Amir Hossein Eyvazkhani
1747696
Ex 10

Task 1) let x_A be any point in the hyperplane, hence $x_A^T\theta + b = 0$ (I)

Imagine a point x_0 with a d distance with the plane.

$$\Rightarrow x_A + d = x_0 \Rightarrow x_A = x_0 - d \overset{\text{I}}{\Rightarrow} (x_0 - d)^T\theta + b = 0 \Rightarrow x_0^T\theta + b = d^T\theta$$

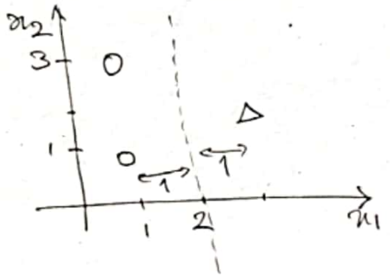
because d is paralel to the normal vector of the hyperplane, we have $d = \lambda\theta$

$$\Rightarrow x_0^T\theta + b = \lambda\theta^T\theta \Rightarrow \lambda = \frac{x_0^T\theta + b}{\theta^T\theta} \Rightarrow d = \frac{x_0^T\theta + b}{\theta^T\theta}\,\theta$$

$$\|d\| = \sqrt{d^Td} = \sqrt{\lambda^2\theta^T\theta} = \lambda\sqrt{\theta^T\theta} \Rightarrow d = \frac{x_0^T\theta + b}{\theta^T\theta}\cdot\sqrt{\theta^T\theta} = \frac{x_0^T\theta + b}{\sqrt{\theta^T\theta}} = \frac{x_0^T\theta + b}{\|\theta\|}$$

Task2)

a)



$x_1 = 2 \Rightarrow (-1\ \ 0)\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + 2 = 0$

$\Rightarrow \theta = (-1\ \ 0),\ b = 2$

$\|\theta\| = 1$ already normalized here

$c = 1$

b) hard margin primal problem is:

minimize $\frac{1}{2}\|\theta\|^2$

s.t. $y_n(x_n^T\theta + b) \geqslant 1$ $\xrightarrow{\text{for all data points}}$

$$\begin{cases} 1\left((1\ \ 1)\begin{pmatrix} -1 \\ 0 \end{pmatrix}\right) + 2 = 1 \geqslant 1 \ ✓ \\ 1\left((1\ \ 3)\begin{pmatrix} -1 \\ 0 \end{pmatrix}\right) + 2 = 1 \geqslant 1 \ ✓ \\ -1\left((3\ \ 2)\begin{pmatrix} -1 \\ 0 \end{pmatrix}\right) + 2 = 1 \geqslant 1 \ ✓ \end{cases}$$

→ inequality constraints are satisfied

for the $x_1 = (1\ \ 1)$ we have $\theta_1 + \theta_2 + 2 \geqslant 1 \Rightarrow \theta_1 + \theta_2 \geqslant -1$

$x_2 = (1\ \ 3) \rightarrow \theta_1 + 3\theta_2 \geqslant -1$

$x_3 = (3\ \ 2) \rightarrow -3\theta_1 - 2\theta_2 \geqslant -1$

→ these show that any other θ_1, θ_2 would result in a higher norm

(c) $\theta^* = \sum_{n=1}^{3} \alpha_n^* x_n y_n \Rightarrow \begin{pmatrix}-1\\0\end{pmatrix} = \begin{pmatrix}1\\1\end{pmatrix}\alpha_1 + \begin{pmatrix}1\\3\end{pmatrix}\alpha_2 - \begin{pmatrix}3\\2\end{pmatrix}\alpha_3$

$$\Rightarrow \begin{cases} \alpha_1+\alpha_2-3\alpha_3=-1 \\ \alpha_1+3\alpha_2-2\alpha_3=0 \end{cases} \quad \text{(I)}$$

also we have $\sum y_n \alpha_n = 0 \Rightarrow \alpha_1+\alpha_2-\alpha_3=0$ and $\forall \alpha \geq 0$ (II)

$$\xrightarrow[\text{Solving these System of linear equations we get}]{\text{(I),(II)}} (\alpha_1^*, \alpha_2^*, \alpha_3^*) = \left(\frac{1}{4}, \frac{1}{4}, \frac{1}{2}\right)$$

(3) k must be positive indefinite; but the kernel function Provided is not. here is an example:

$$\begin{matrix} a=(1,-2) \\ b=(0,+1) \end{matrix} \rightarrow k(a,b) = \frac{-2}{\sqrt{5}} < 0$$

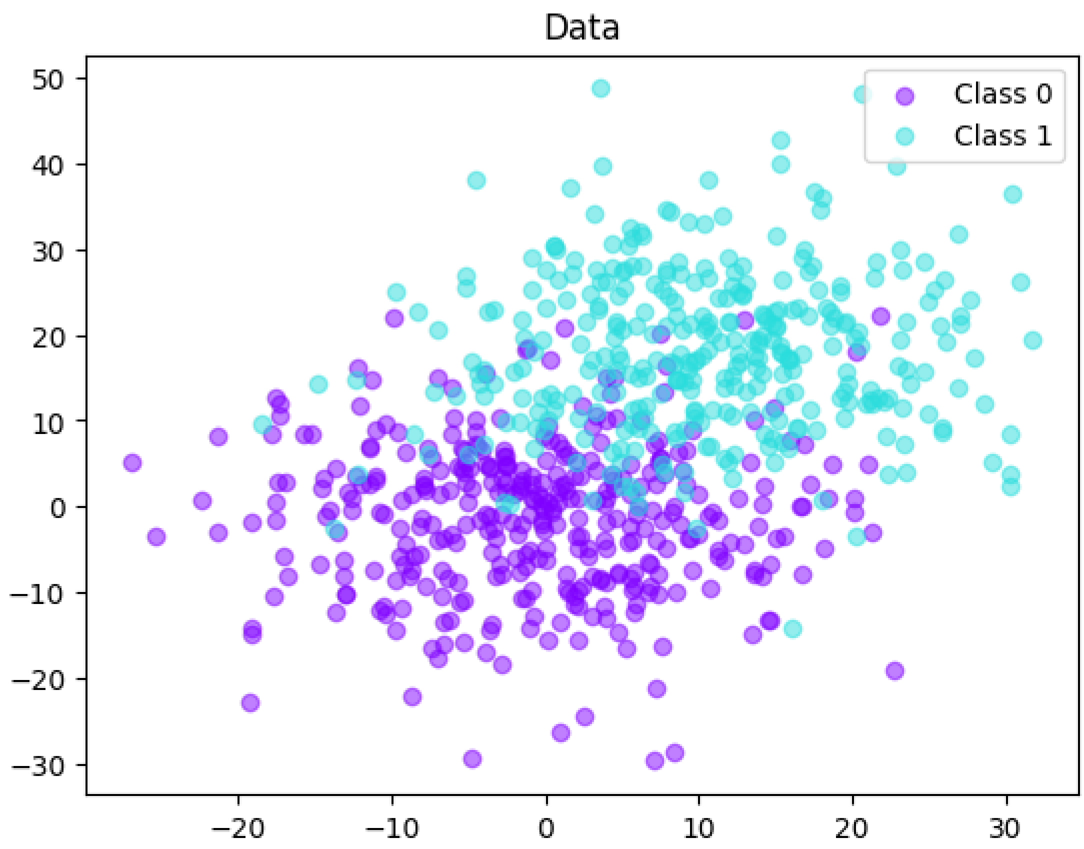
Task 4

In [ ]:
```
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn import svm
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```
# The following lines generate a random set of points in the 2D space. Please refer to make_blobs function in scikit-
X,Y = make_blobs(n_samples=1000, n_features=2, centers=np.array([[0,0],[10,18]]), cluster_std=np.array([9.0,9.0]))
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.30, shuffle=True)
```

