practica-4

October 3, 2024

1 Práctica 4

1.0.1 Dataset 1

[Pregunta 1] Abre este script y explica su contenido. Podras ver que se utiliza el primer dataset de ejemplo y se realiza la representacion gráfica del SVM. Comenta que tipo de kernel se esta utilizando y cuáles son los parámetros de entrenamiento.

[9]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

from sklearn import svm
```

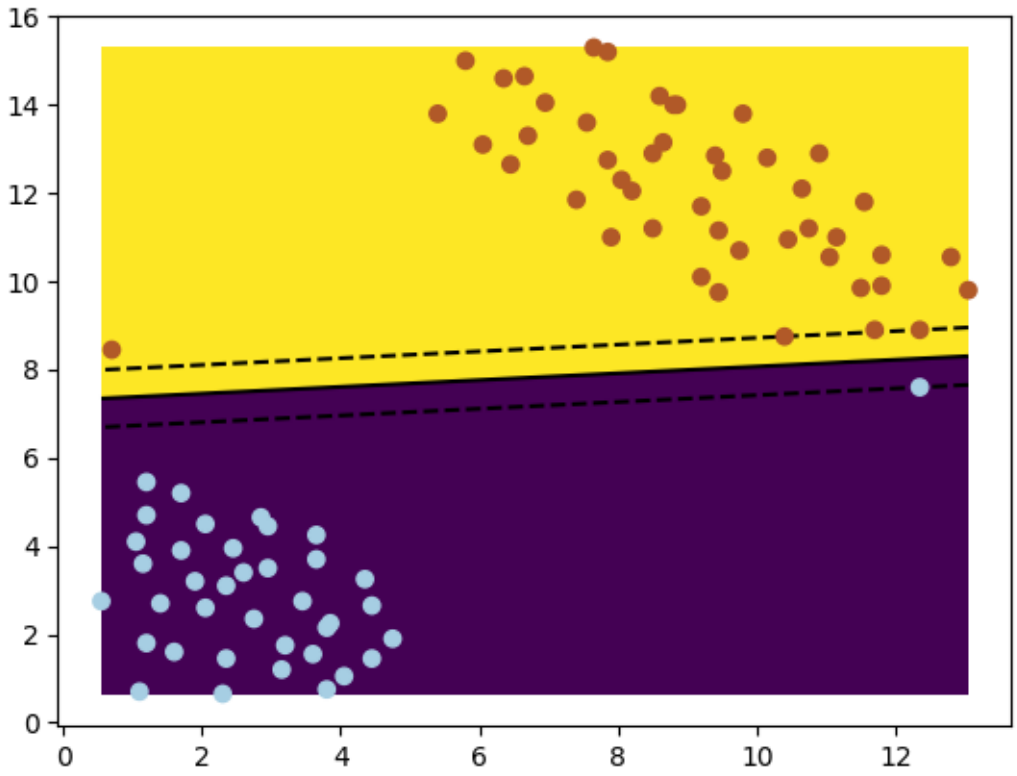
[36]:
```
def libsvmScript(dataset, kernelType='linear', C_value=100, gammaValue=0):
    # Load the dataset
    data = pd.read_csv(dataset,header=None)
    X = data.iloc[:,:-1].values
    y = data.iloc[:,-1].values

    # Train the SVM model
    svm_model = svm.SVC(kernel=kernelType,C=C_value, gamma=gammaValue)
    svm_model.fit(X, y)

    # Show the points
    plt.figure(1)
    plt.clf()
    plt.scatter(X[:, 0], X[:, 1], c=y, zorder=10, cmap=plt.cm.Paired)

    # Show the separating hyperplane
    plt.axis('tight')
    # Extract the limit of the data to construct the mesh
    x_min = X[:, 0].min()
    x_max = X[:, 0].max()
    y_min = X[:, 1].min()
    y_max = X[:, 1].max()

    # Create the mesh and obtain the Z value returned by the SVM
    XX, YY = np.mgrid[x_min:x_max:500j, y_min:y_max:500j]
```

```
    Z = svm_model.decision_function(np.c_[XX.ravel(), YY.ravel()])

    # Make a color plot including the margin hyperplanes (Z=-1 and Z=1) and the
    # separating hyperplane (Z=0)
    Z = Z.reshape(XX.shape)
    plt.pcolormesh(XX, YY, Z > 0)
    plt.contour(XX, YY, Z, colors=['k', 'k', 'k'], linestyles=['--', '-', '--'],
                    levels=[-1, 0, 1])

    plt.show()
```

[61]:
```
def showDataset(dataset, kernelType='linear', C_value=100, gammaValue=0):
    # Load the dataset
    data = pd.read_csv(dataset,header=None)
    X = data.iloc[:,:-1].values
    y = data.iloc[:,-1].values

    # Train the SVM model
    svm_model = svm.SVC(kernel=kernelType,C=C_value, gamma=gammaValue)
    svm_model.fit(X, y)

    # Show the points
    plt.figure(1)
    plt.clf()
    plt.scatter(X[:, 0], X[:, 1], c=y, zorder=10, cmap=plt.cm.Paired)

    # Show the separating hyperplane
    # plt.axis('tight')
    # Extract the limit of the data to construct the mesh
    x_min = X[:, 0].min()
    x_max = X[:, 0].max()
    y_min = X[:, 1].min()
    y_max = X[:, 1].max()

    # Create the mesh and obtain the Z value returned by the SVM
    XX, YY = np.mgrid[x_min:x_max:500j, y_min:y_max:500j]
    Z = svm_model.decision_function(np.c_[XX.ravel(), YY.ravel()])

    # Make a color plot including the margin hyperplanes (Z=-1 and Z=1) and the
    # separating hyperplane (Z=0)
    Z = Z.reshape(XX.shape)
    # plt.pcolormesh(XX, YY, Z > 0)
    # plt.contour(XX, YY, Z, colors=['k', 'k', 'k'], linestyles=['--', '-',␣
↪'--'],
    #                 levels=[-1, 0, 1])

    plt.show()
```

[83]:
```
def libsvmScriptNoColor(dataset, kernelType='linear', C_value=100,␣
↪gammaValue=0):
    # Load the dataset
    data = pd.read_csv(dataset,header=None)
    X = data.iloc[:,:-1].values
    y = data.iloc[:,-1].values

