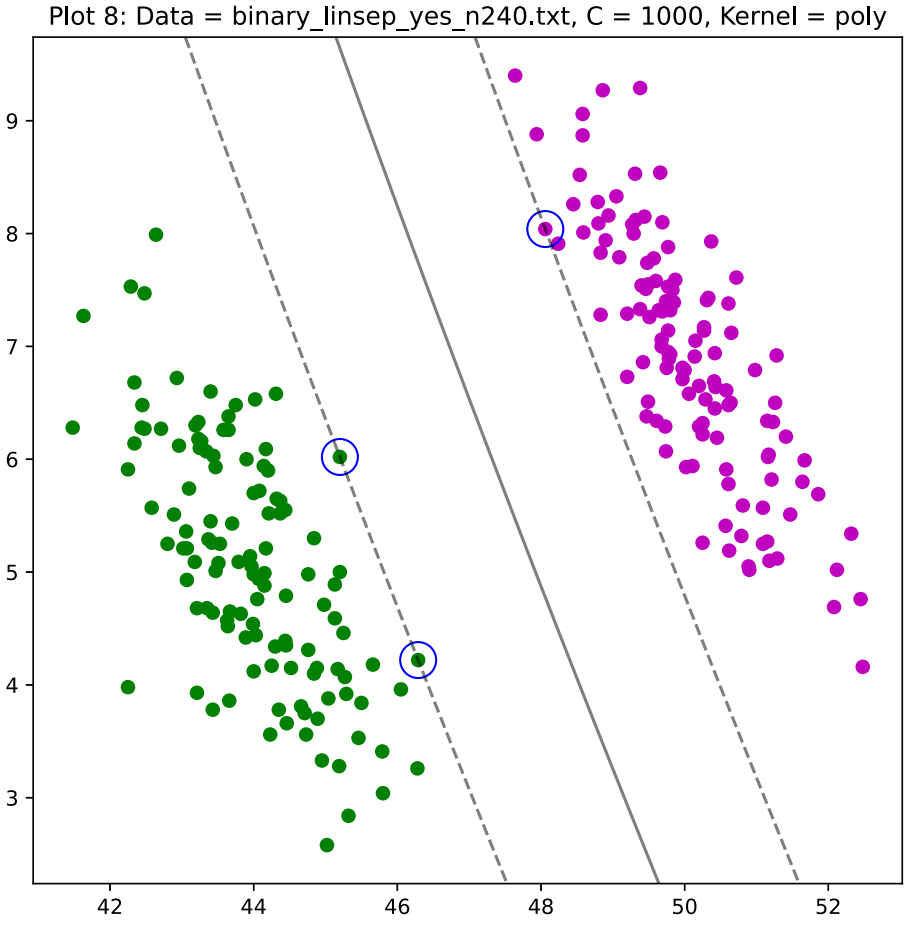

Plot 3: Data = binary_linsep_yes_n240.txt, C = 0.01, Kernel = rbf
9
8
7
6
5
4
3
42
44
46
48
50
52

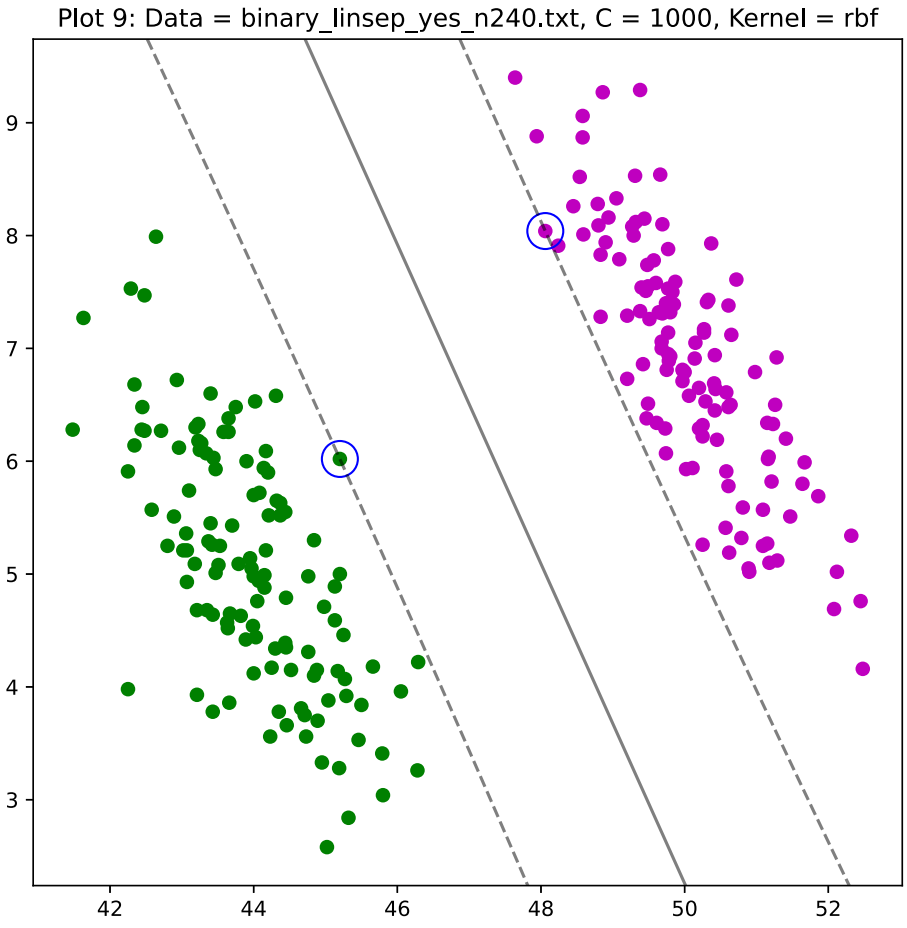

Plot 4: Data = binary_linsep_yes_n240.txt, C = 10, Kernel = linear
9
8
7
6
5
4
3
42
44
46
48
50
52

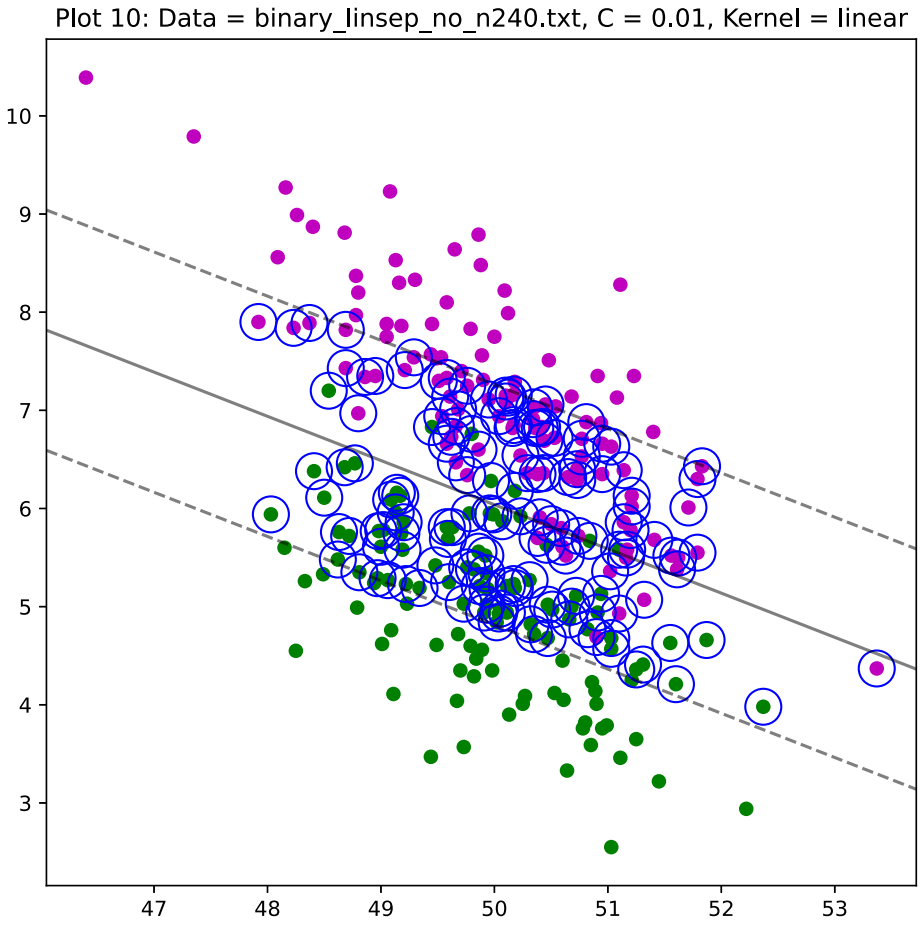

Plot 5: Data = binary_linsep_yes_n240.txt, C = 10, Kernel = poly
9
8
7
6
5
4
3
42
44
46
48
50
52

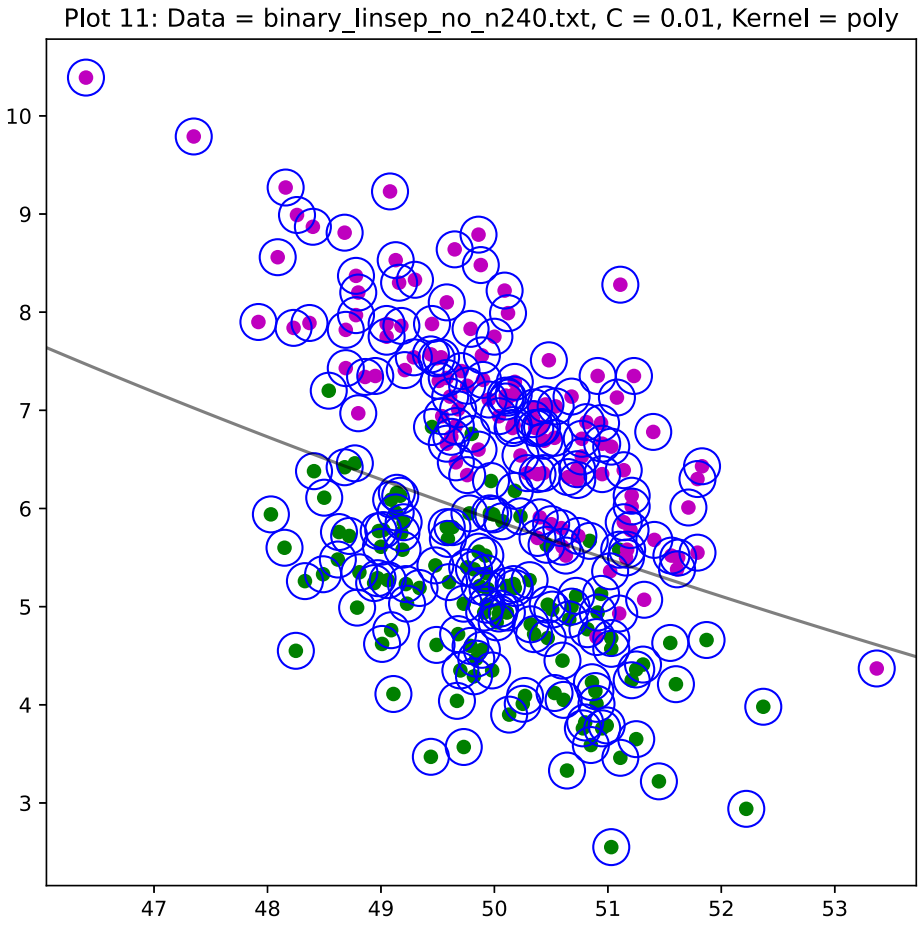

Plot 6: Data = binary_linsep_yes_n240.txt, C = 10, Kernel = rbf
9
8
7
6
5
4
3
42
44
46
48
50
52

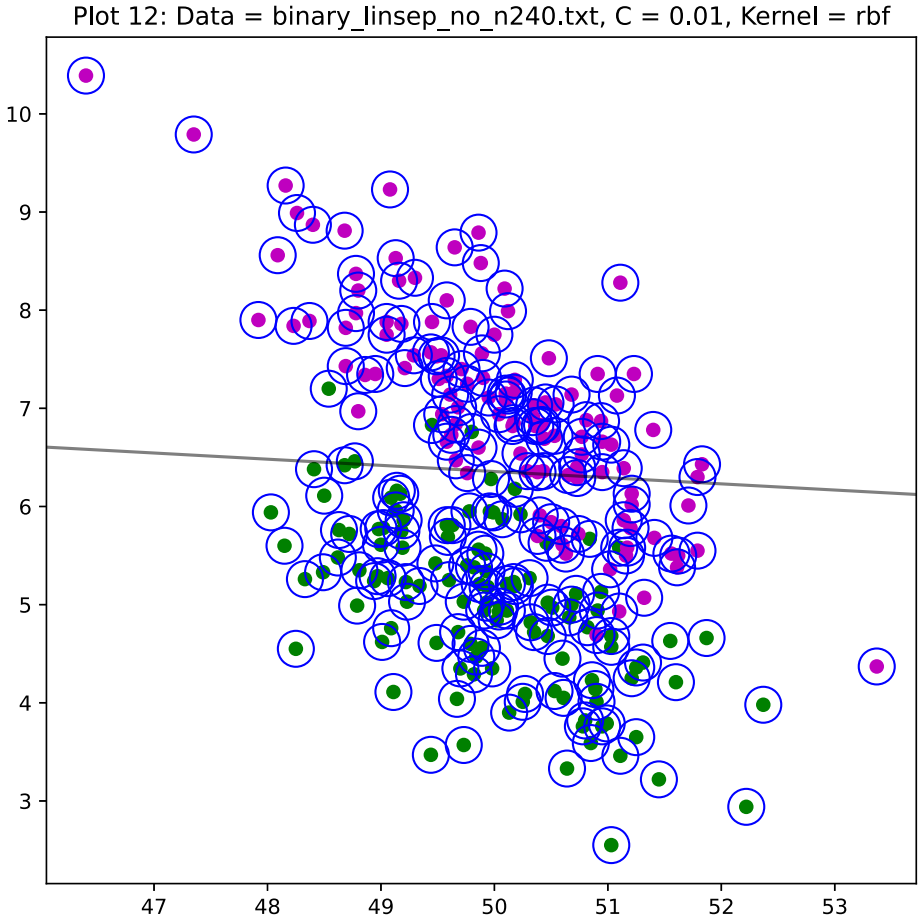

Plot 7: Data = binary_linsep_yes_n240.txt, C = 1000, Kernel = linear
9
8
7
6
5
4
3
42
44
46
48
50
52