In [ ]:
```
def plot_dataset(x,y):
    # This function would plot the generated points
    plt.figure()
    unique_classes = np.unique(y)
    colors = cm.magma(np.linspace(0.0,1.0), unique_classes.size)
    rainbow = cm.get_cmap('rainbow',4)
    for this_class in unique_classes:
        color = rainbow(this_class)
        indices = np.where(y == this_class)
        points = x[indices]
        plt.scatter(
            points[:,0],
            points[:,1],
            color=color,
            label="Class {}".format(this_class),
            alpha=0.5
        )
        plt.title('Data')
    plt.legend()
    plt.show()
```

In [ ]:
```python
plot_dataset(X_train,Y_train)
```



In [ ]:
```python
# The following lines learn a SVM over the generated data.
# Please refer to the svm.SVC() class in scikit-learn for further details.
clf = svm.SVC(kernel='linear', degree=7, C=20, max_iter=1000,  verbose=True)
```

In [ ]:
```python
def fit_data(clf, train_features, train_labels, normalize=False):
    if normalize:
        normalizer = StandardScaler().fit(train_features)
        data = normalizer.transform(train_features)
    else:
        data = train_features
        normalizer=None
```

```
    clf.fit(data, train_labels)
    return clf, normalizer
```

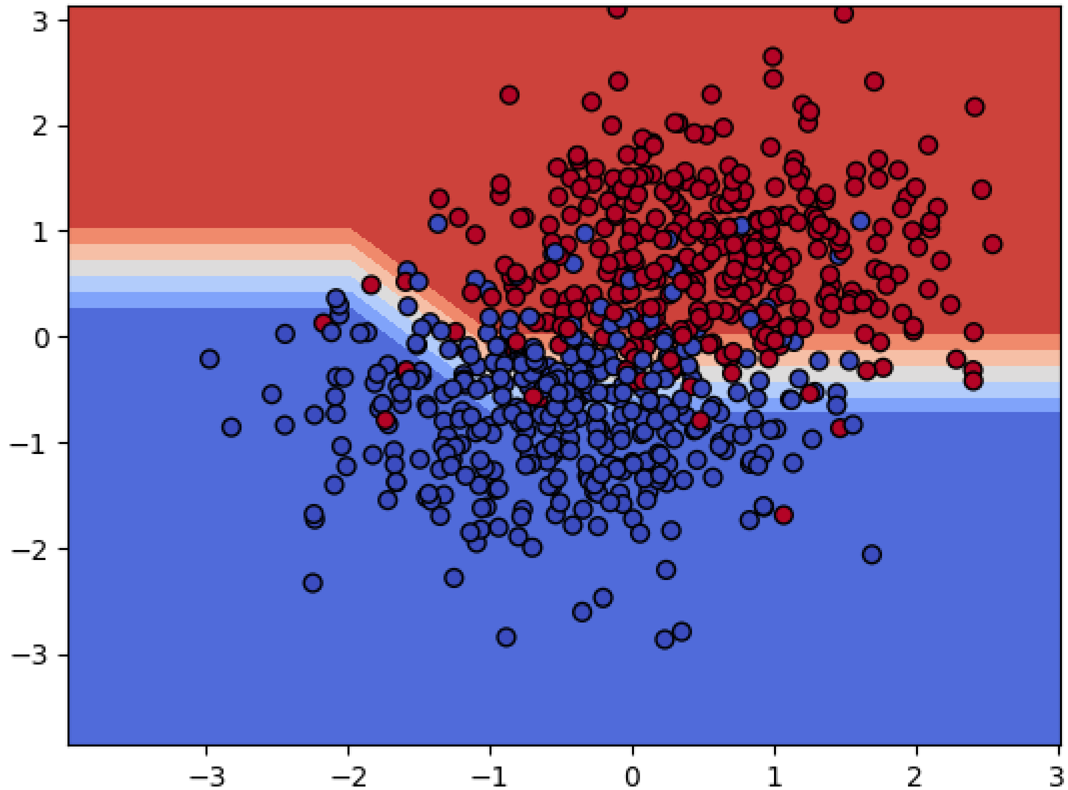
In [ ]:
```
clf, normalizer = fit_data(clf, X_train, Y_train, normalize=True)
```

```
[LibSVM]
```

In [ ]:
```
# These are helper functions. Please do not modify them for this tutorial

def make_meshgrid(x, y, h=1):
    x_min, x_max = x.min() - 1, x.max() + 1
    y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    return xx, yy

def plot_contours(ax, clf, xx, yy, **params):
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    out = ax.contourf(xx, yy, Z, **params)
    return out
```

In [ ]:
```
# This function plots the learnt decision boundary.
# You will need to modify this function to plot the support vectors
def plot_decision_boundary(clf, x,y, normalizer=None):
    if normalizer is not None:
        x = normalizer.transform(x)
    xx,yy = make_meshgrid(x[:,0], x[:,1])
    fig, ax = plt.subplots()
    plot_contours(ax, clf, xx, yy, cmap=cm.coolwarm, alpha=1.0, normalizer=normalizer)
    ax.scatter(x[:,0], x[:,1], c=y, cmap=plt.cm.coolwarm, s=40, edgecolors='k')
    plt.show()
```

In [ ]:
```
plot_decision_boundary(clf,X_train,Y_train, normalizer=normalizer)
```



In [ ]:
```
def predict_test(clf,x_test, y_test, normalizer=None):
# If normalizer is None, then the data will be directly predicted and the accuracy comp
# Otherwise, the x_test should be normalized using the provided normalizer and then pre
# Please refer to the documentation of StandardScaler in sklearn to see how to do this.
    if normalizer:
        normalizer = StandardScaler().fit(x_test)
        x_test = normalizer.transform(x_test)
    else:
        x_test = x_test
        normalizer=None
    pred=clf.predict(x_test)
    accuracy=(pred==y_test).mean()
    return accuracy
```