    # Train the SVM model
    svm_model = svm.SVC(kernel=kernelType,C=C_value, gamma=gammaValue)
    svm_model.fit(X, y)

    # Show the points
    plt.figure(1)
    plt.clf()
    plt.scatter(X[:, 0], X[:, 1], c=y, zorder=10, cmap=plt.cm.Paired)

    # Show the separating hyperplane
    plt.axis('tight')
    # Extract the limit of the data to construct the mesh
    x_min = X[:, 0].min()
    x_max = X[:, 0].max()
    y_min = X[:, 1].min()
    y_max = X[:, 1].max()

    # Create the mesh and obtain the Z value returned by the SVM
    XX, YY = np.mgrid[x_min:x_max:500j, y_min:y_max:500j]
    Z = svm_model.decision_function(np.c_[XX.ravel(), YY.ravel()])

    # Make a color plot including the margin hyperplanes (Z=-1 and Z=1) and the
    # separating hyperplane (Z=0)
    Z = Z.reshape(XX.shape)
    # plt.pcolormesh(XX, YY, Z > 0)
    plt.contour(XX, YY, Z, colors=['k', 'k', 'k'], linestyles=['--', '-', '--'],
                levels=[-1, 0, 1])
    plt.scatter(svm_model.support_vectors_[:, 0], svm_model.support_vectors_[:,␣
↪1], s=80,
            facecolors='none', zorder=10)

    plt.show()
```

[18]:
```
libsvmScript('dataset1.csv')
```

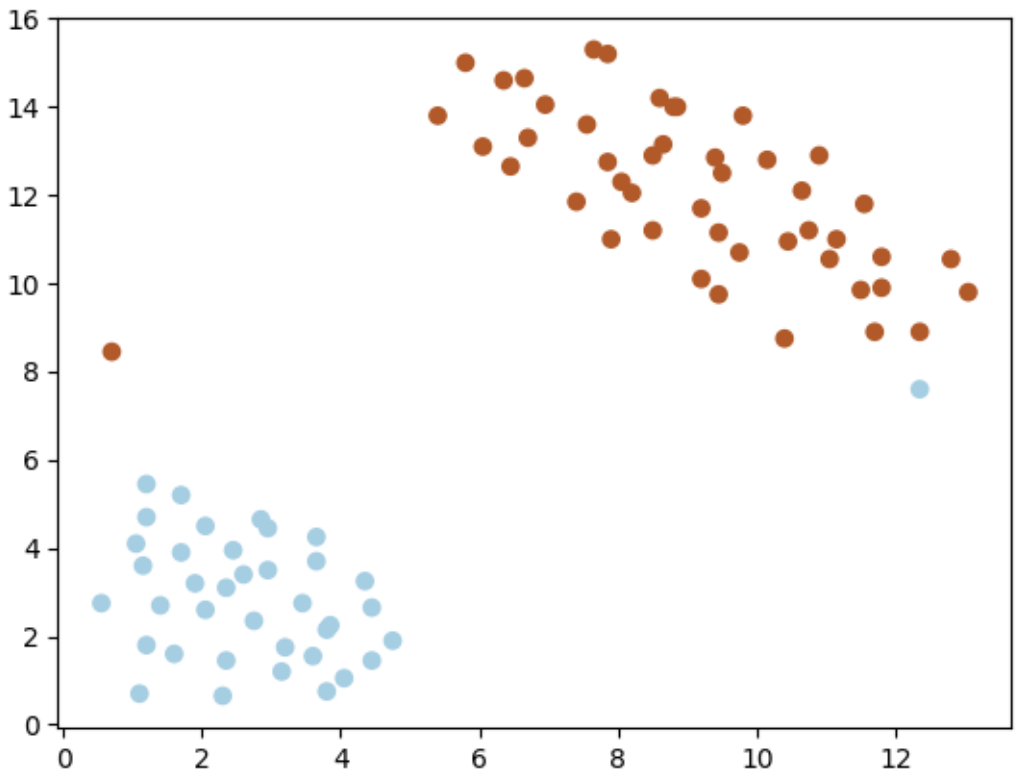


Este script contiene una SVM para clasificación, en este caso del dataset1
Primero se importan las bibliotecas necesarias (numpy, matplotlib.pyplot, pandas y svm de sklearn)
Después se carga el dataset, cargando las características en `x` y las etiquetas en `y`
Después entrena el modelo SVM con kernel lineal y el hiperparámetro `C = 100`
Por último, se genera un gráfico de dispersión de los puntos y se dibuja el hiperplano producido por el SVM.

[Pregunta 2] Intuitivamente, ¿que hiperplano crees que incurrirá en un menor error de test en la tarea de separar las dos clases de puntos?

[12]:
```
# Load the dataset
data = pd.read_csv('dataset1.csv',header=None)
X = data.iloc[:,:-1].values
y = data.iloc[:,-1].values