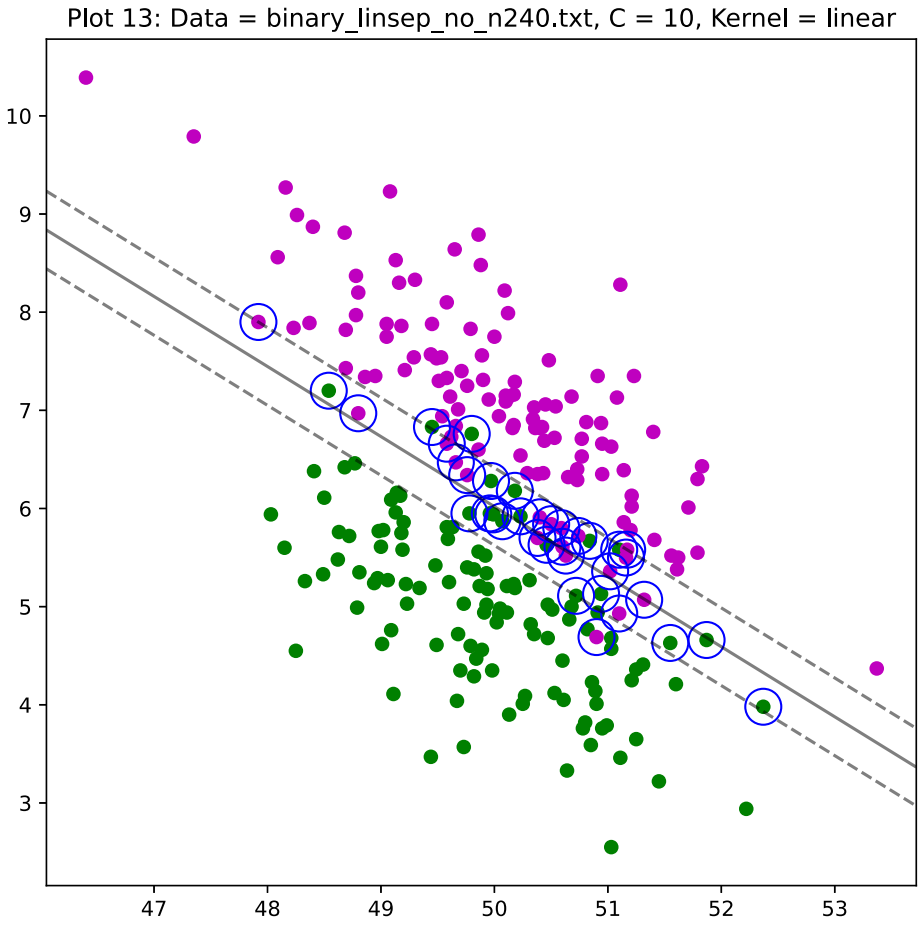

Plot 8: Data = binary_linsep_yes_n240.txt, C = 1000, Kernel = poly
9
8
7
6
5
4
3
42
44
46
48
50
52

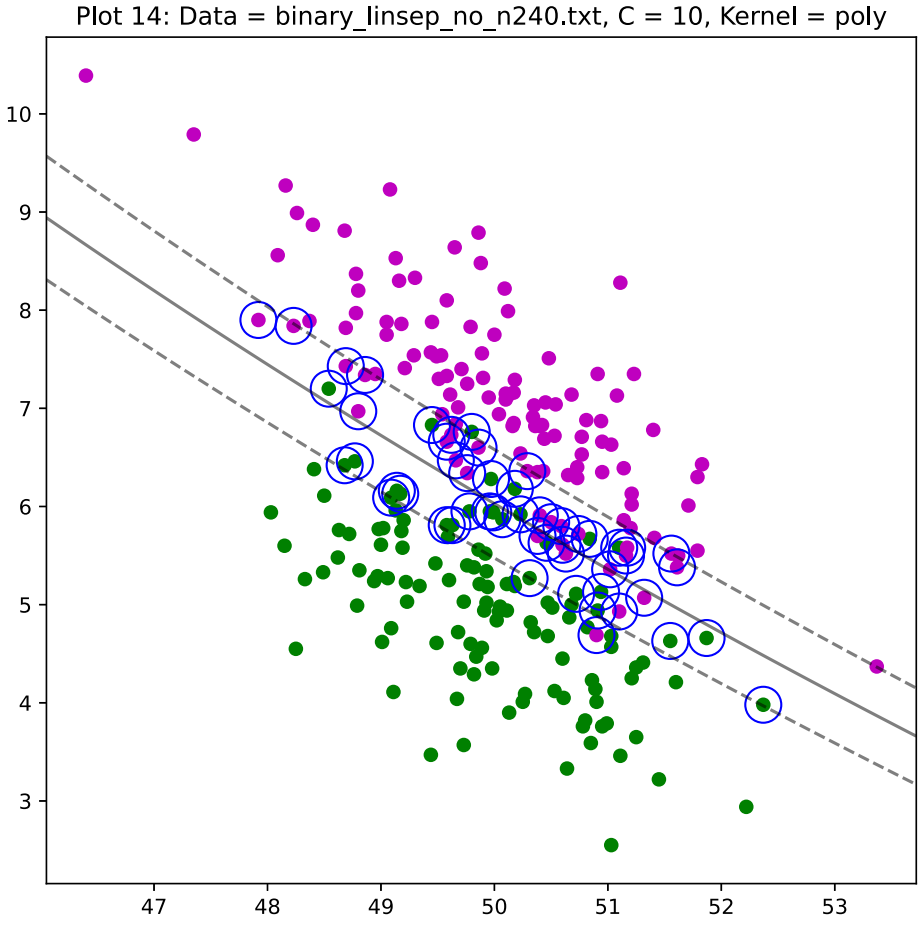

Plot 9: Data = binary_linsep_yes_n240.txt, C = 1000, Kernel = rbf
9
8
7
6
5
4
3
42
44
46
48
50
52

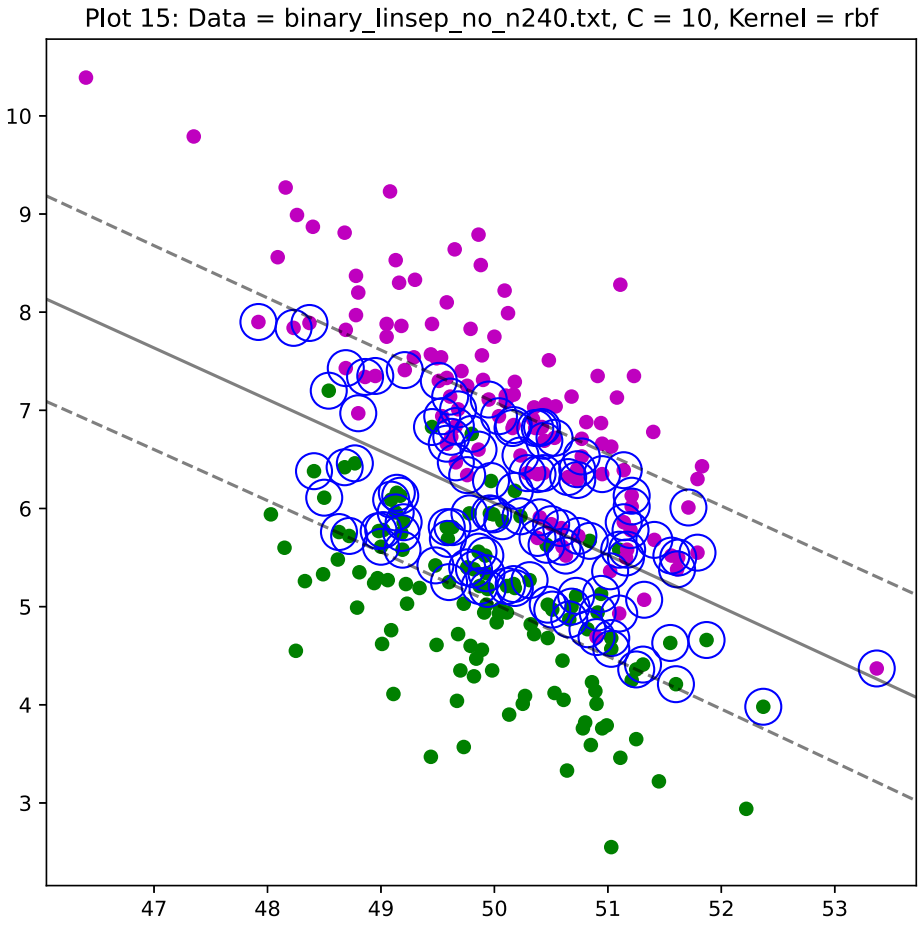

Plot 10: Data = binary_linsep_no_n240.txt, C = 0.01, Kernel = linear
10
9
8
7
6
5
4
3
47
48
49
50
51
52
53

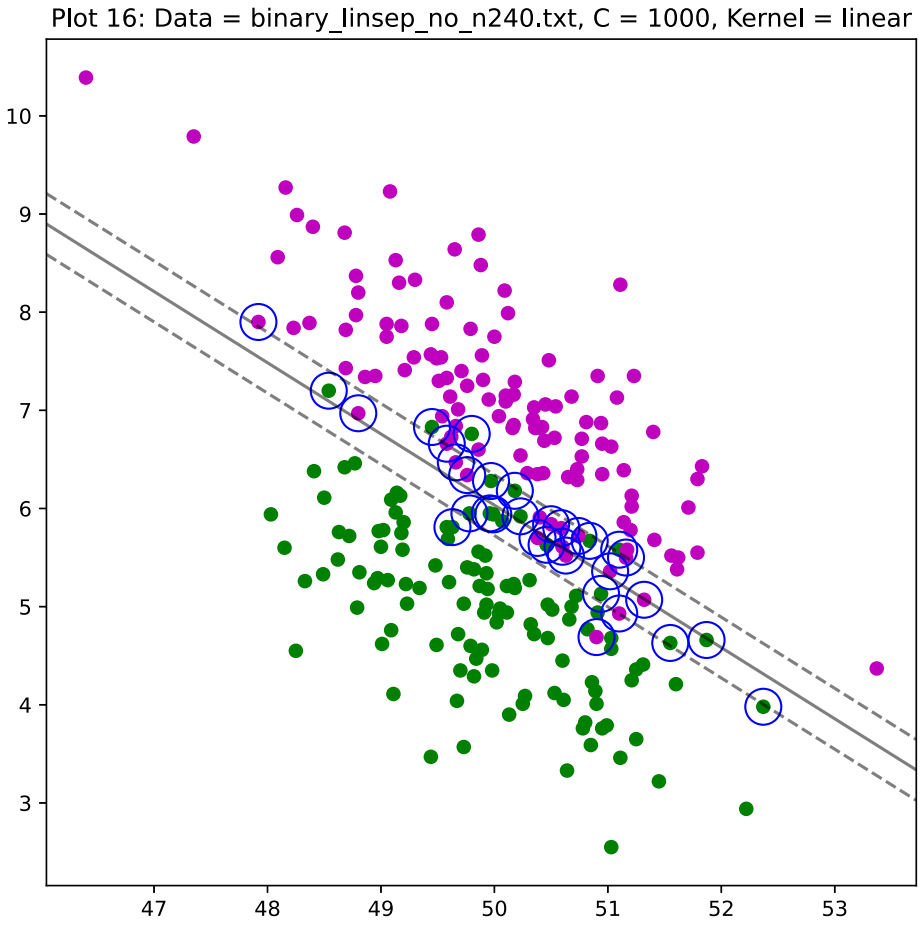

Plot 11: Data = binary_linsep_no_n240.txt, C = 0.01, Kernel = poly
10
9
8
7
6
5
4
3
47
48
49
50
51
52
53

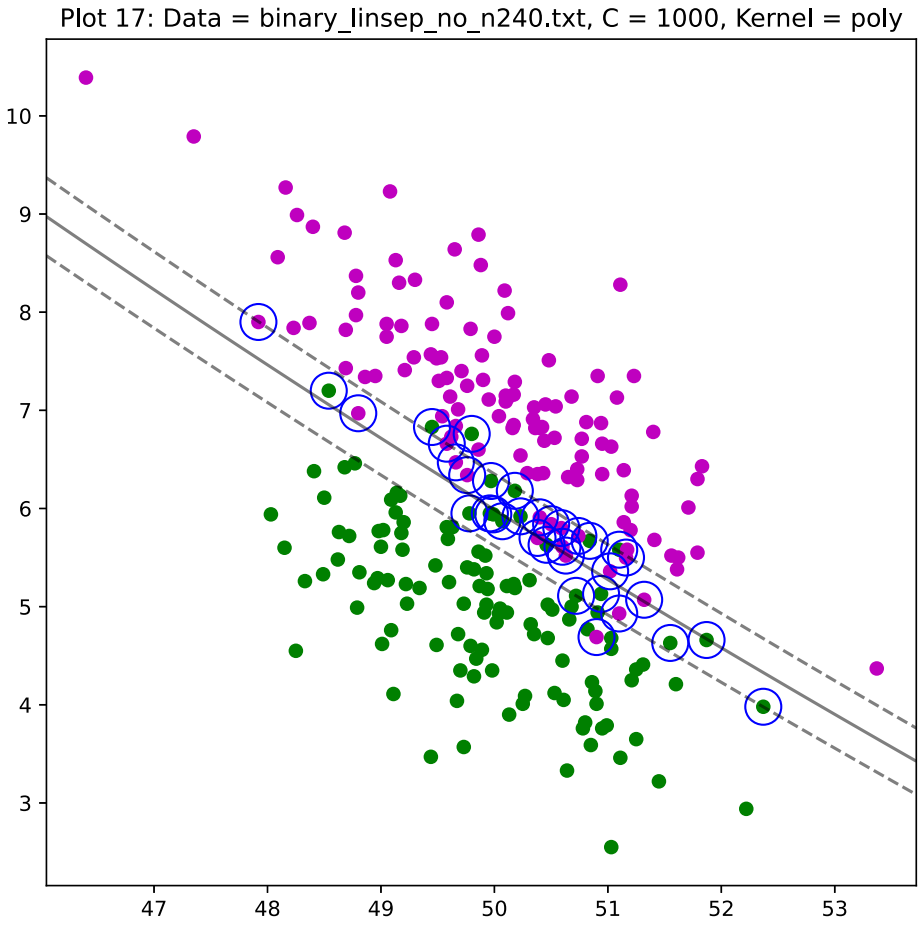

Plot 12: Data = binary_linsep_no_n240.txt, C = 0.01, Kernel = rbf
10
9
8
7
6
5
4
3
47
48
49
50
51
52
53