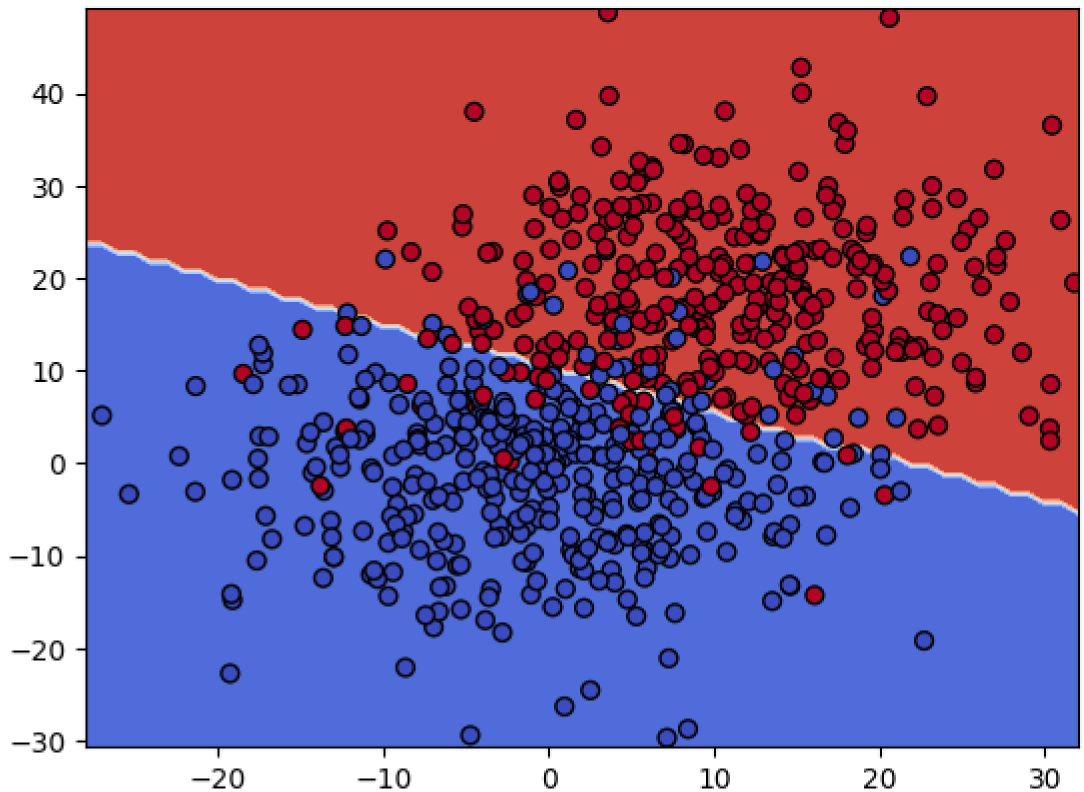
In [ ]:
```
print(predict_test(clf,X_test,Y_test, normalizer))
```

0.67

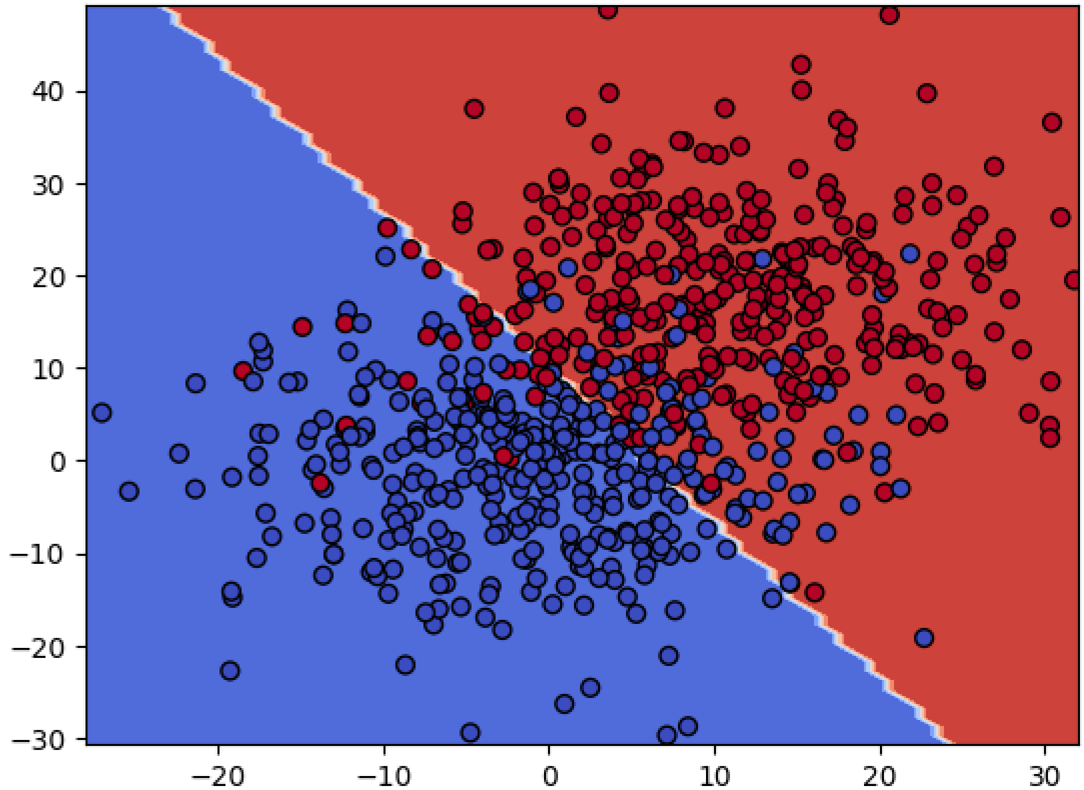
Part a)

```
In [ ]: C=np.array([0.1, 1, 10, 20, 50])
for c in C:
    clf = svm.SVC(kernel='linear', degree=7, C=c, max_iter=1000, verbose=False)
    clf, normalizer= fit_data(clf, X_train, Y_train, normalize=False)
    print('\nFor C=',c,' the number of support vectors for each class {0,1} is',clf.n_support_)
    print('\nFor C=',c,' the accuracy is: %0.2f'%predict_test(clf,X_test,Y_test, normalizer))
    plot_decision_boundary(clf,X_train,Y_train, normalizer=normalizer)
```