# Train the SVM model
svm_model = svm.SVC(kernel='linear',C=100)
svm_model.fit(X, y)

# Show the points
plt.figure(1)
plt.clf()
```

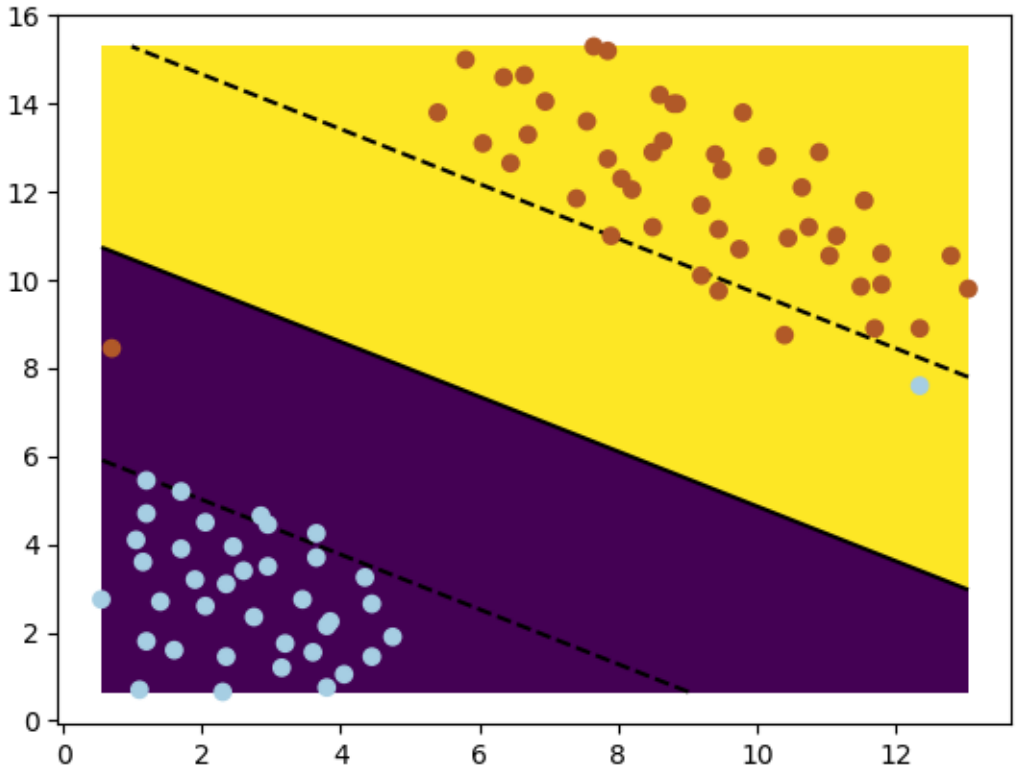
```
plt.scatter(X[:, 0], X[:, 1], c=y, zorder=10, cmap=plt.cm.Paired)

# Show the separating hyperplane
# plt.axis('tight')
# Extract the limit of the data to construct the mesh
x_min = X[:, 0].min()
x_max = X[:, 0].max()
y_min = X[:, 1].min()
y_max = X[:, 1].max()

# Create the mesh and obtain the Z value returned by the SVM
XX, YY = np.mgrid[x_min:x_max:500j, y_min:y_max:500j]
Z = svm_model.decision_function(np.c_[XX.ravel(), YY.ravel()])

# Make a color plot including the margin hyperplanes (Z=-1 and Z=1) and the
# separating hyperplane (Z=0)
Z = Z.reshape(XX.shape)
# plt.pcolormesh(XX, YY, Z > 0)
# plt.contour(XX, YY, Z, colors=['k', 'k', 'k'], linestyles=['--', '-', '--'],
                # levels=[-1, 0, 1])