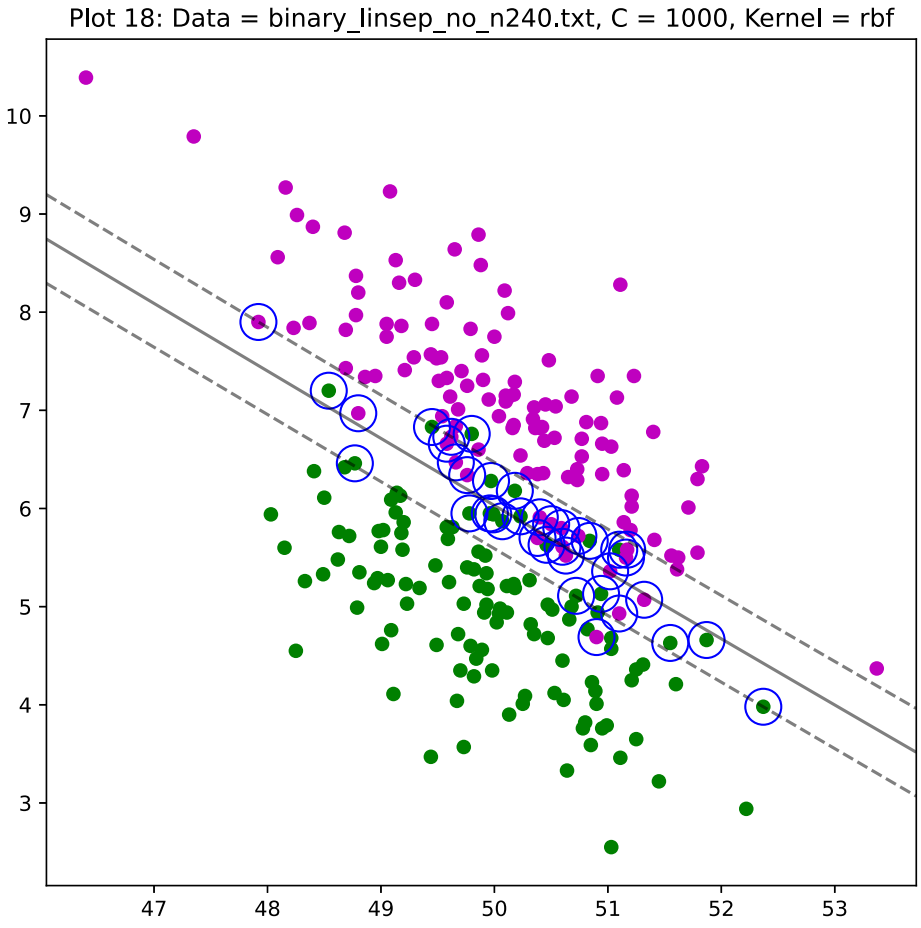

Plot 13: Data = binary_linsep_no_n240.txt, C = 10, Kernel = linear
10
9
8
7
6
5
4
3
47
48
49
50
51
52
53

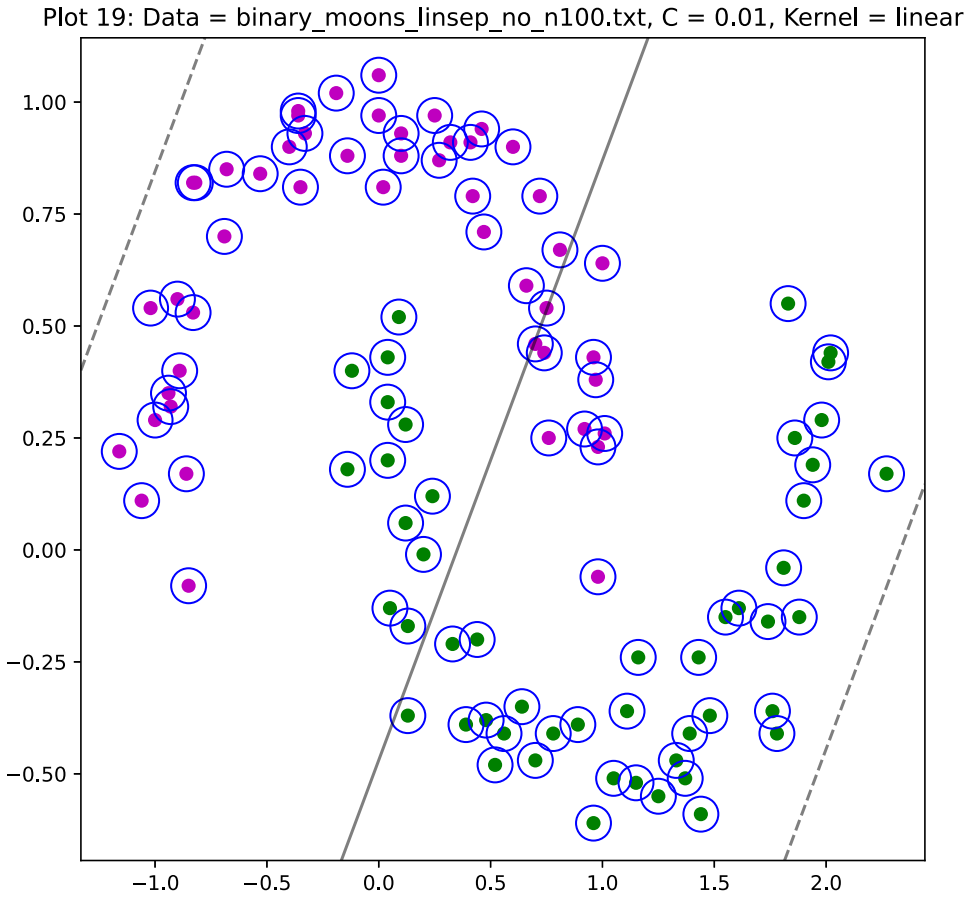

Plot 14: Data = binary_linsep_no_n240.txt, C = 10, Kernel = poly
10
9
8
7
6
5
4
3
47
48
49
50
51
52
53

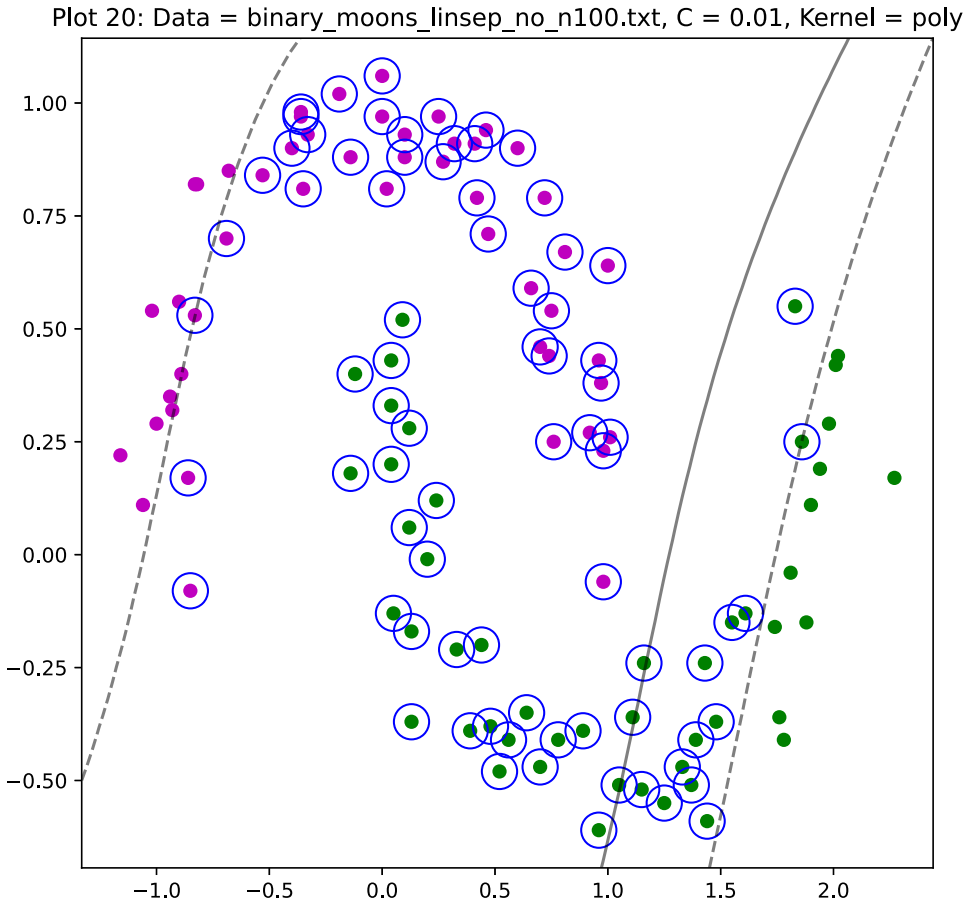

Plot 15: Data = binary_linsep_no_n240.txt, C = 10, Kernel = rbf
10
9
8
7
6
5
4
3
47
48
49
50
51
52
53

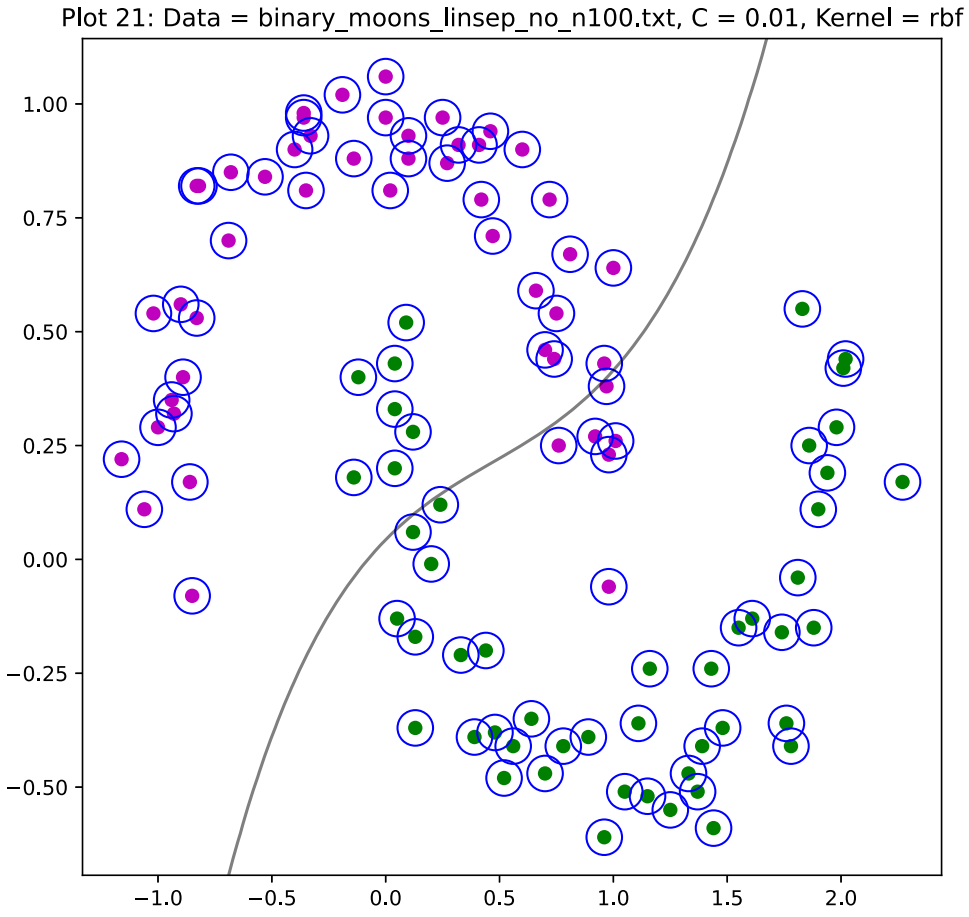

Plot 16: Data = binary_linsep_no_n240.txt, C = 1000, Kernel = linear
10
9
8
7
6
5
4
3
47
48
49
50
51
52
53

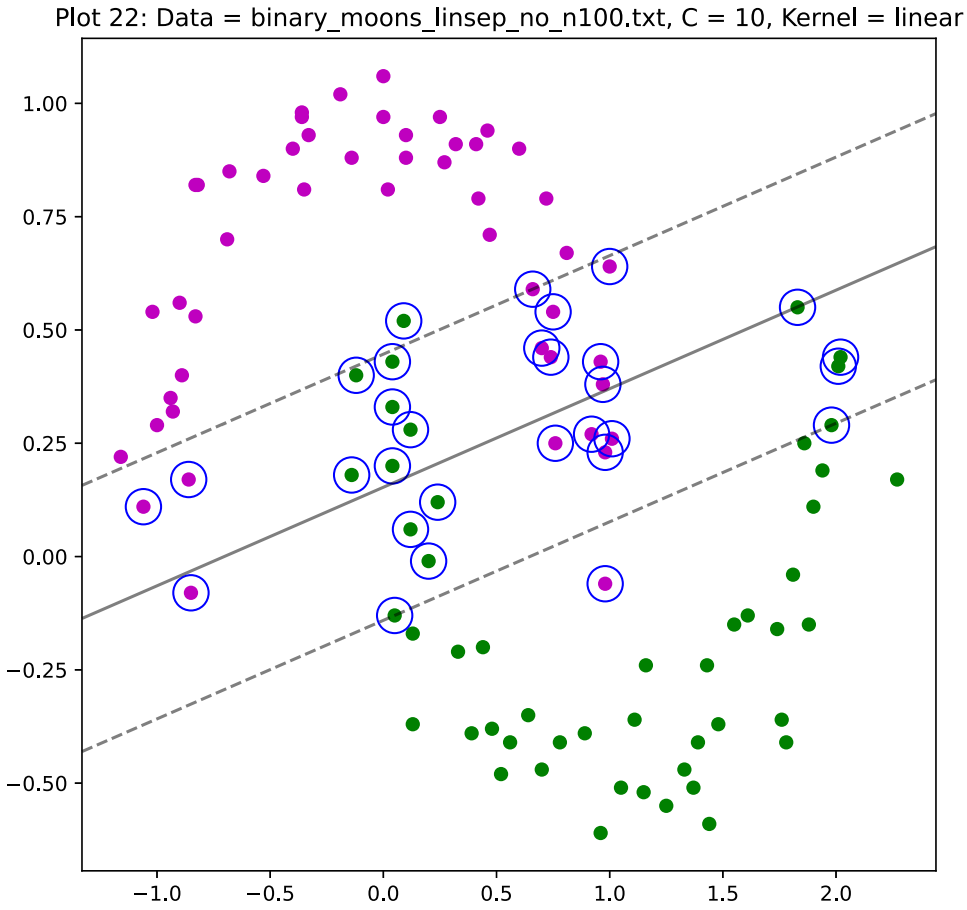

Plot 17: Data = binary_linsep_no_n240.txt, C = 1000, Kernel = poly
10
9
8
7
6
5
4
3
47
48
49
50
51
52
53