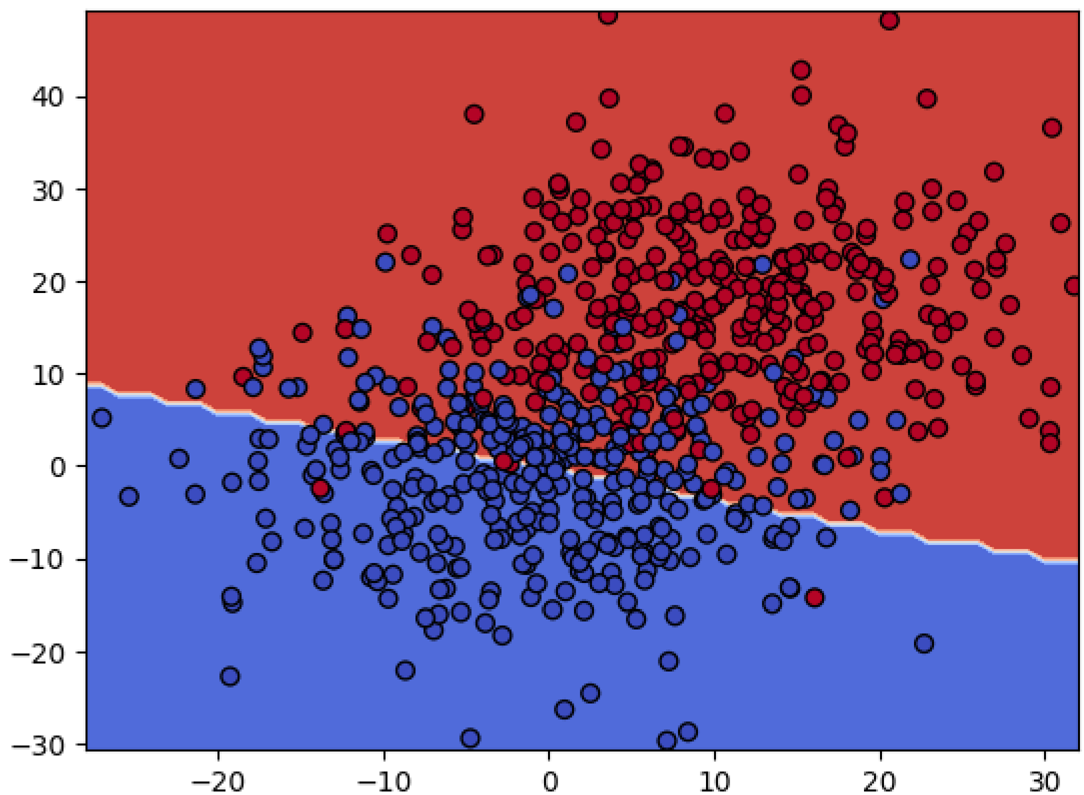
```
For C= 0.1  the number of support vectors for each class {0,1} is [97 96]

For C= 0.1  the accuracy is: 0.86
```



```
For C= 1.0  the number of support vectors for each class {0,1} is [ 71 107]

For C= 1.0  the accuracy is: 0.81
```

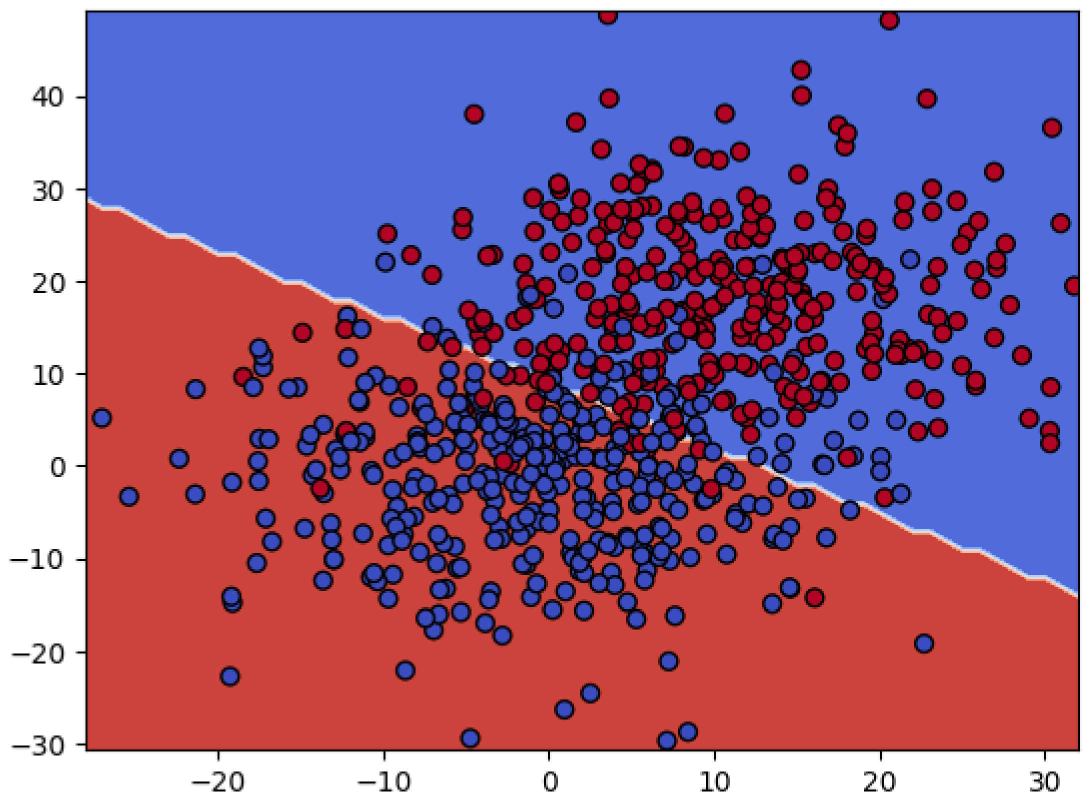



```
For C= 10.0  the number of support vectors for each class {0,1} is [38 64]

For C= 10.0  the accuracy is: 0.76
```