plt.show()
```



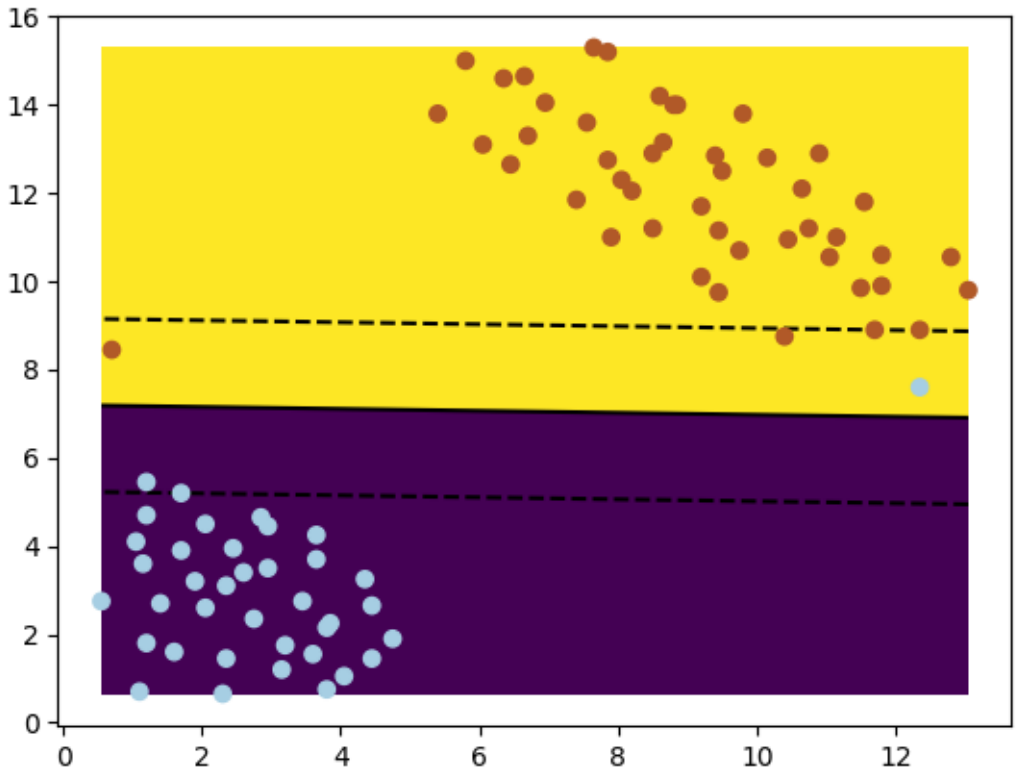
Un hiperplano horizontal con `y = 8`

[Pregunta 3] Modifica el script probando varios valores de C, en concreto, C {0.01, 0.1, 1, 10, 100, 1000, 10000}. Observa que sucede, explica por qué y escoge el valor más adecuado.

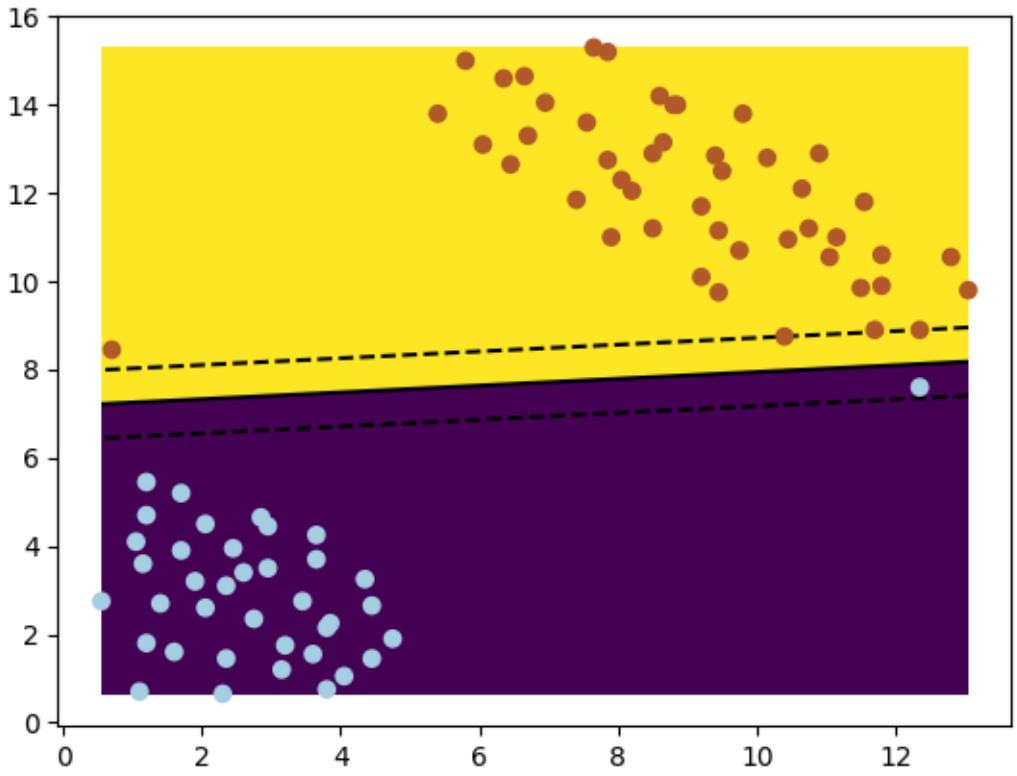
[21]:
```
libsvmScript('dataset1.csv', 'linear', 0.01)
```



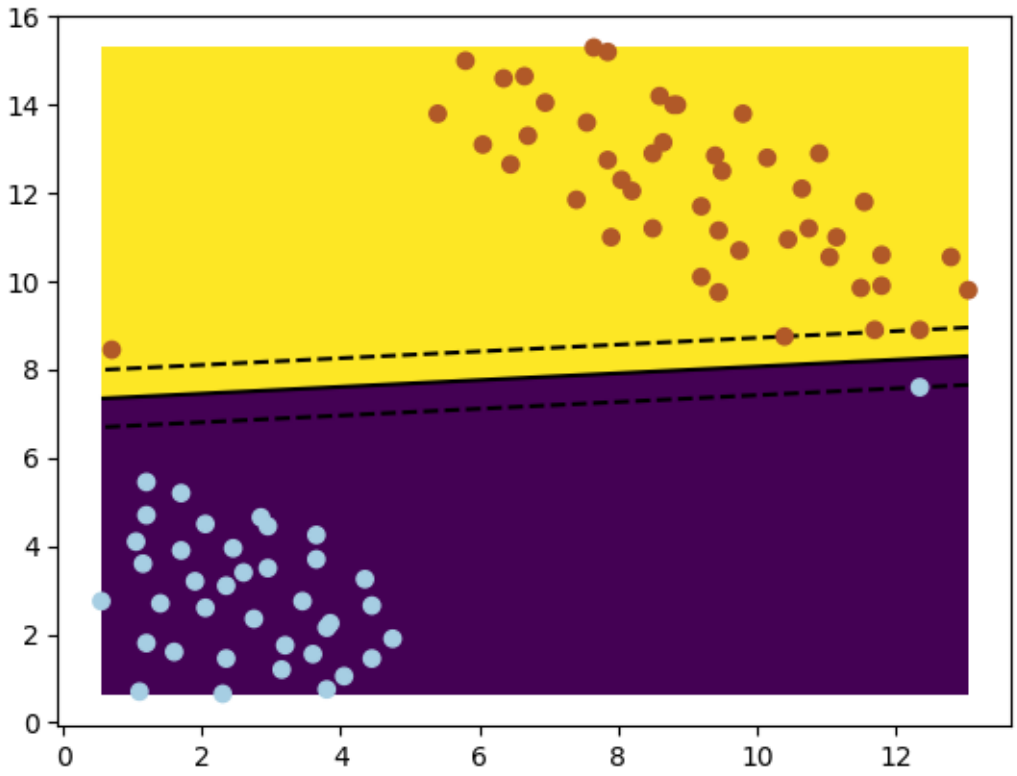
[22]:
```
libsvmScript('dataset1.csv', 'linear', 0.1)
```



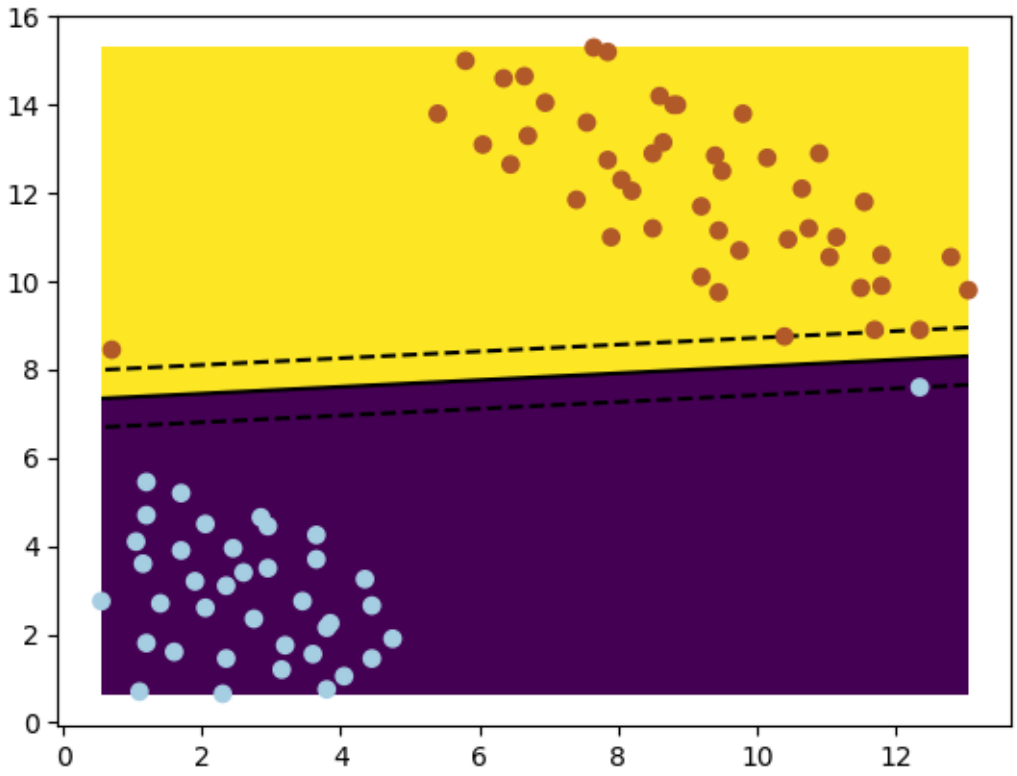
[23]:
```