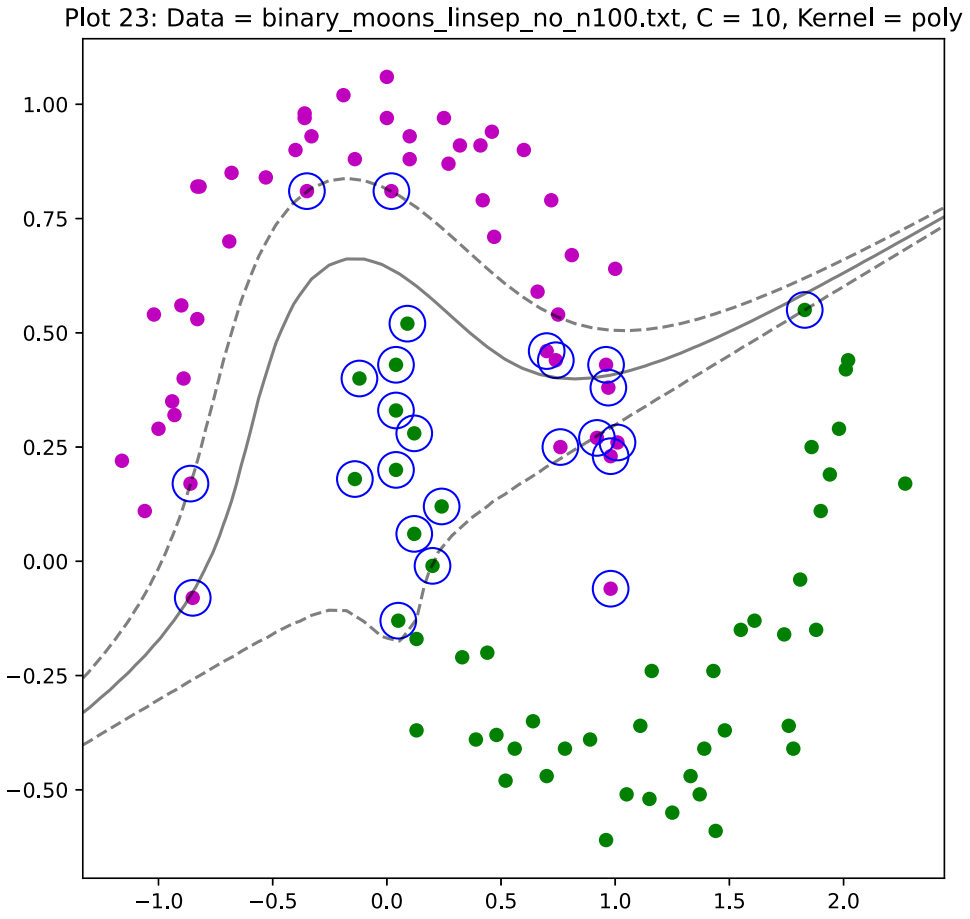

Plot 18: Data = binary_linsep_no_n240.txt, C = 1000, Kernel = rbf
10
9
8
7
6
5
4
3
47
48
49
50
51
52
53

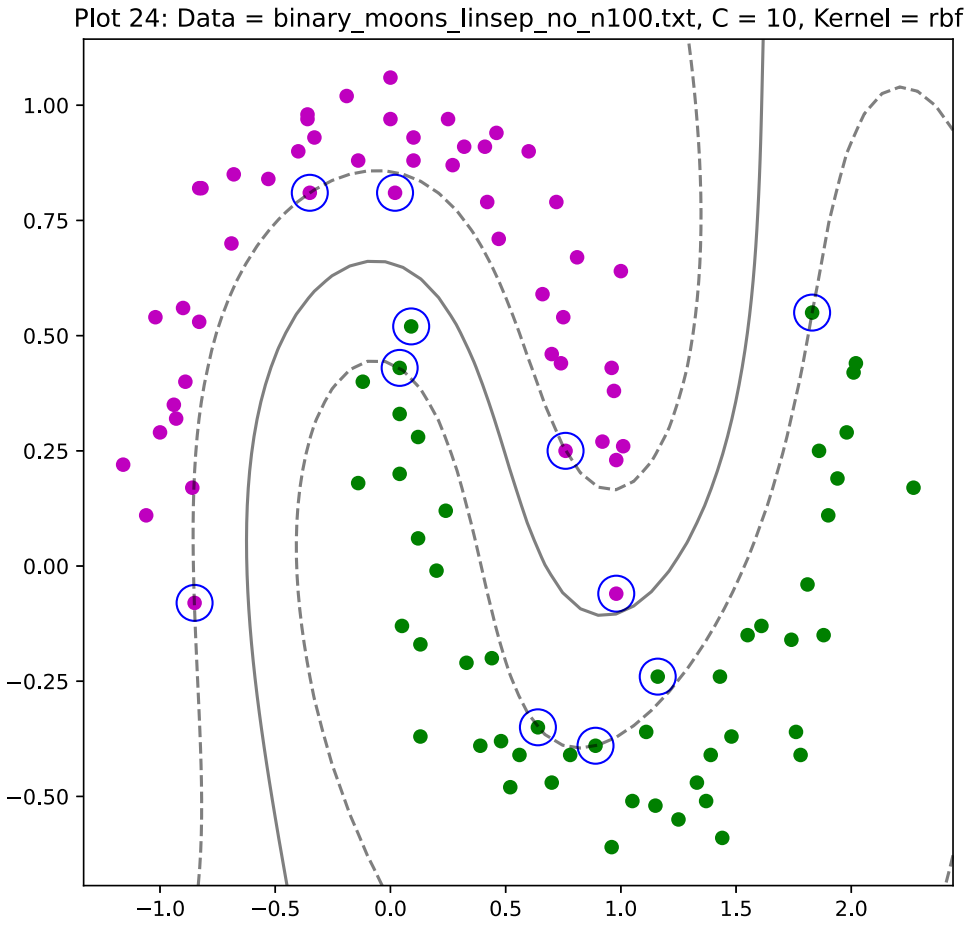

Plot 19: Data = binary_moons_linsep_no_n100.txt, C = 0.01, Kernel = linear
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

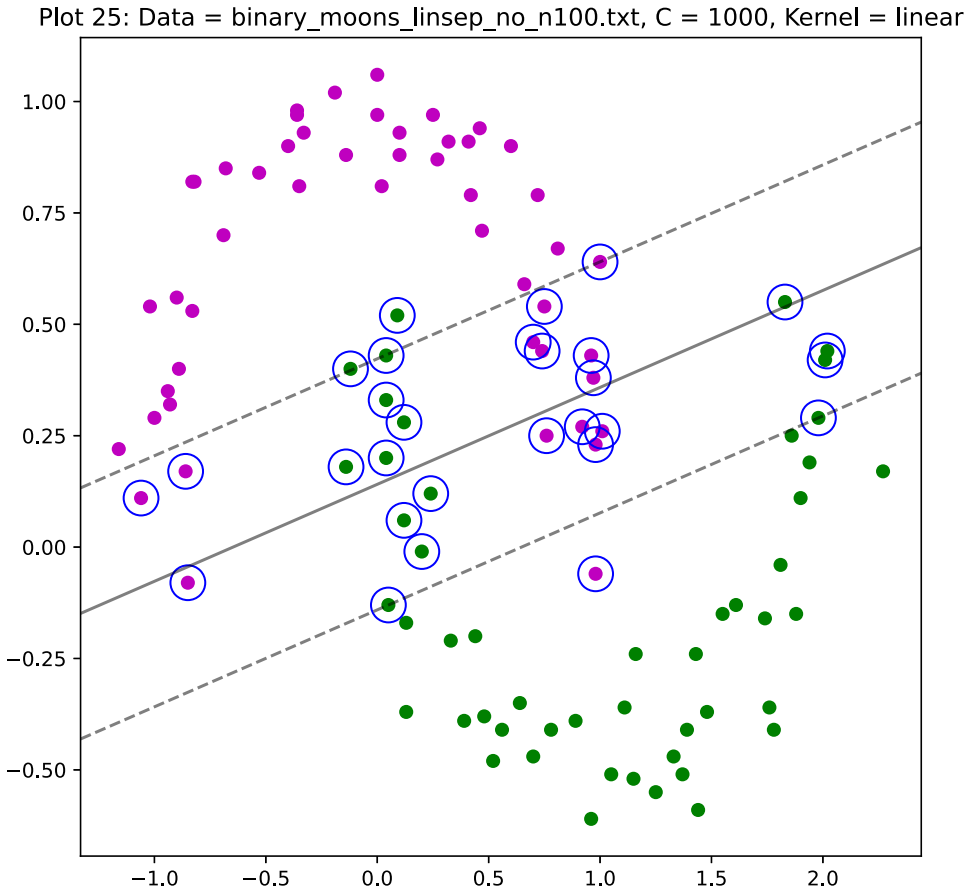

Plot 20: Data = binary_moons_linsep_no_n100.txt, C = 0.01, Kernel = poly
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

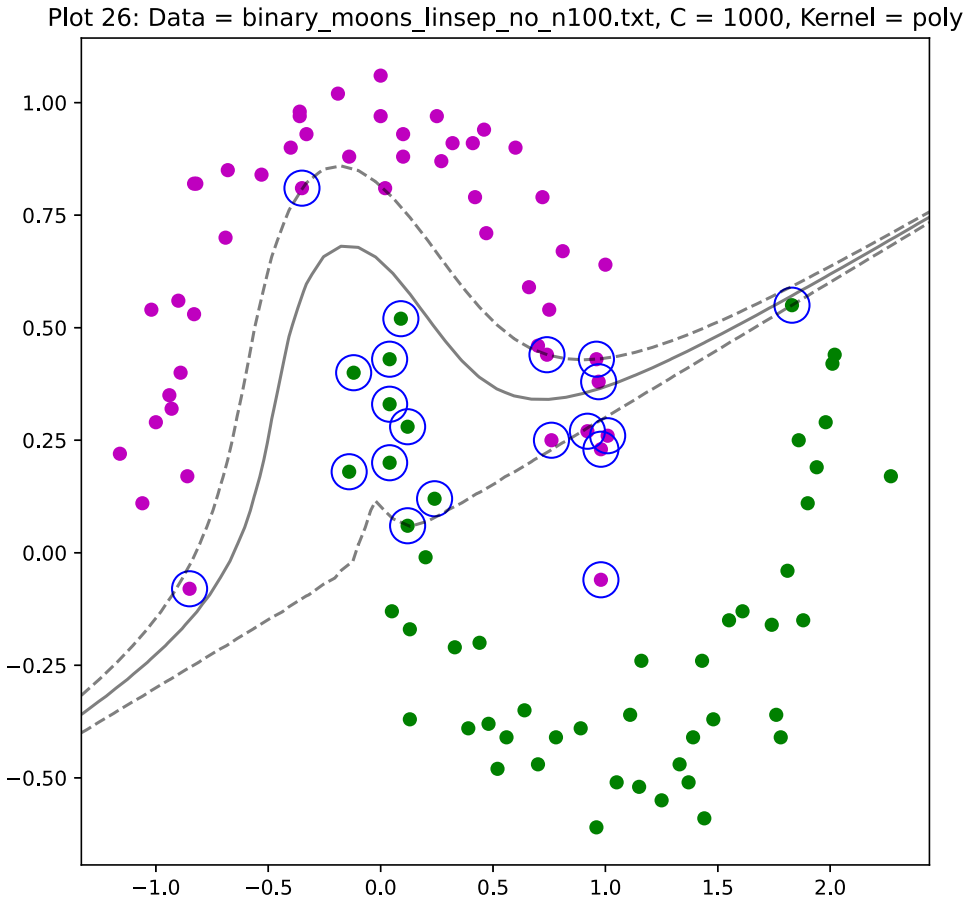

Plot 21: Data = binary_moons_linsep_no_n100.txt, C = 0.01, Kernel = rbf
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

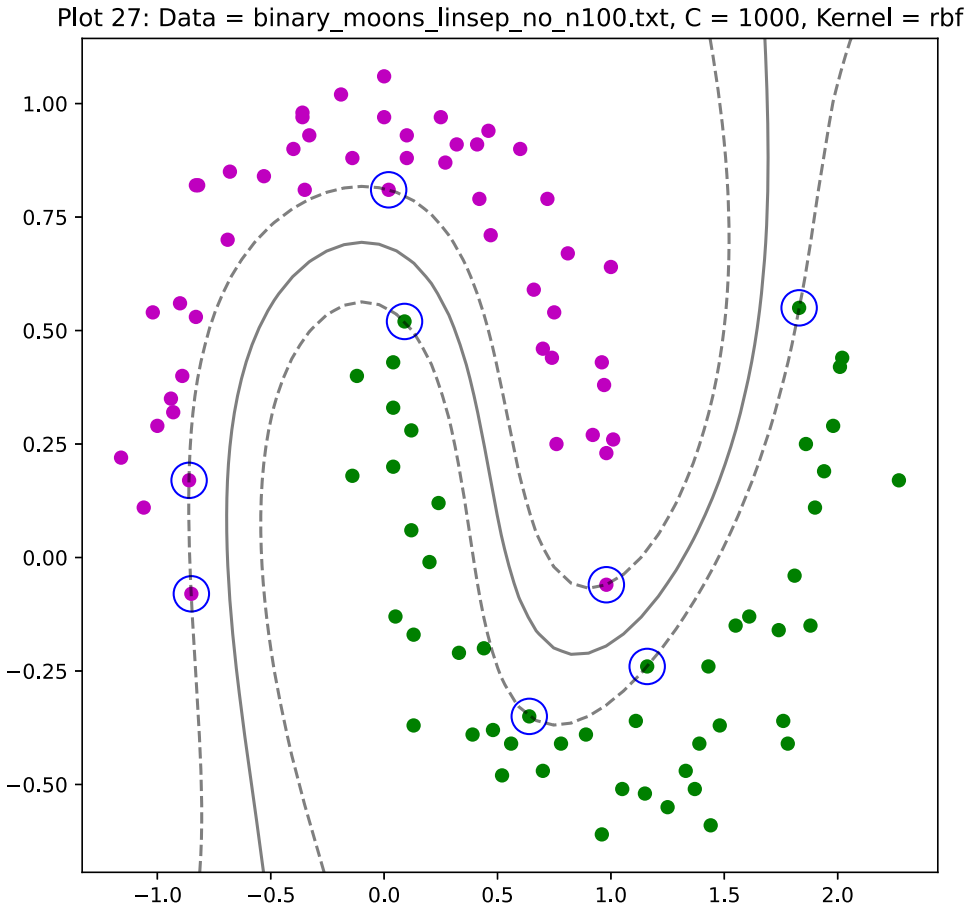

Plot 22: Data = binary_moons_linsep_no_n100.txt, C = 10, Kernel = linear
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