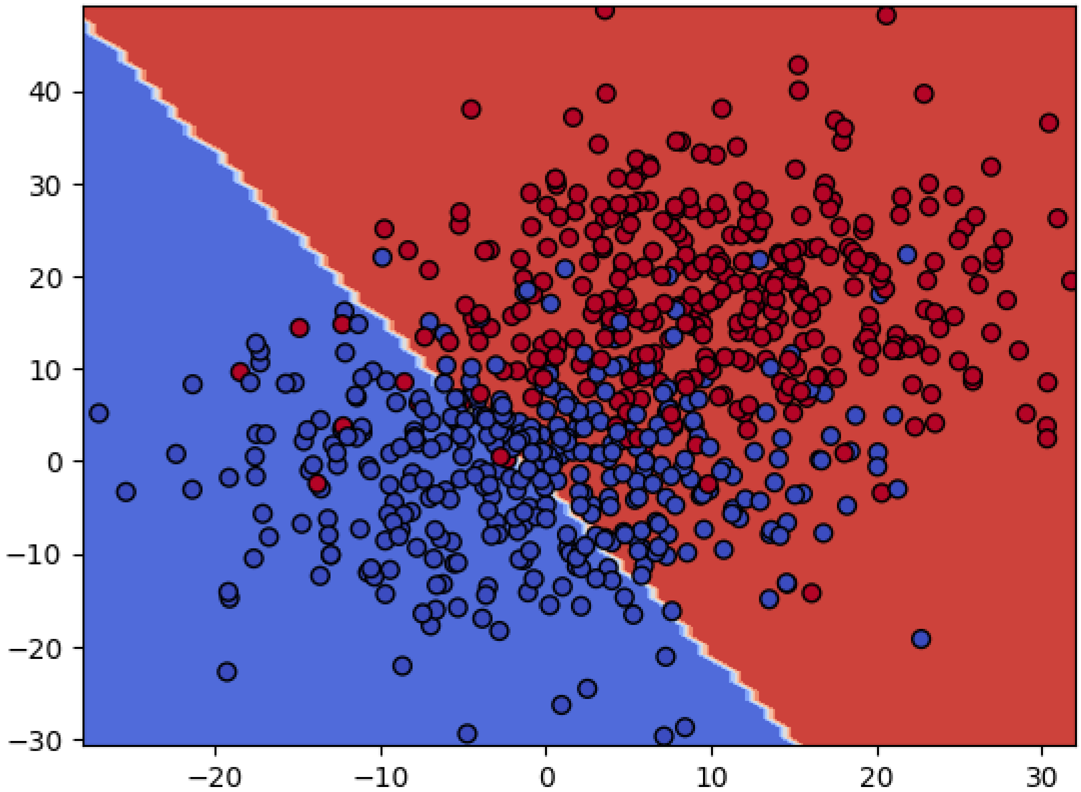


For C= 20.0 the number of support vectors for each class {0,1} is [28 46]

For C= 20.0 the accuracy is: 0.16




```
For C= 50.0  the number of support vectors for each class {0,1} is [19 34]

For C= 50.0  the accuracy is: 0.75
```

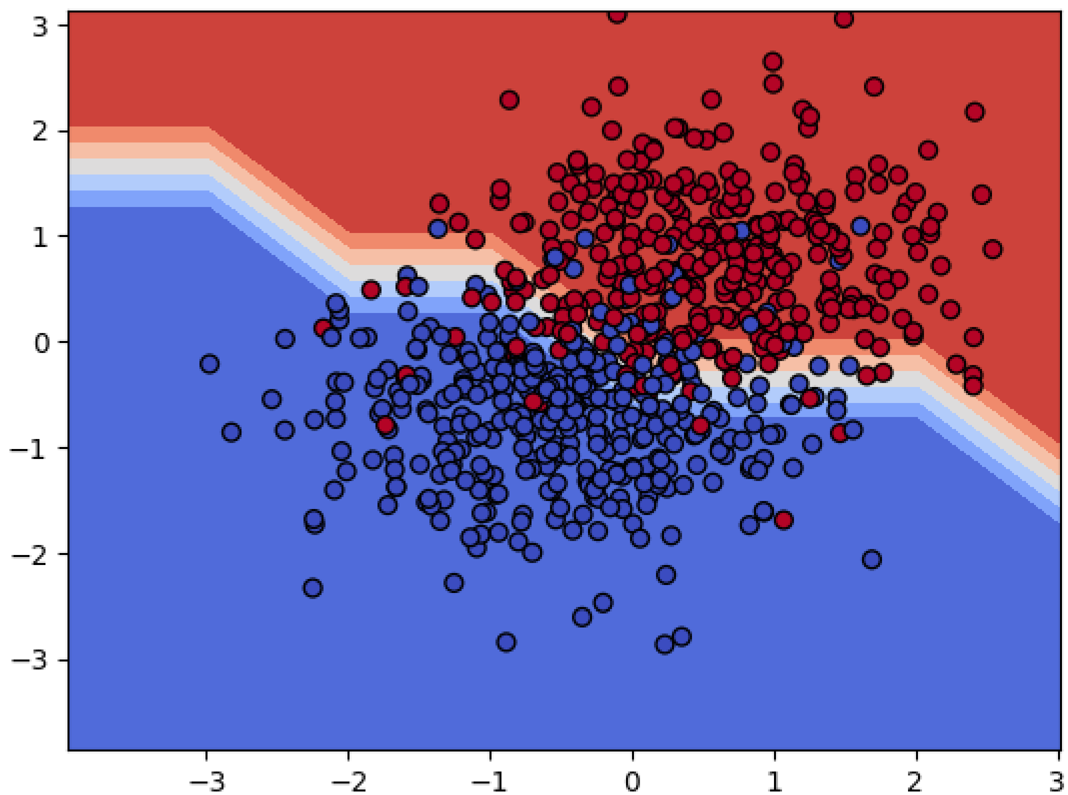


higher C would result in less regularization. hence we would have a more complex model, and less points to be misclassified (more tolerant to it) ; also we may have a larger margin. We have less number of support vectors

Part b)

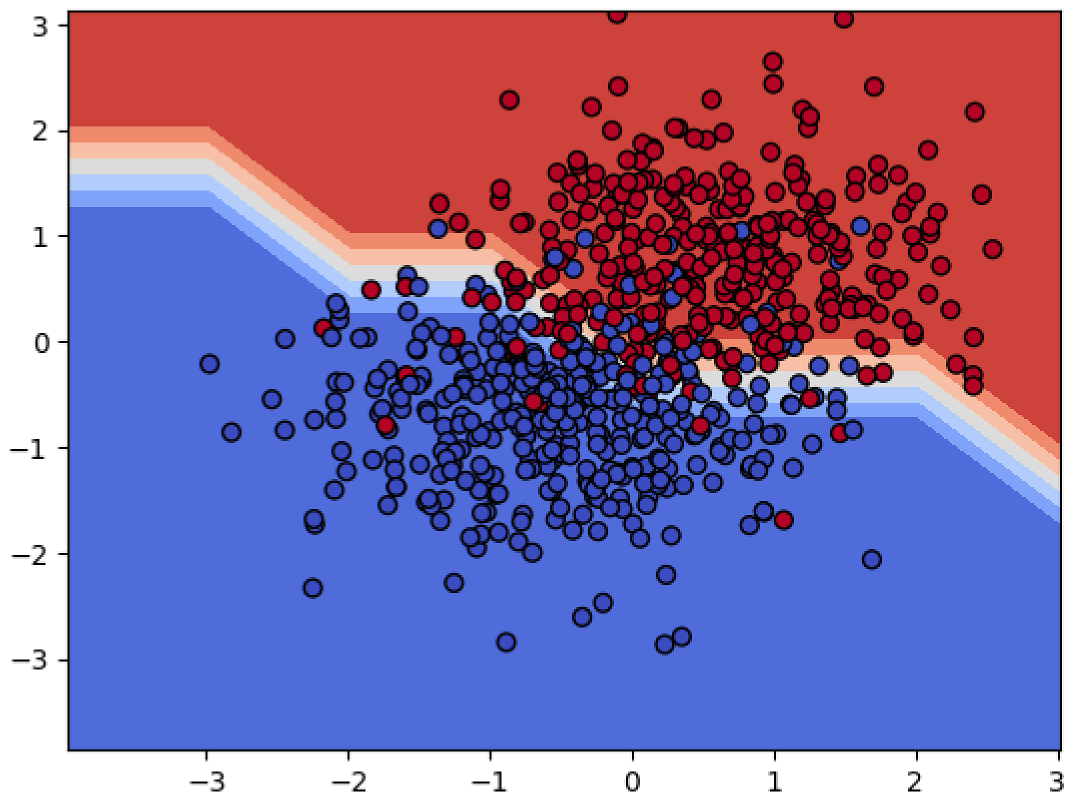
In [ ]:
```
C=np.array([0.1, 1, 10, 20, 50])
for c in C:
    clf = svm.SVC(kernel='linear', degree=7, C=c, max_iter=1000, verbose=False)
    clf, normalizer= fit_data(clf, X_train, Y_train, normalize=True)
    print('\nFor C=',c,' the number of support vectors for each class {0,1} is',clf.n_support_)
    print('\nFor C=',c,' the accuracy is: %0.2f'%predict_test(clf,X_test,Y_test, normalizer))
    plot_decision_boundary(clf,X_train,Y_train, normalizer=normalizer)
```

```
For C= 0.1  the number of support vectors for each class {0,1} is [113 113]

For C= 0.1  the accuracy is: 0.85
```