libsvmScript('dataset1.csv', 'linear', 1)
```



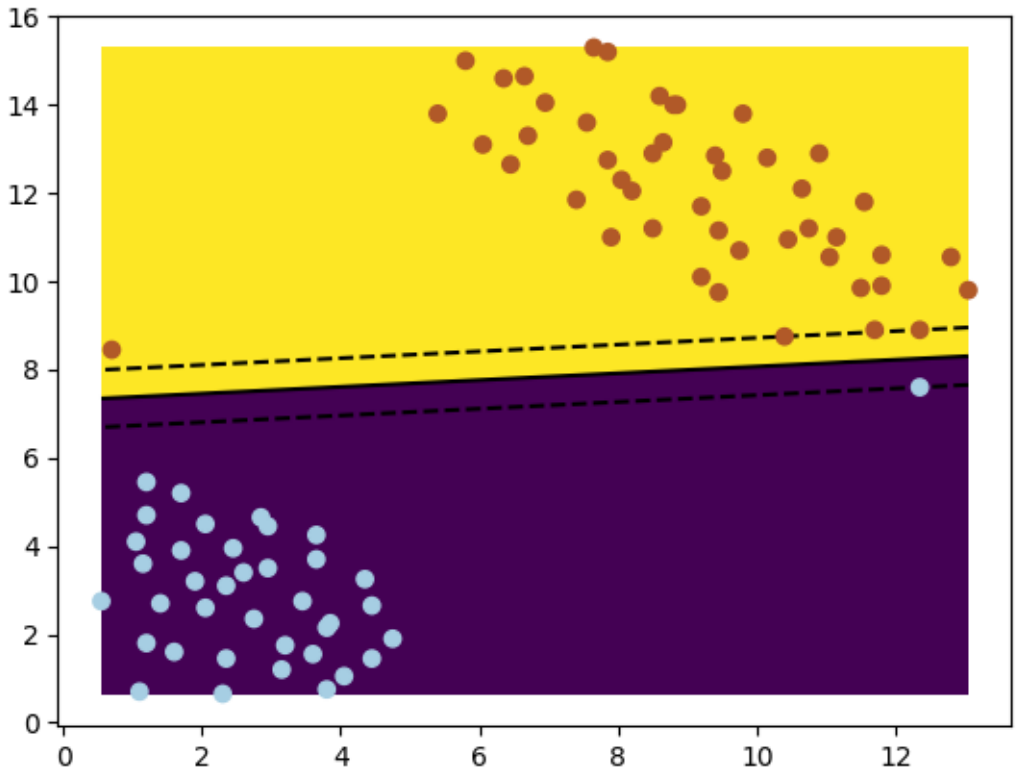
[24]:
```
libsvmScript('dataset1.csv', 'linear', 10)
```



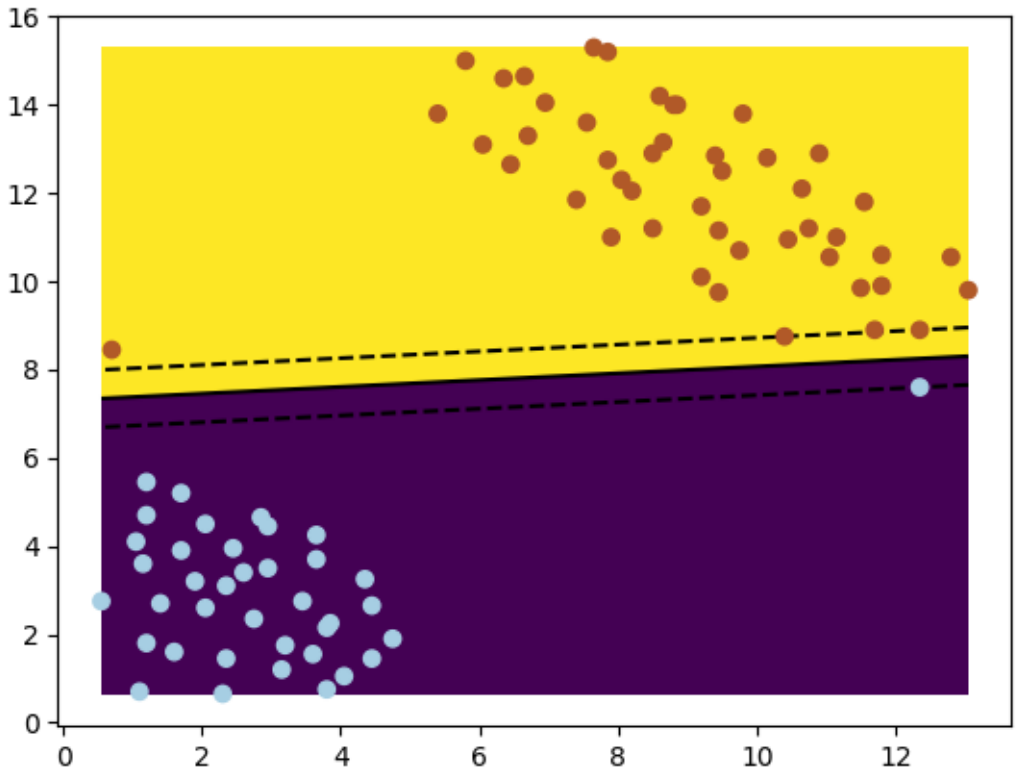
[25]:
```
libsvmScript('dataset1.csv', 'linear', 100)
```



[26]:
```
libsvmScript('dataset1.csv', 'linear', 1000)
```



[27]:
```
libsvmScript('dataset1.csv', 'linear', 10000)
```



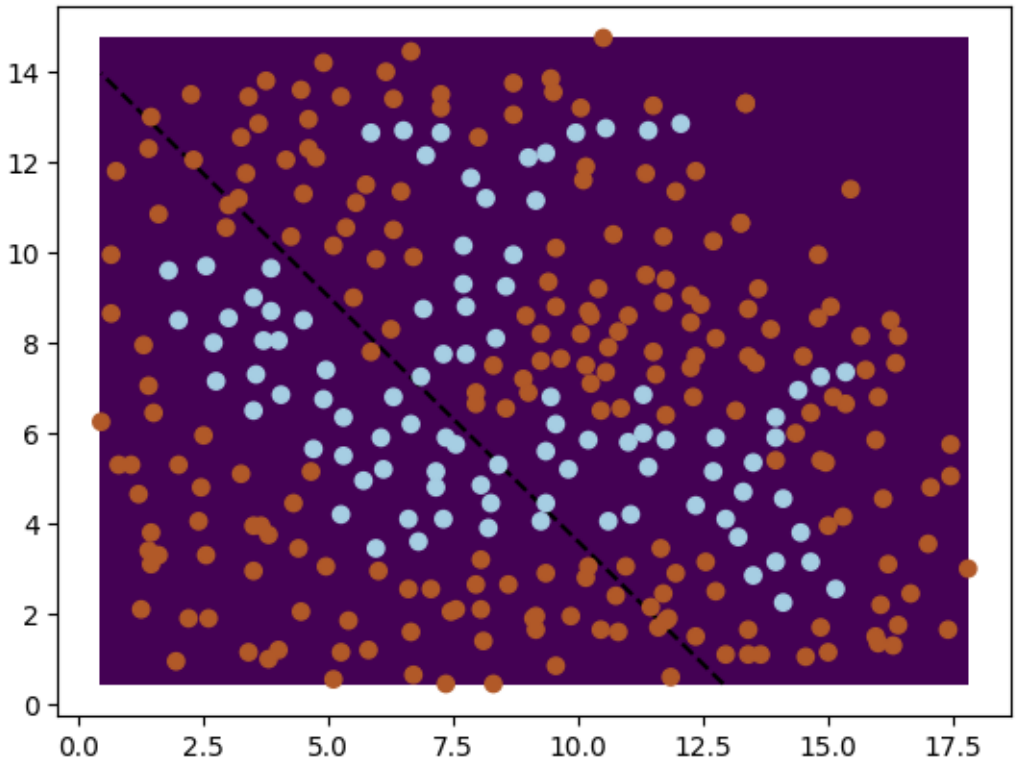
Con valores de C más bajos, el modelo tiende a tener un margen mucho mayor a cambio de un mayor número de errores.