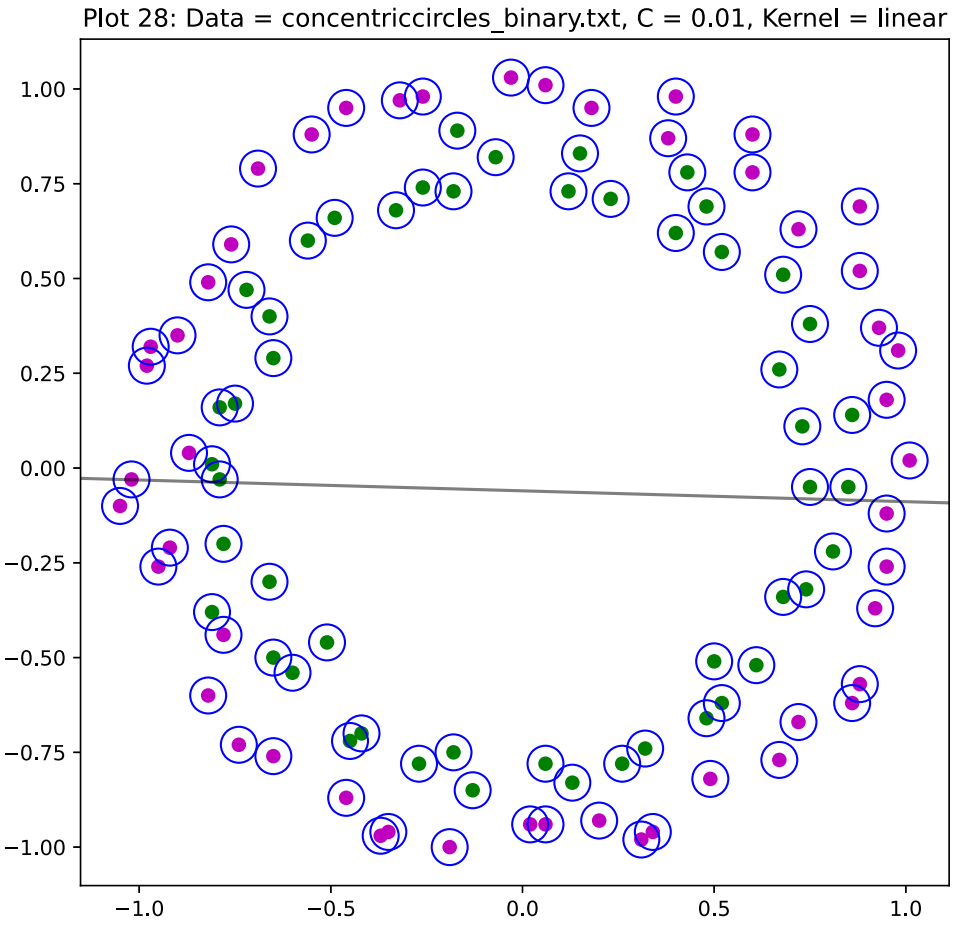

Plot 23: Data = binary_moons_linsep_no_n100.txt, C = 10, Kernel = poly
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

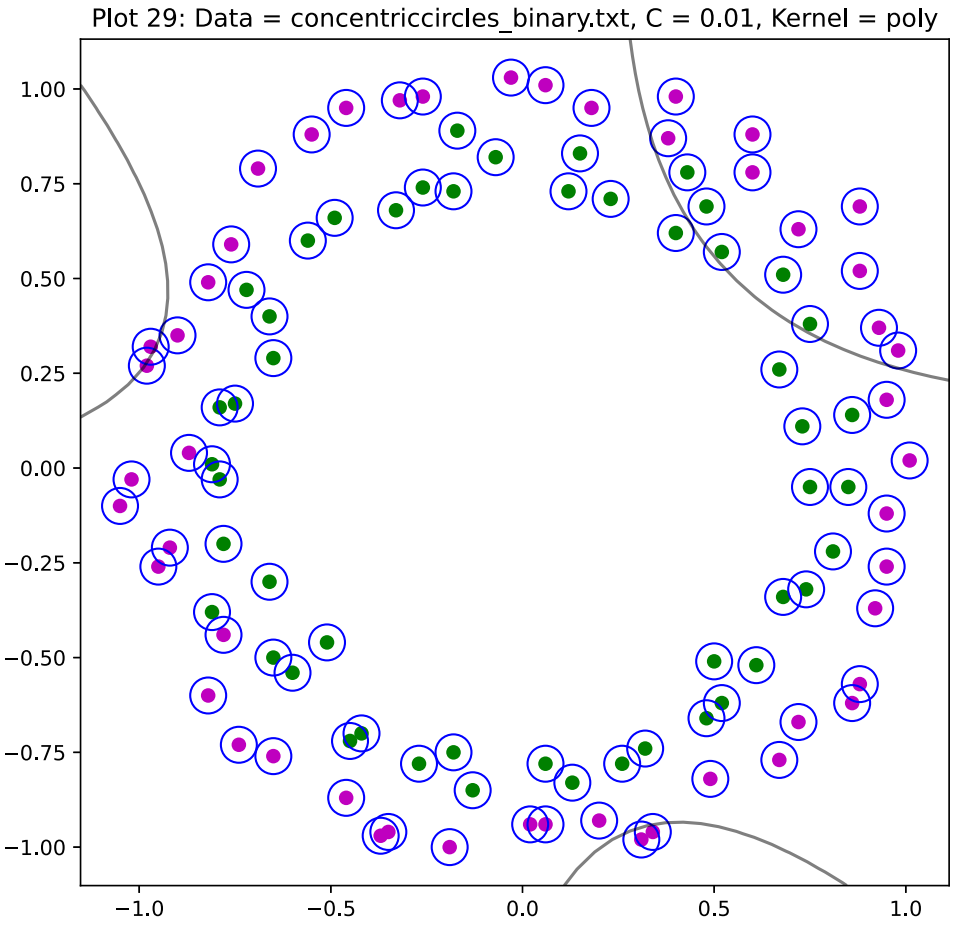

Plot 24: Data = binary_moons_linsep_no_n100.txt, C = 10, Kernel = rbf
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

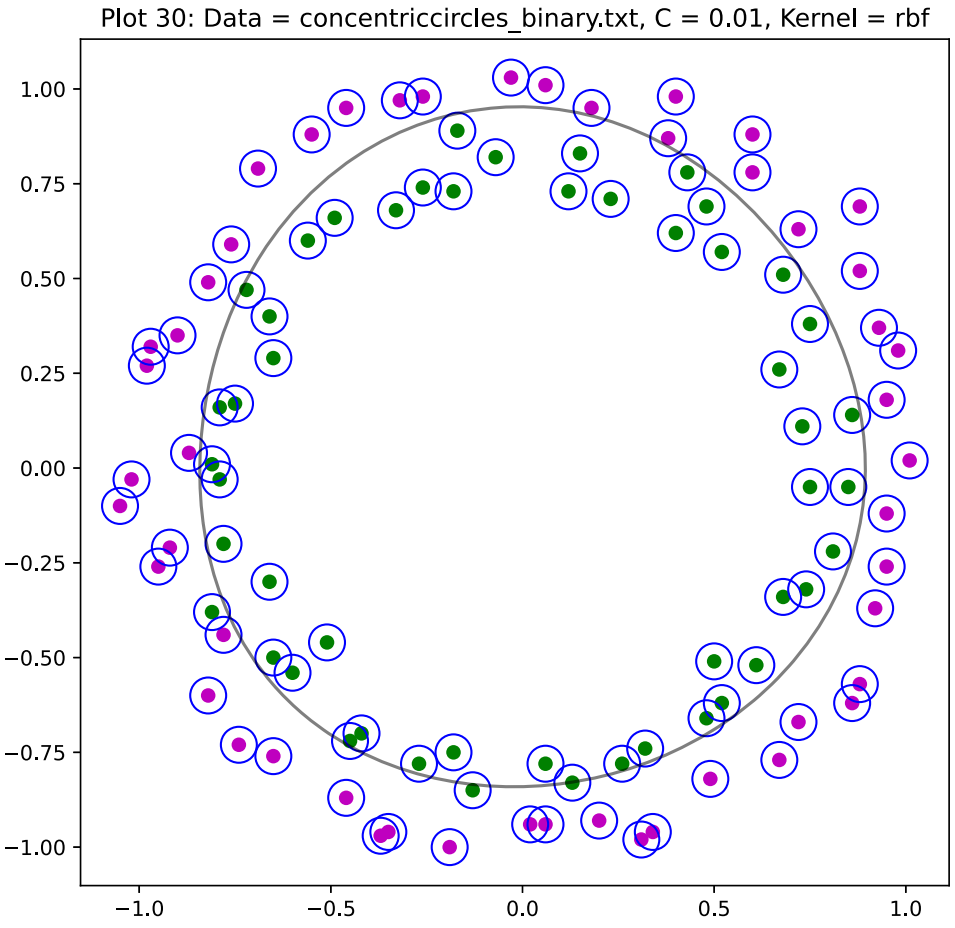

Plot 25: Data = binary_moons_linsep_no_n100.txt, C = 1000, Kernel = linear
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

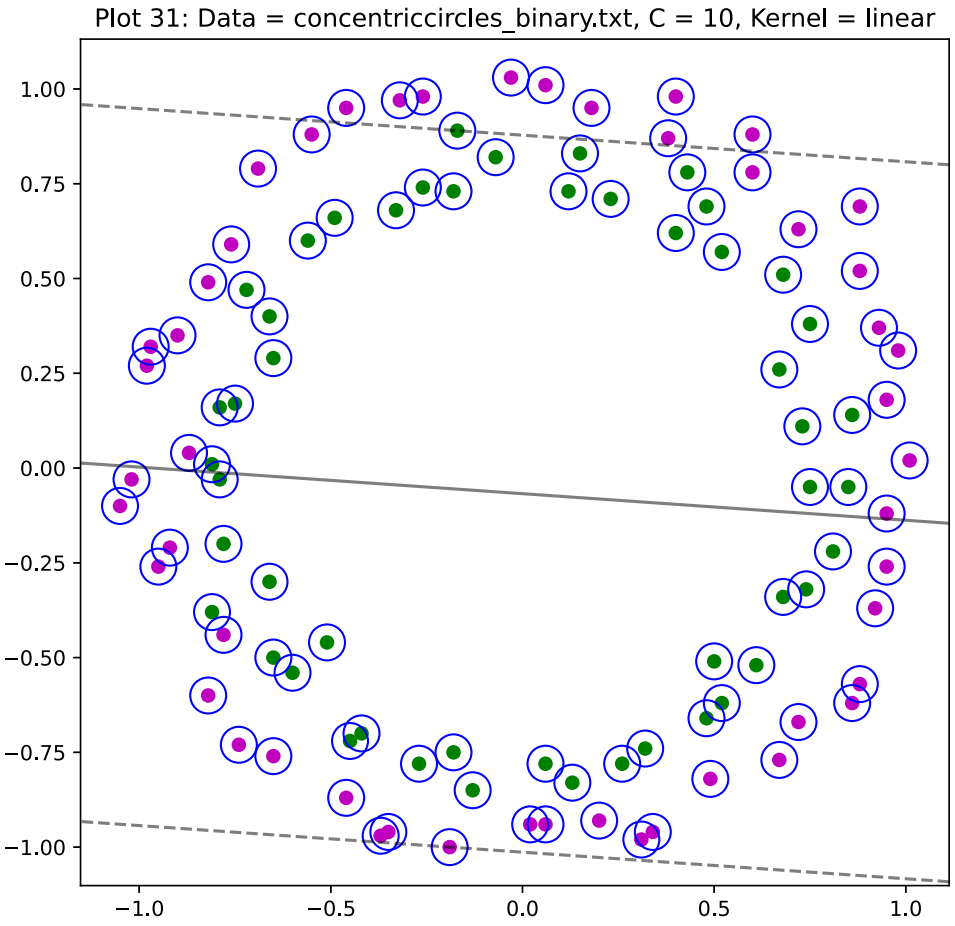

Plot 26: Data = binary_moons_linsep_no_n100.txt, C = 1000, Kernel = poly
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

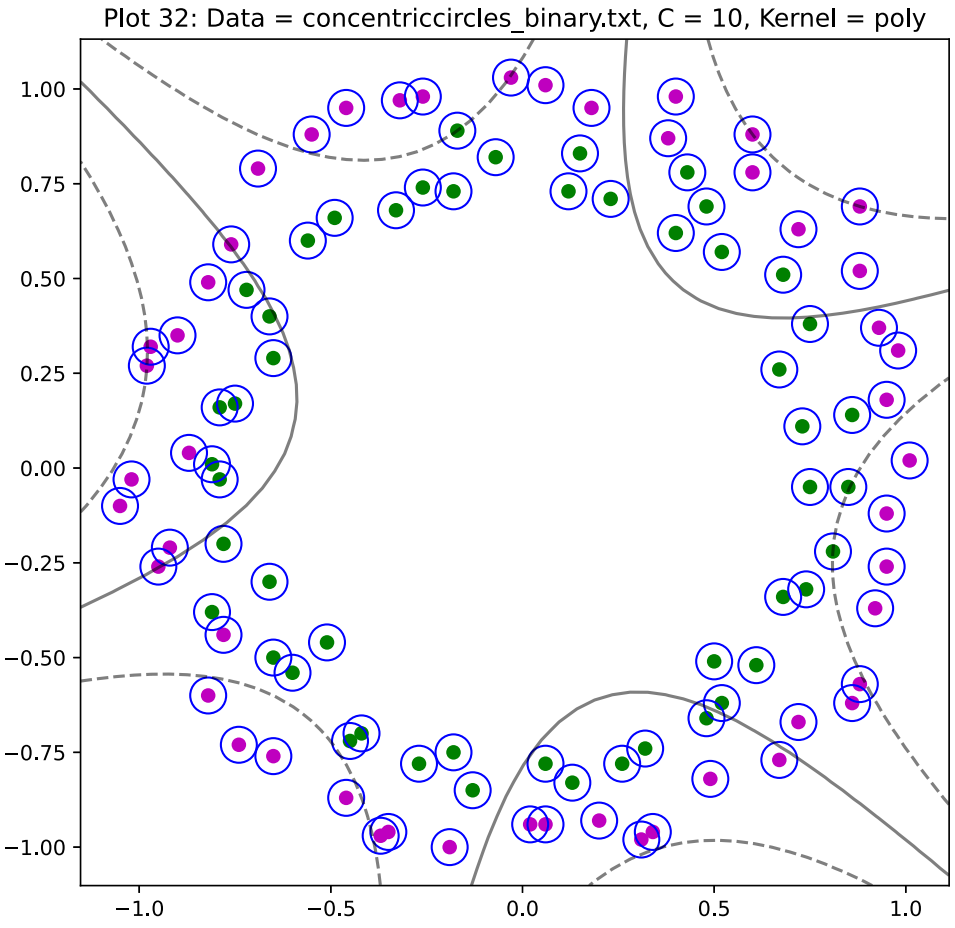

Plot 27: Data = binary_moons_linsep_no_n100.txt, C = 1000, Kernel = rbf
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