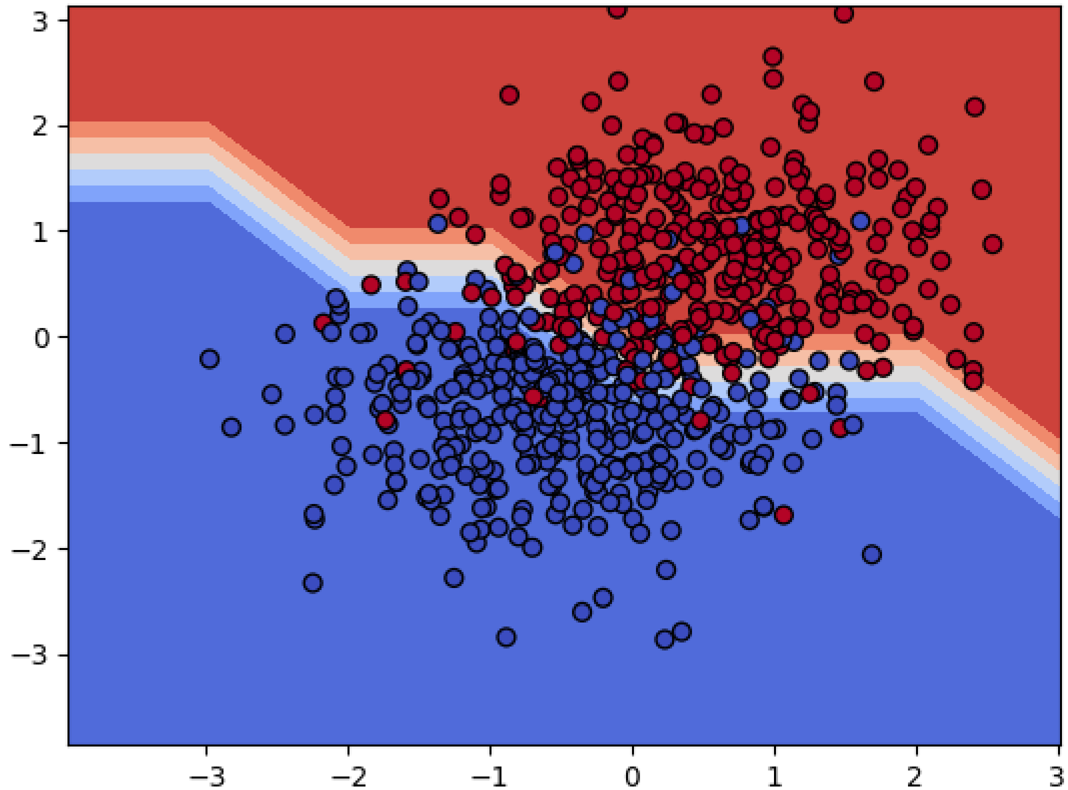
For C= 1.0 the number of support vectors for each class {0,1} is [97 98]

For C= 1.0 the accuracy is: 0.85



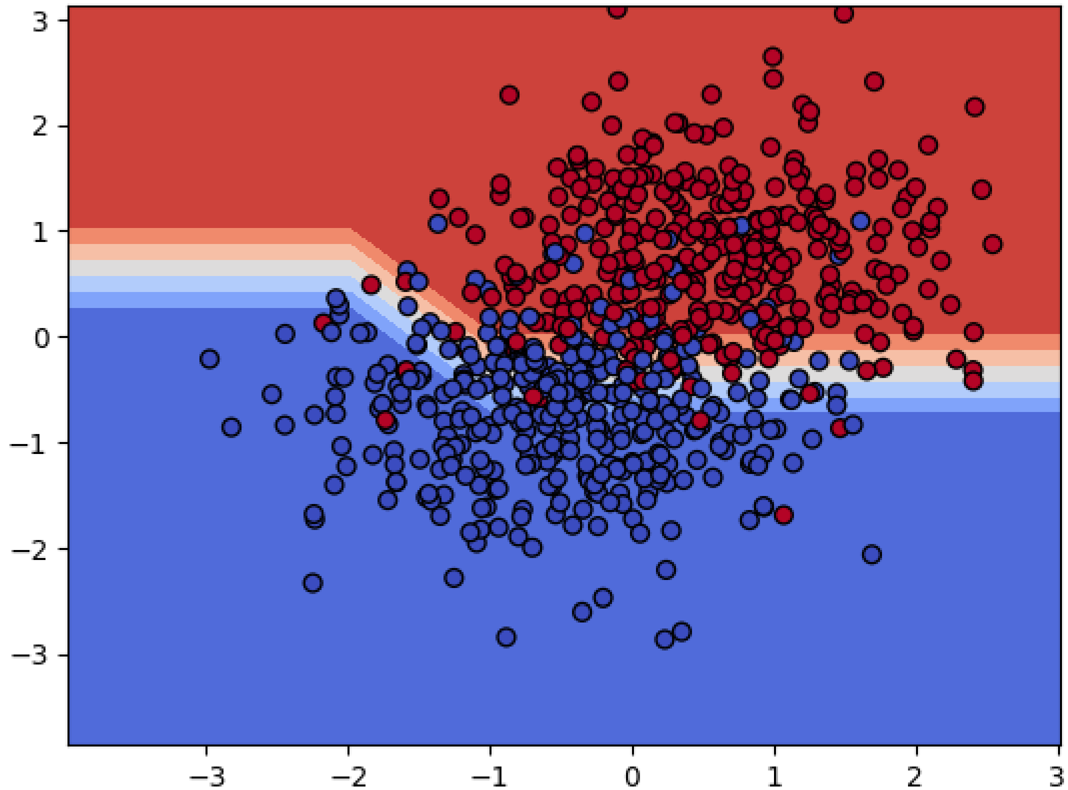
For C= 10.0 the number of support vectors for each class {0,1} is [95 96]