Con valores intermedios, el modelo busca un equilibrio entre un amplio margen y la minimización de errores.
Con valores altos y muy altos, el modelo se centra en intentar clasificar todos los puntos correctamente.

En mi opinión, el valor más adecuado es 10, ya que está en equilibrio entre maximizar el margen y minimizar los errores.

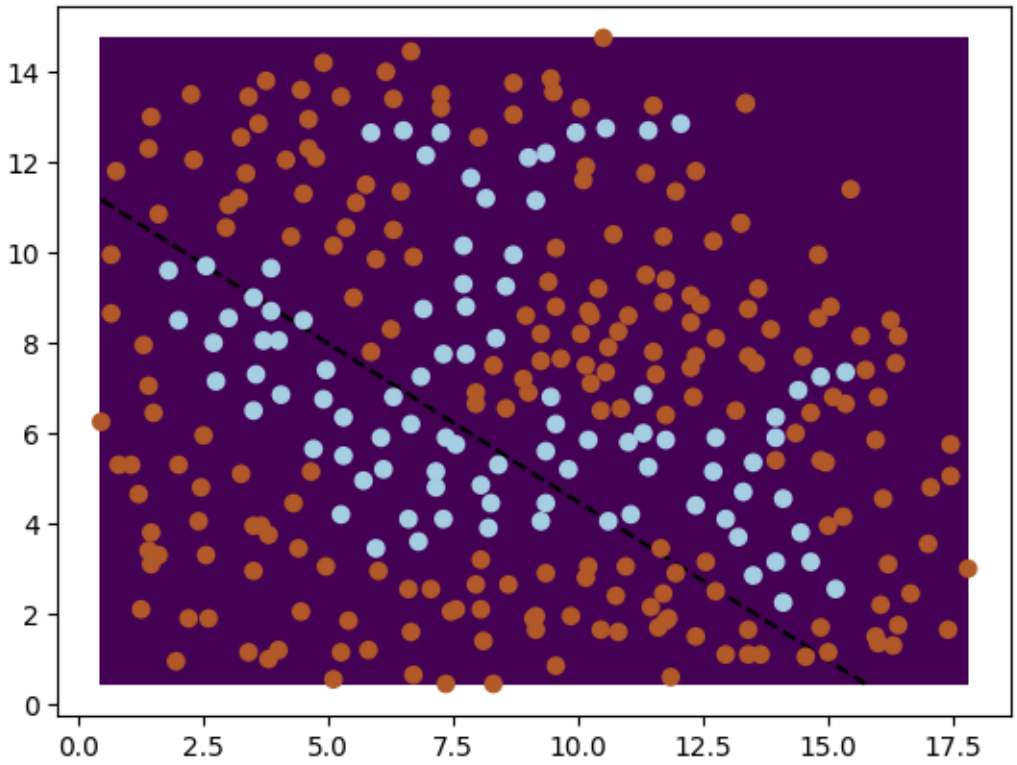
1.0.2 Dataset 2

[Pregunta 4] Prueba a lanzar una SVM lineal con los valores para C que se utilizaron en la pregunta anterior. ¿Consigues algun resultado satisfactorio en el sentido de que no haya errores en el conjunto de entrenamiento? ¿Por qué?