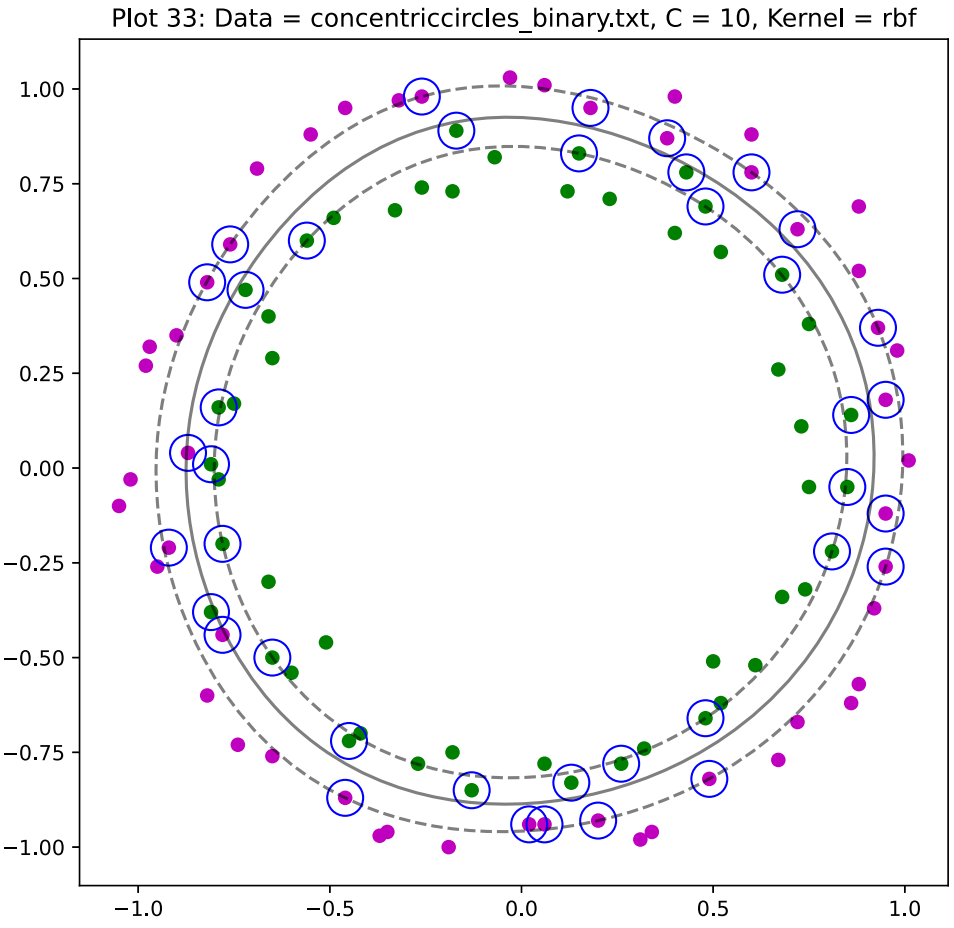

Plot 28: Data = concentriccircles_binary.txt, C = 0.01, Kernel = linear
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.0
−0.5
0.0
0.5
1.0

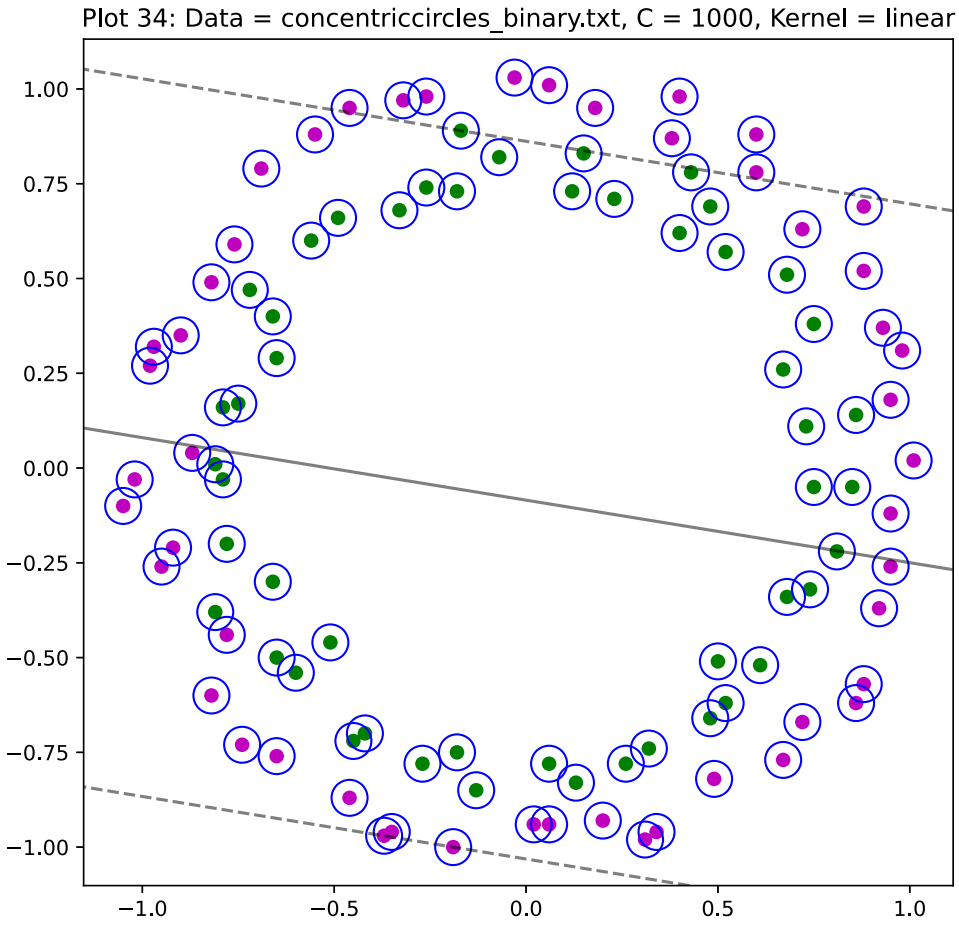

Plot 29: Data = concentriccircles_binary.txt, C = 0.01, Kernel = poly
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.0
−0.5
0.0
0.5
1.0

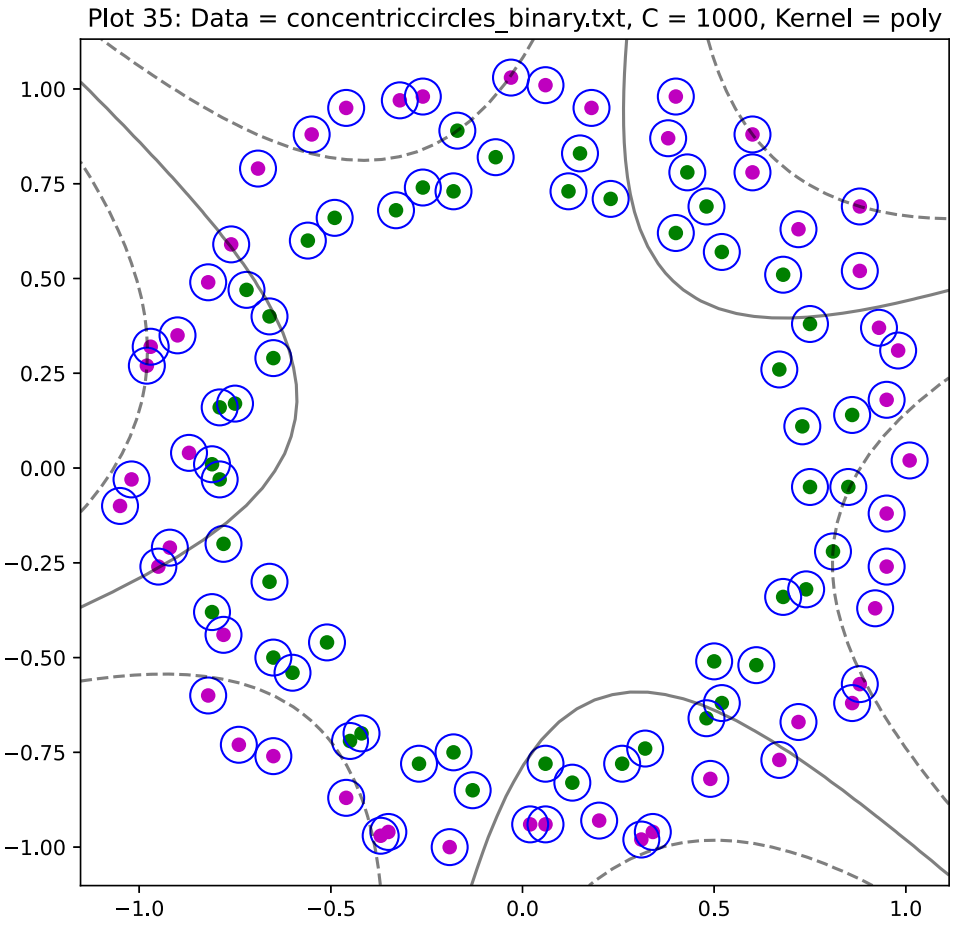

Plot 30: Data = concentriccircles_binary.txt, C = 0.01, Kernel = rbf
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.0
−0.5
0.0
0.5
1.0

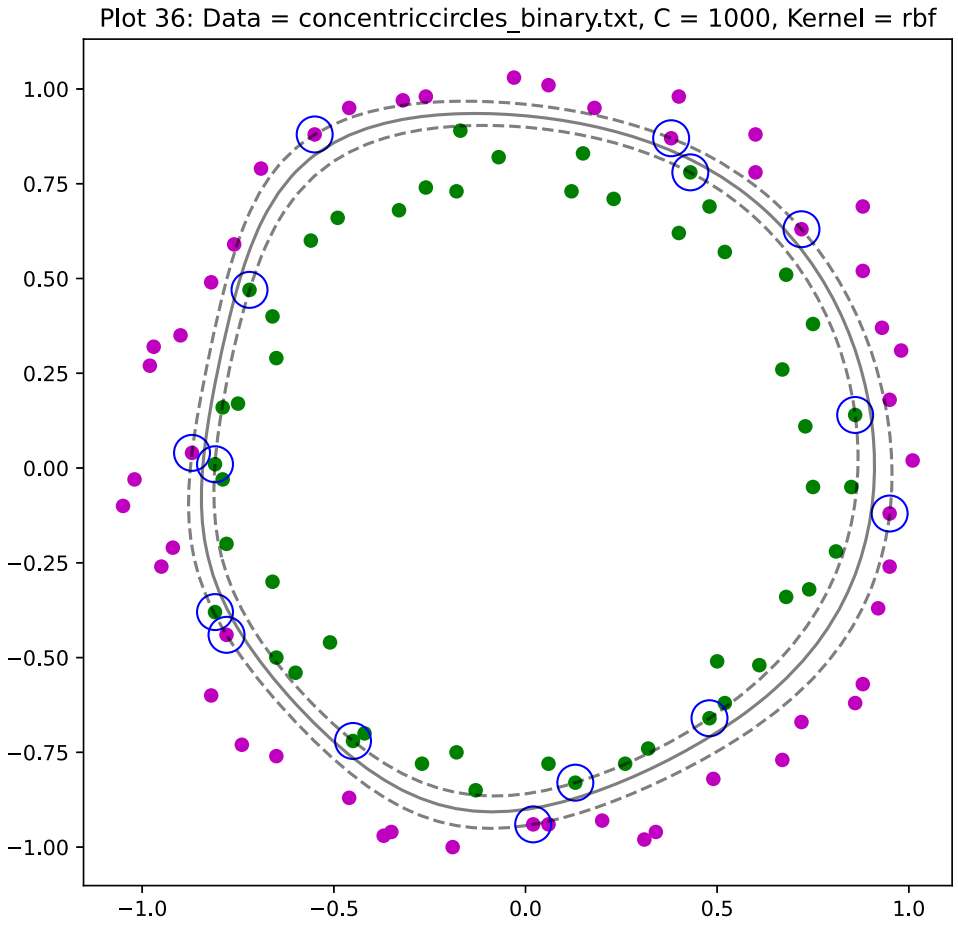

Plot 31: Data = concentriccircles_binary.txt, C = 10, Kernel = linear
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.0
−0.5
0.0
0.5
1.0


Plot 32: Data = concentriccircles_binary.txt, C = 10, Kernel = poly
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.0
−0.5
0.0
0.5
1.0


Plot 33: Data = concentriccircles_binary.txt, C = 10, Kernel = rbf
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.0
−0.5
0.0
0.5
1.0


Plot 34: Data = concentriccircles_binary.txt, C = 1000, Kernel = linear
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.0
−0.5
0.0
0.5
1.0


Plot 35: Data = concentriccircles_binary.txt, C = 1000, Kernel = poly
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.0
−0.5
0.0
0.5
1.0




```
Support vector values (Shape = (8, 2)) for case of interest:

[[-0.86  0.17]
 [ 0.02  0.81]
 [ 0.98 -0.06]
 [-0.85 -0.08]
 [ 0.64 -0.35]
 [ 0.09  0.52]
 [ 1.83  0.55]
 [ 1.16 -0.24]]
```