For C= 10.0 the accuracy is: 0.85

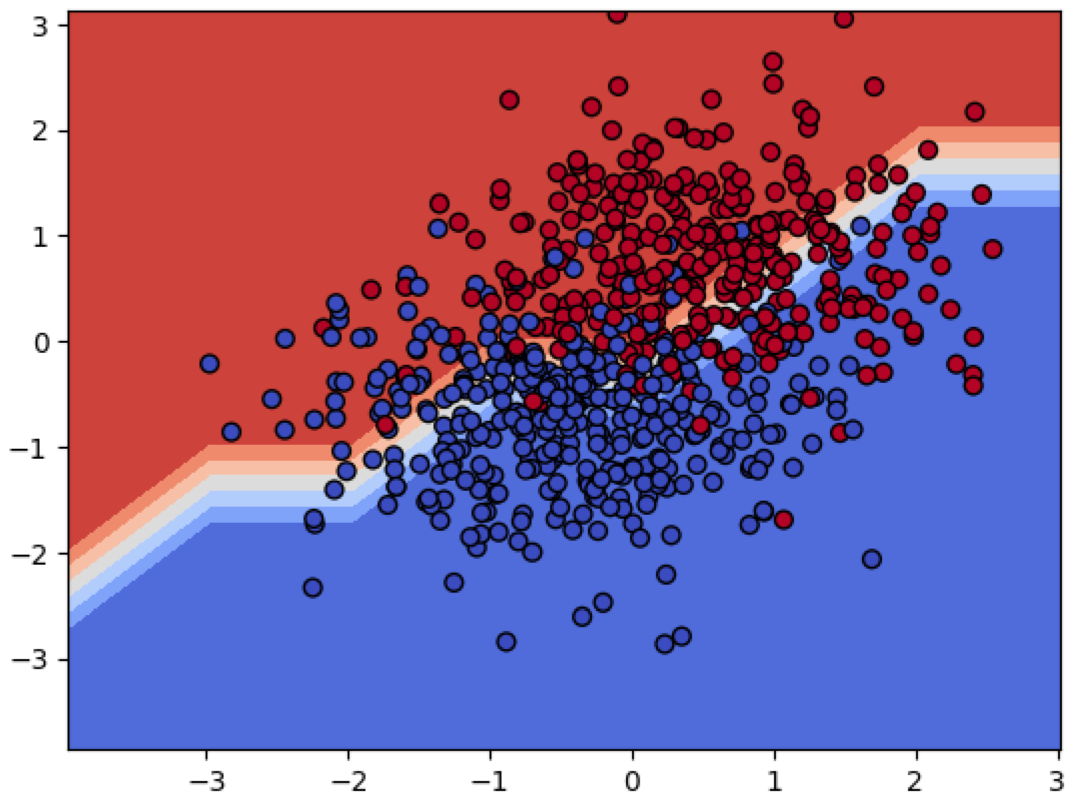


For C= 20.0 the number of support vectors for each class {0,1} is [96 94]

For C= 20.0 the accuracy is: 0.86




```
For C= 50.0  the number of support vectors for each class {0,1} is [92 92]

For C= 50.0  the accuracy is: 0.67
```

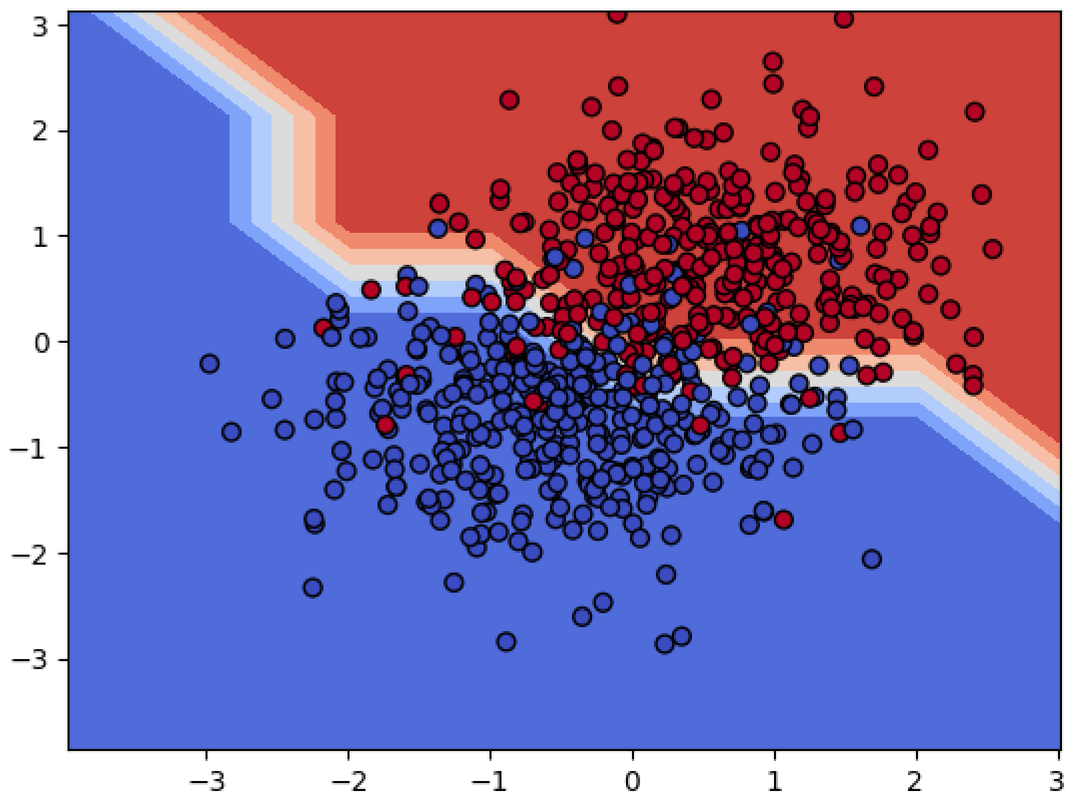


Normalizing the data helps in achieving faster convergence during the training process. Features with larger scales might dominate the optimization process without normalization, leading to longer training times.

part c)