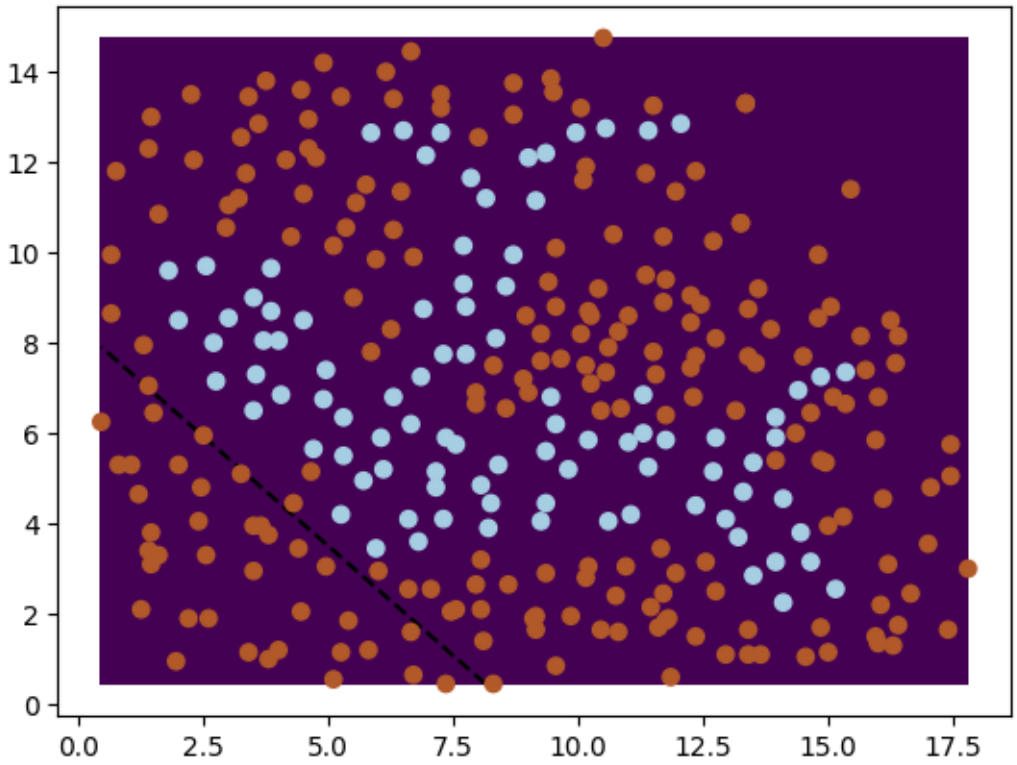
[32]:
```
libsvmScript('dataset2.csv', 'linear', 0.1)
```



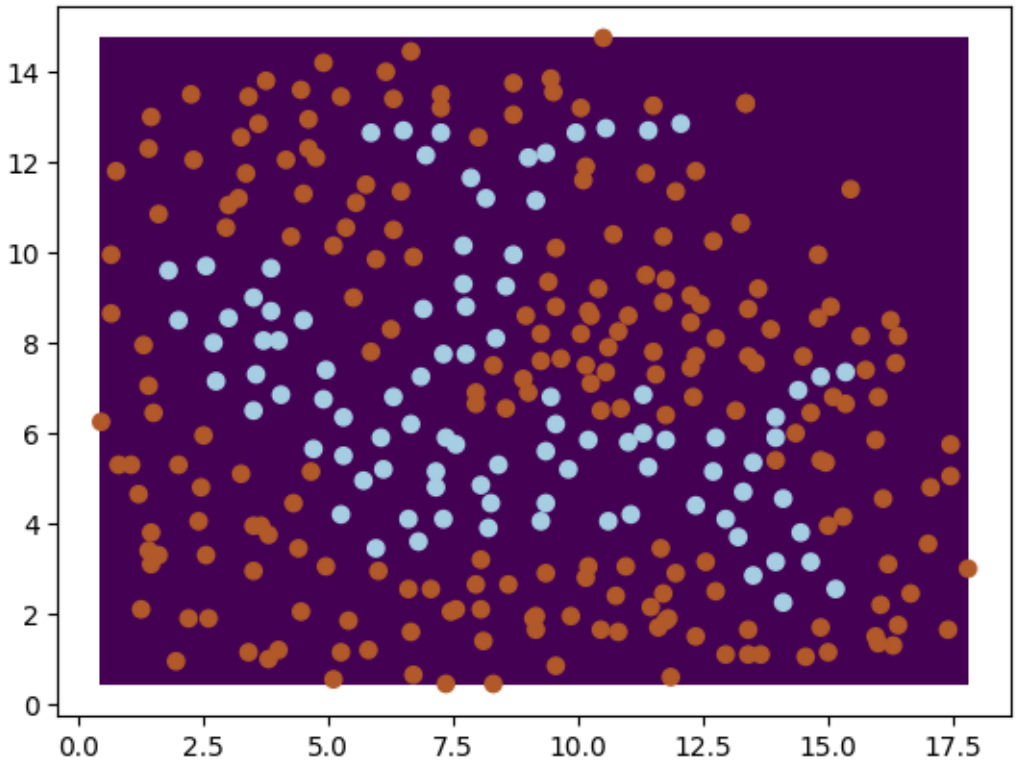
[31]:
```
libsvmScript('dataset2.csv', 'linear', 1)
```



[29]:
```
libsvmScript('dataset2.csv', 'linear', 10)
```



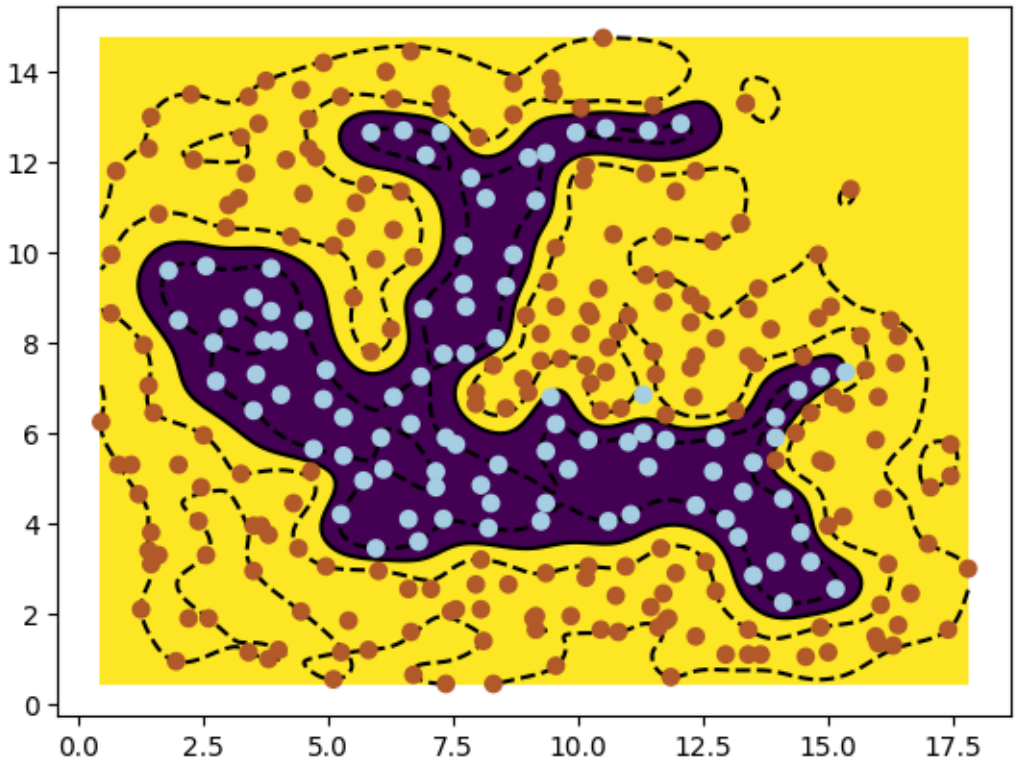
[30]:
```
libsvmScript('dataset2.csv', 'linear', 100)
```



No, no lo conseguimos. Esto se debe a que el conjunto de datos no es linealmente separable.