In [ ]:

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.svm import SVR
```

3.) Support Vector Regression

In [2]:
```
""" Helper function to perform data standardization """
def data_std(X, mean, std):
    return (X - mean)/std
```

Part a.)

In [3]:
```
""" Load and parse data """
data = np.genfromtxt("train.txt", delimiter="\t")
train_X = data[:, 0:2]
train_y = data[:, 2]
data = np.genfromtxt("test.txt", delimiter="\t")
test_X = data[:, 0:2]
test_y = data[:, 2]

""" Establish standardized data """
# Mean of training features: (# of features, )
mean = np.mean(train_X, axis = 0)
# Standard deviation of training features: (# of features, )
std = np.std(train_X, axis = 0)

train_X_st = data_std(train_X, mean, std)
test_X_st = data_std(test_X, mean, std)
```

In [4]:
```
""" Sklearn SVR model """
# Default: C = 1, Kernel = rbf
model = SVR(epsilon=0.0001).fit(train_X_st, train_y)
pred = model.predict(test_X_st)

""" Compute RMSE """
RMSE = np.sqrt(np.mean(np.power(test_y - pred, 2)))
print("RMSE = {}".format(RMSE))
```

```
RMSE = 24.033860070431174
```

Part b.)

Same as part a.), but loop through permutations of C and ϵ

In [5]:
```
Cs = [0.01, 10, 100]
eps = [0.0001, 0.01, 1]
for i in range(len(Cs)):
    for j in range(len(eps)):
        model = SVR(C=Cs[i], epsilon=eps[j]).fit(train_X_st, train_y)
        pred = model.predict(test_X_st)
```

```
        RMSE = np.sqrt(np.mean(np.power(test_y - pred, 2)))
        print("C = {}, epsilon = {} -> RMSE = {}".format(Cs[i], eps[j], RMSE))
```

```
C = 0.01, epsilon = 0.0001 -> RMSE = 49.86230821904665
C = 0.01, epsilon = 0.01 -> RMSE = 49.86230821904665
C = 0.01, epsilon = 1 -> RMSE = 49.719198470155014
C = 10, epsilon = 0.0001 -> RMSE = 7.206613901052269
C = 10, epsilon = 0.01 -> RMSE = 7.2030835074726145
C = 10, epsilon = 1 -> RMSE = 7.195908569734601
C = 100, epsilon = 0.0001 -> RMSE = 4.356151814584229
C = 100, epsilon = 0.01 -> RMSE = 4.354976661414509
C = 100, epsilon = 1 -> RMSE = 4.189717810970863
```

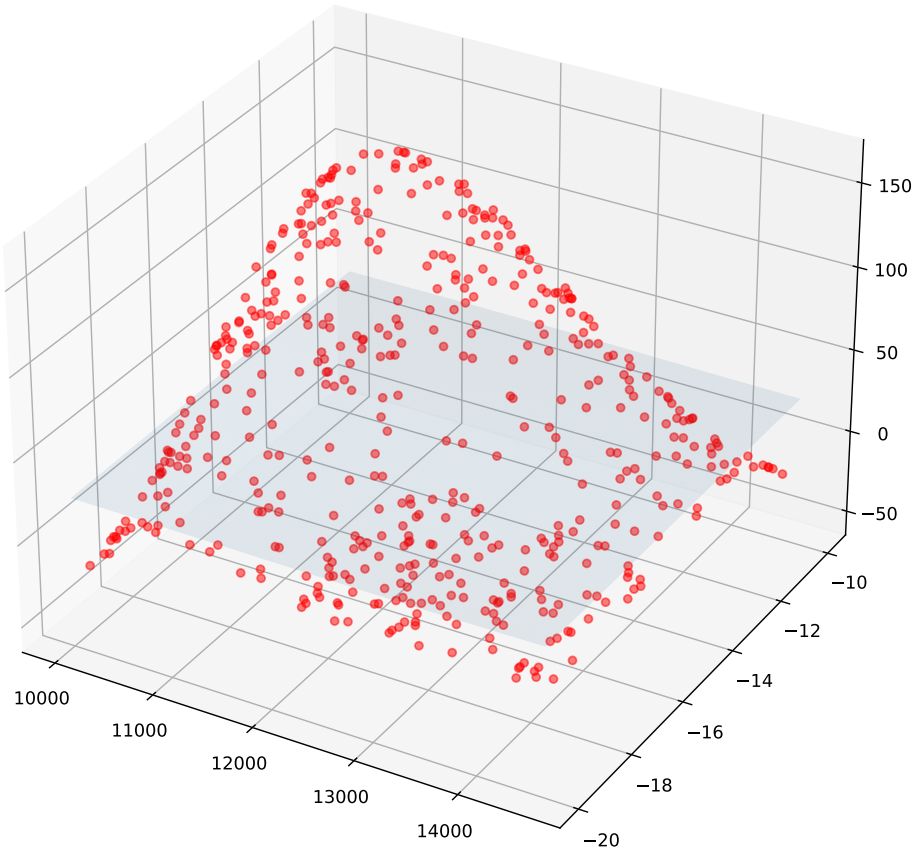
The higher penalty strengths C are much more effective in reducing the classification error, however, only to a certain value before the model is overfitted. While keeping C constant, it seems that a larger epsilon value yields less RMSE.

Part c.)

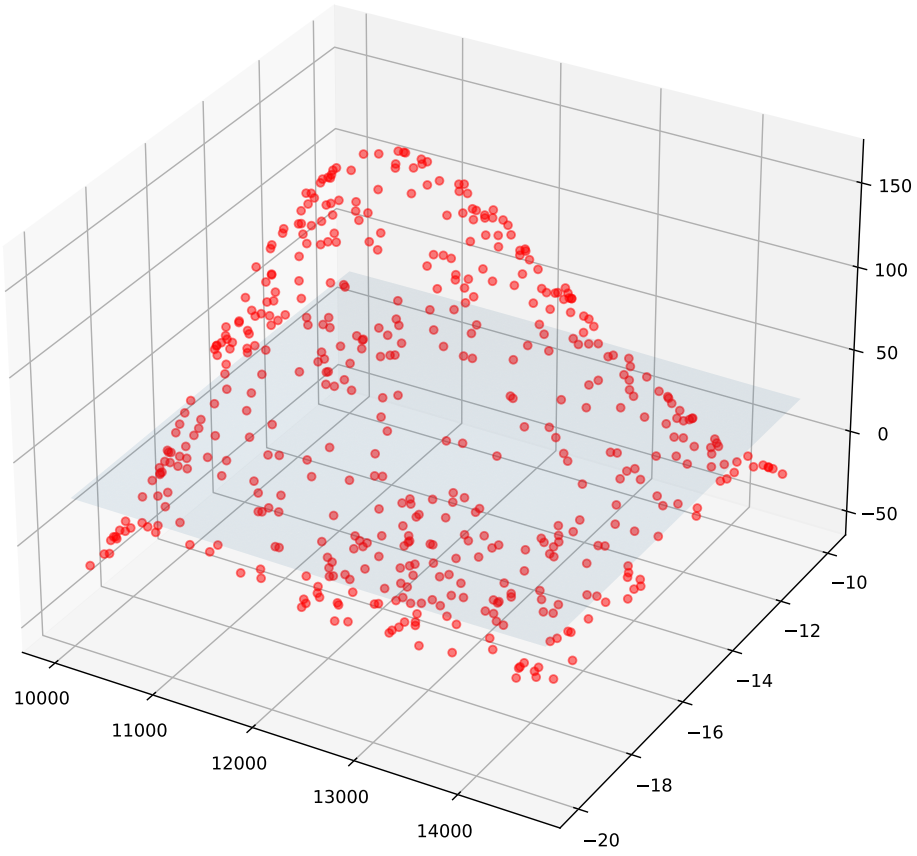
In [6]:

```
""" Create feature mesh and standardize the feature mesh """
xx, yy = np.meshgrid(np.linspace(test_X[:, 0].min(), test_X[:, 0].max(), 100), np.linsp
feat_mesh = data_std(np.c_[xx.ravel(), yy.ravel()], mean, std)

""" Same as part b.), but loop through permutations of C and epsilon """
Cs = [0.01, 10, 100]
eps = [0.0001, 0.01, 1]
count = 0
for i in range(len(Cs)):
    for j in range(len(eps)):
        model = SVR(C=Cs[i], epsilon=eps[j]).fit(train_X_st, train_y)
        pred = model.predict(feat_mesh)