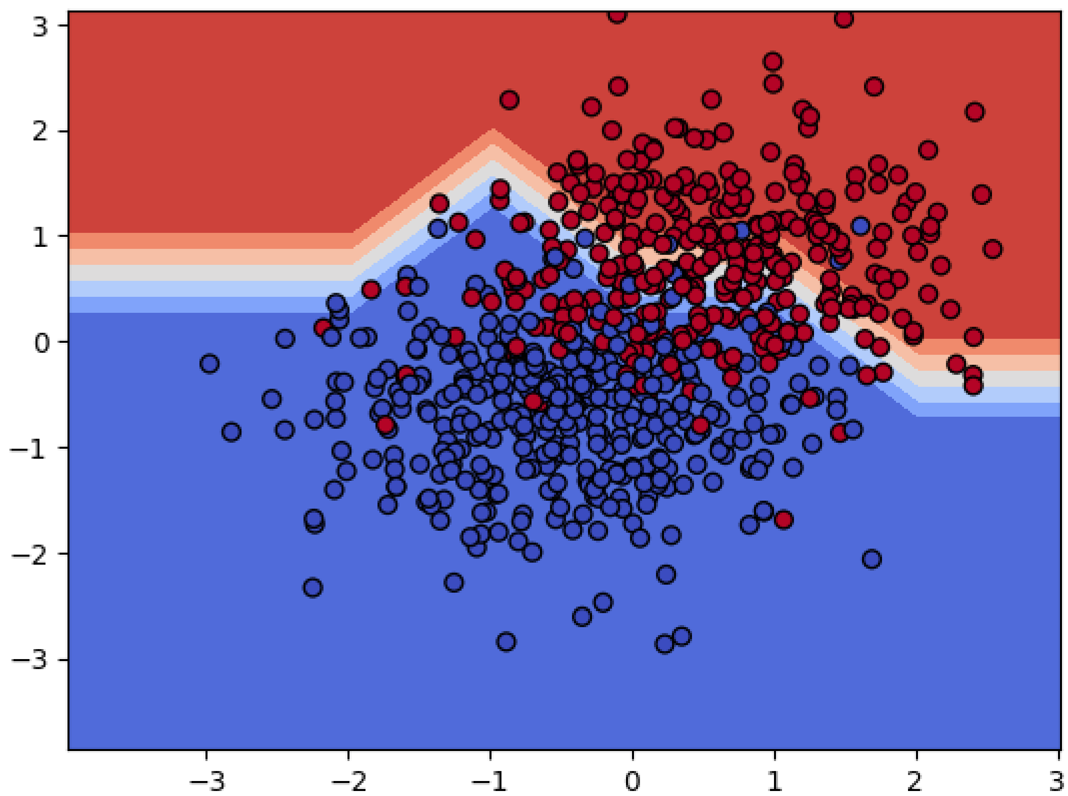
In [ ]:
```
c = 1
clf = svm.SVC(kernel='rbf', degree=7, C=c, max_iter=1000, verbose=False)
clf, normalizer= fit_data(clf, X_train, Y_train, normalize=True)
print('\nFor C=',c,' the number of support vectors for each class {0,1} is',clf.n_support_)
print('\nFor C=',c,' the accuracy is: %0.2f'%predict_test(clf,X_test,Y_test, normalizer))
plot_decision_boundary(clf,X_train,Y_train, normalizer=normalizer)
```

```
For C= 1  the number of support vectors for each class {0,1} is [103 102]

For C= 1  the accuracy is: 0.86
```



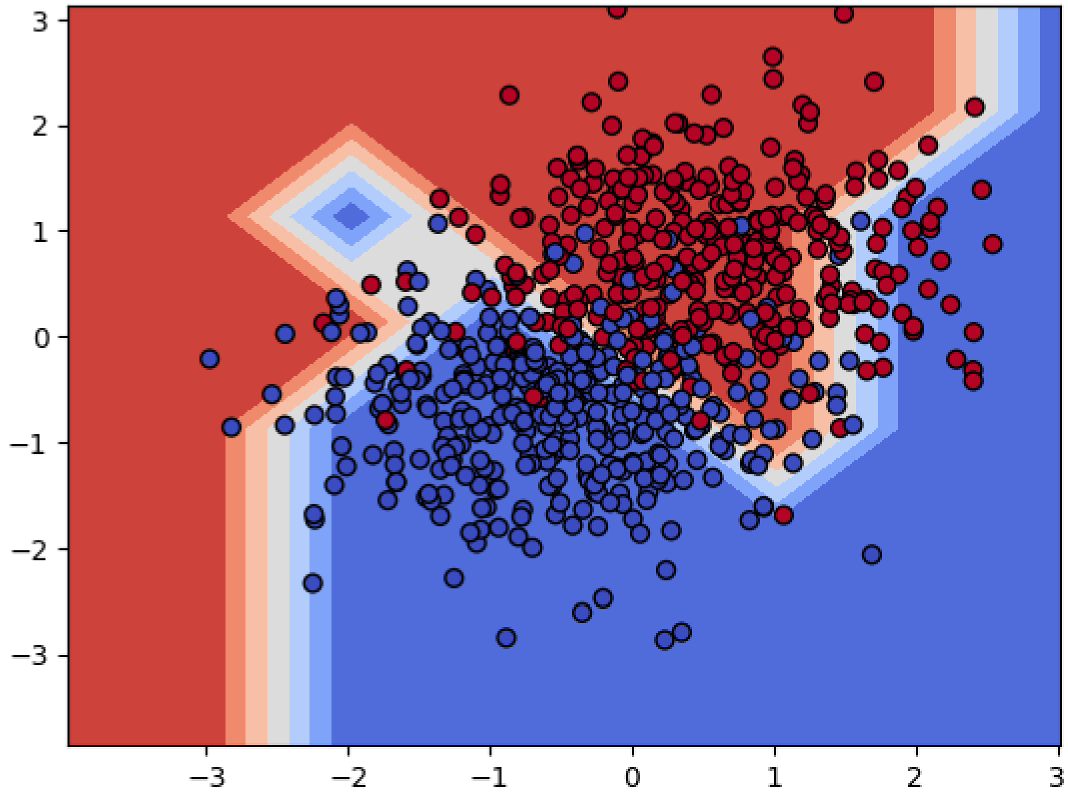
In [ ]:
```
c = 1
clf = svm.SVC(kernel='poly', degree=7, C=c, max_iter=1000, verbose=False)
clf, normalizer= fit_data(clf, X_train, Y_train, normalize=True)
print('\nFor C=',c,' the number of support vectors for each class {0,1} is',clf.n_support_)
print('\nFor C=',c,' the accuracy is: %0.2f'%predict_test(clf,X_test,Y_test, normalizer))
plot_decision_boundary(clf,X_train,Y_train, normalizer=normalizer)
```

```
For C= 1  the number of support vectors for each class {0,1} is [205 204]

For C= 1  the accuracy is: 0.65
```




```
In [ ]: c = 1
        clf = svm.SVC(kernel='sigmoid', degree=7, C=c, max_iter=1000, verbose=False)
        clf, normalizer= fit_data(clf, X_train, Y_train, normalize=True)
        print('\nFor C=',c,' the number of support vectors for each class {0,1} is',clf.n_support_)
        print('\nFor C=',c,' the accuracy is: %0.2f'%predict_test(clf,X_test,Y_test, normalizer))
        plot_decision_boundary(clf,X_train,Y_train, normalizer=normalizer)
```

```
For C= 1  the number of support vectors for each class {0,1} is [74 75]

For C= 1  the accuracy is: 0.79
```



RBF kernel has the better accuracy here