        """ Plot each regression model as a surface """
        fig = plt.figure(count, figsize = (7.5, 7.5))
        ax = Axes3D(fig)
        ax.plot_trisurf(xx.flatten(), yy.flatten(), pred, linewidth=0, antialiased=Fals
        ax.scatter3D(test_X[:, 0], test_X[:, 1], test_y, c='r', alpha=0.5)
        plt.title("C = {}, epsilon = {}".format(Cs[i], eps[j]))
        plt.show()
        count += 1
```

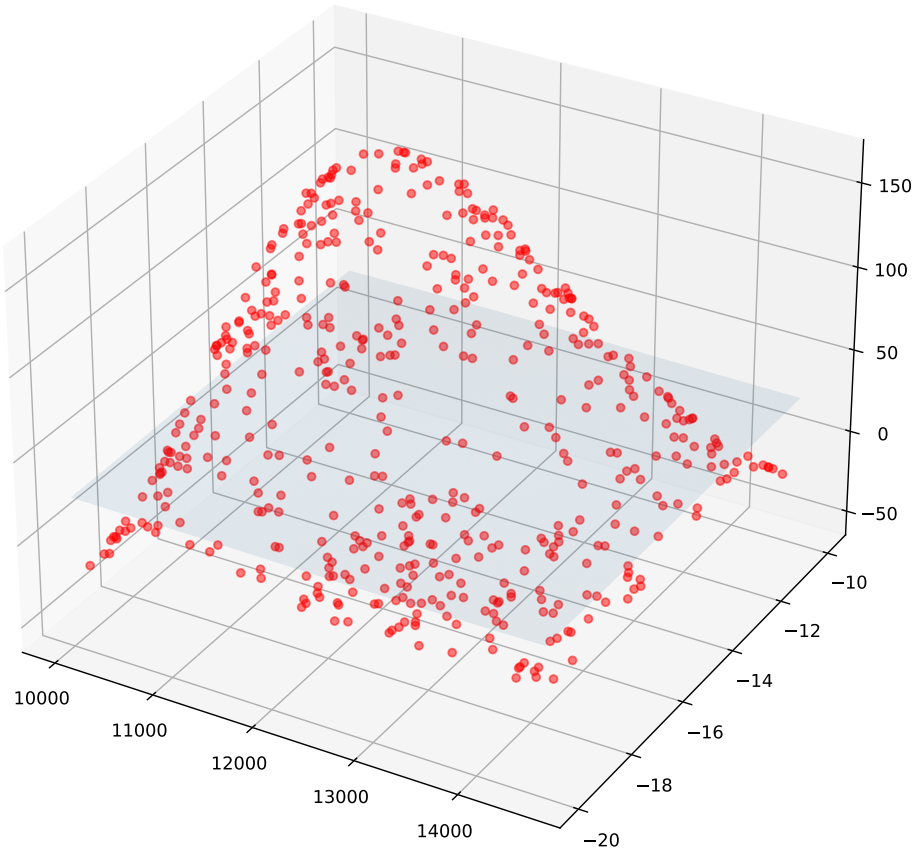
C = 0.01, epsilon = 0.0001

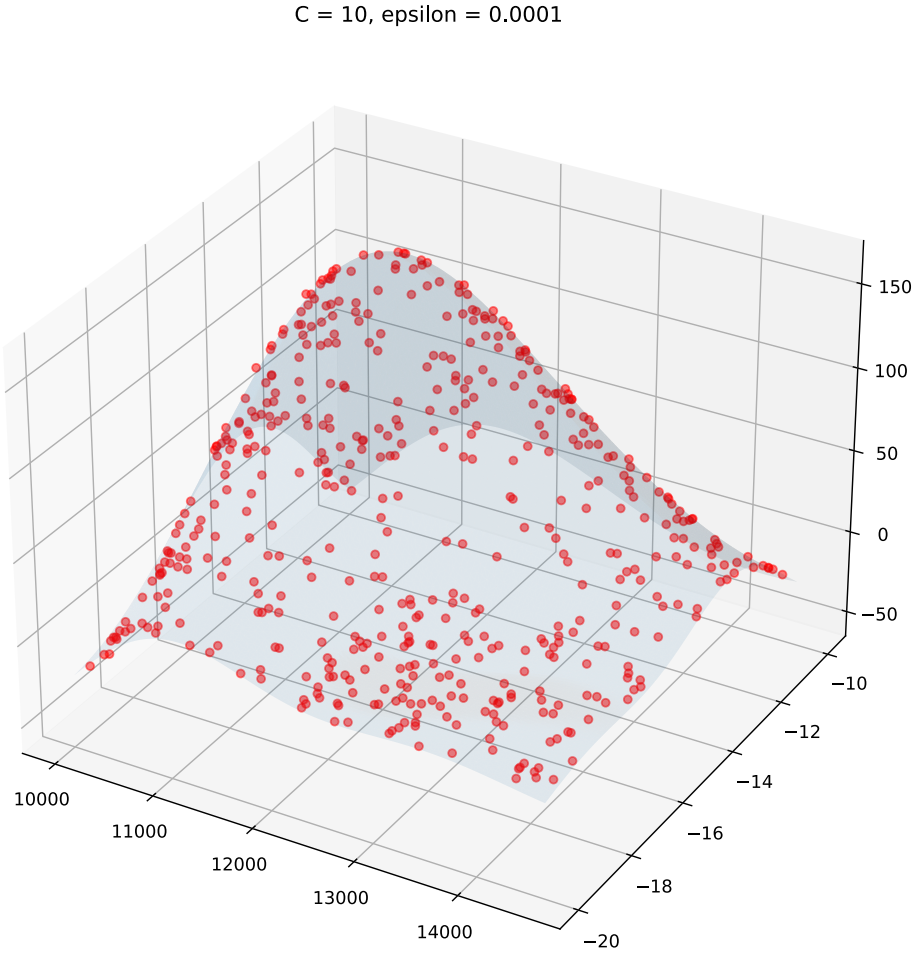


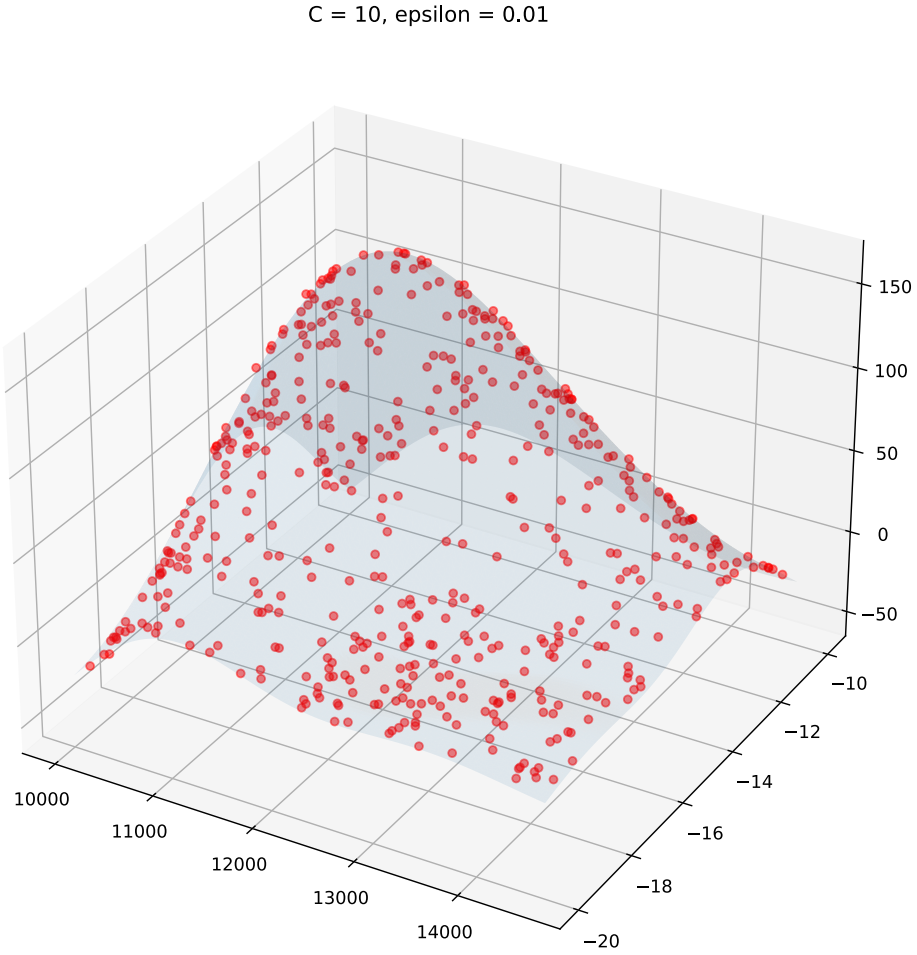
C = 0.01, epsilon = 0.01



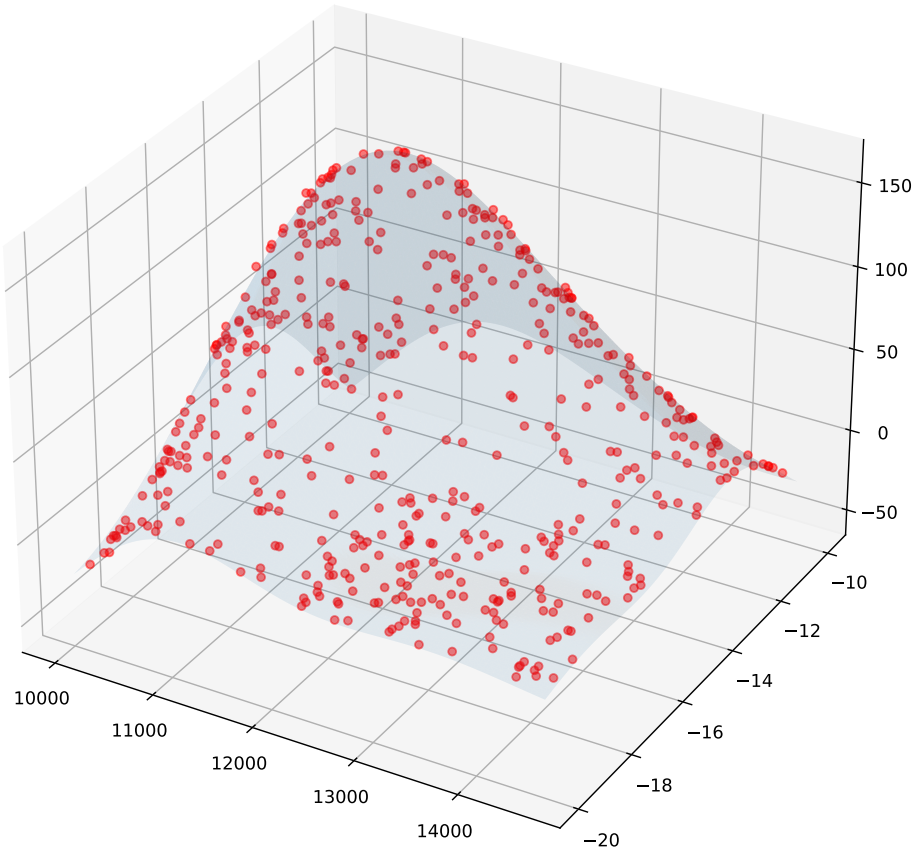
C = 0.01, epsilon = 1



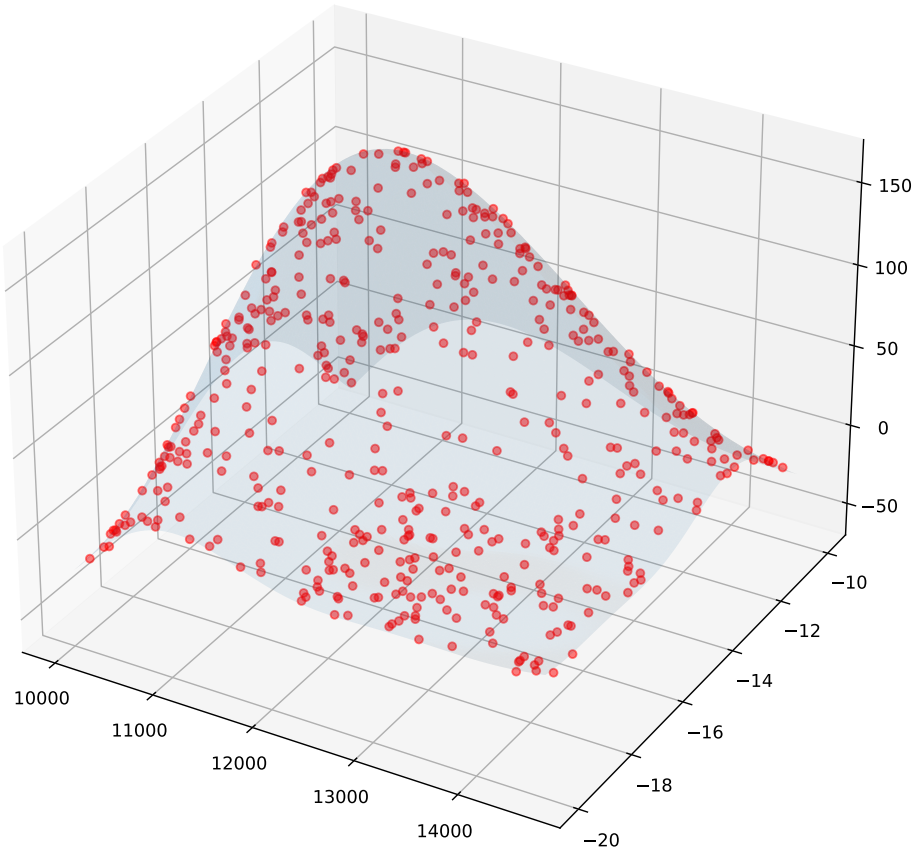

C = 10, epsilon = 0.0001
150
100
50
0
−50
10000
11000
12000
13000
14000
−10
−12
−14
−16
−18
−20


C = 10, epsilon = 0.01
150
100
50
0
−50
10000
11000
12000
13000
14000
−10
−12
−14
−16
−18
−20

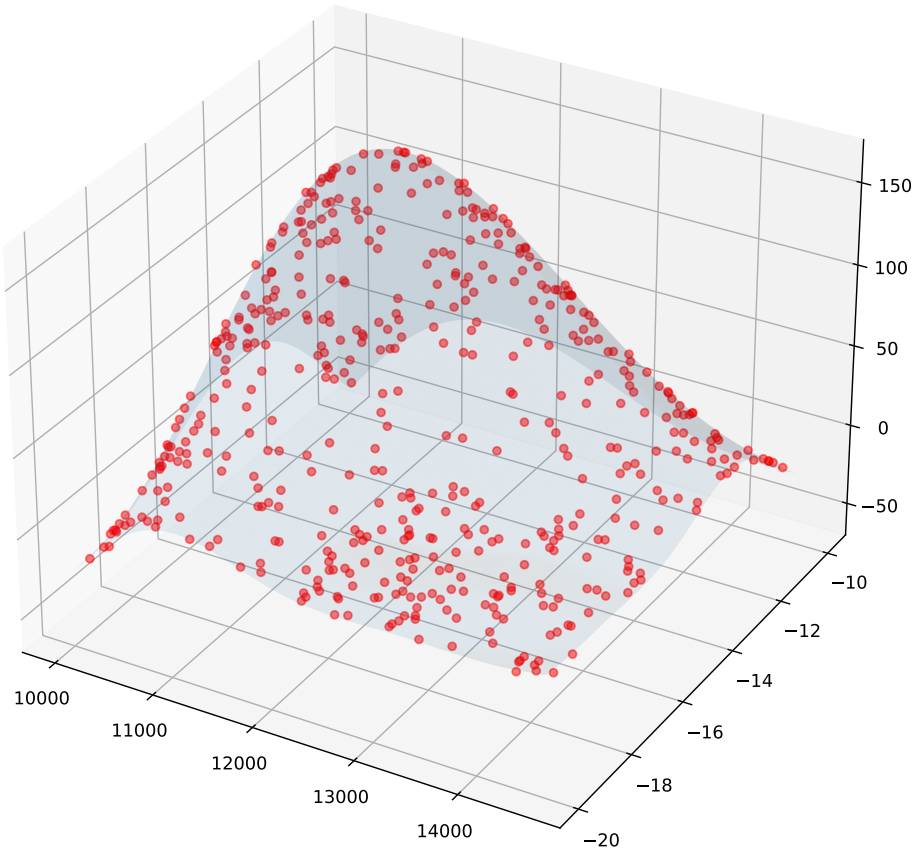
C = 10, epsilon = 1

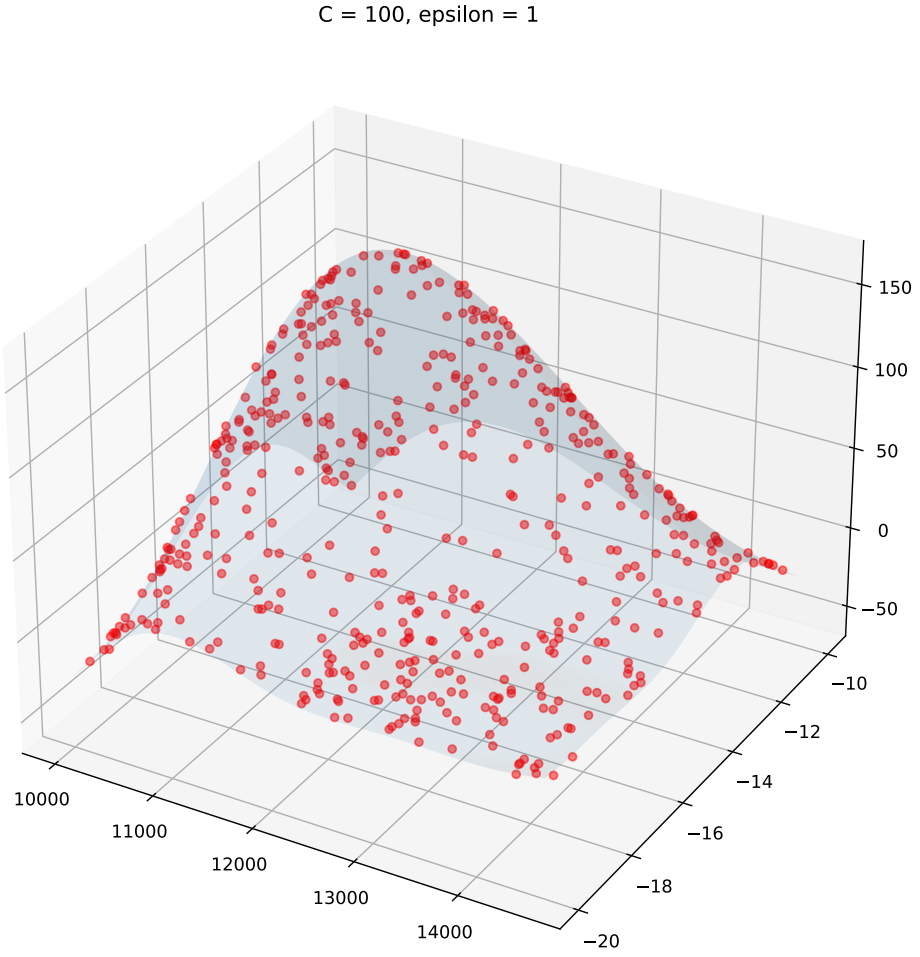


C = 100, epsilon = 0.0001



C = 100, epsilon = 0.01




C = 100, epsilon = 1
150
100
50
0
−50
−10
−12
−14
−16
−18
−20
10000
11000
12000
13000
14000

In [ ]:

Homework 6 Written Answers

Exercise 1c.
From the 4 different penalty strengths C that are observed above, there seems to be little difference shown across the 4 graphs. The general trend for the classification boundaries of the nonlinear data set is that the main boundary and +1 margin both shift to the right along +x axis, all the while -1 margin seemingly stays put. This behavior shows that the actual margin is shrinking as the value for C goes up, and alludes to a greater penalty for having a large margin.

Exercise 2b.
Based on the observations from the graphs above for SVC, it seems like the rbf kernel provides the best classification boundary contour. In terms of penalty strengths C, the number of points within the boundary decreases as the value of C increases. This is true because we are enforcing a harder (narrower) margin in the model. For the Moons dataset, I will pick the one with least support vectors (100 support vectors) as that model is more generalizable onto other data sets being tested.