[Pregunta 5] Propón una configuración de SVM no lineal (utilizando el kernel tipo RBF o Gaussiano) que resuelva el problema. El resultado debería ser similar al de la Figura 3. Muévete en potencias de 2 utilizando un rango del tipo {2^-2, 2^-1, 2^0, 2^1, 2^2} ¿Que valores has considerado para C y para ?. Además, incluye un ejemplo de una configuración de parámetros que produzca sobre-entrenamiento y otra que produzca infra-entrenamiento.

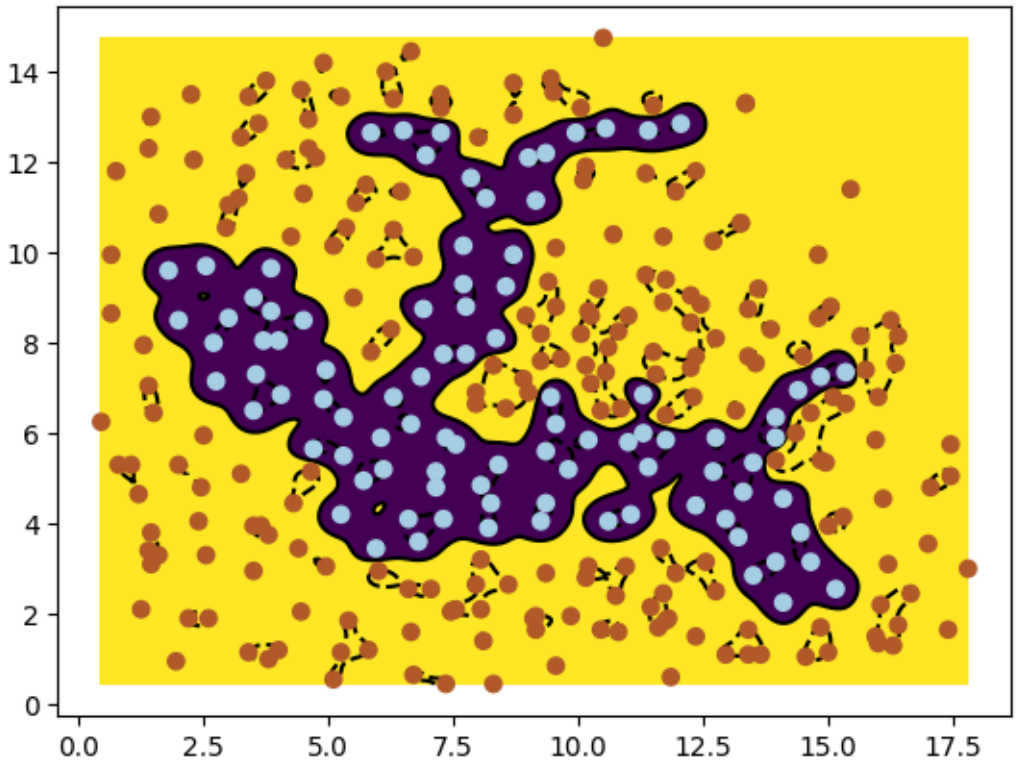
[52]:
```
libsvmScript('dataset2.csv', 'rbf', 4, 1)
```



Hemos considerado `C = 4` y `= 1`

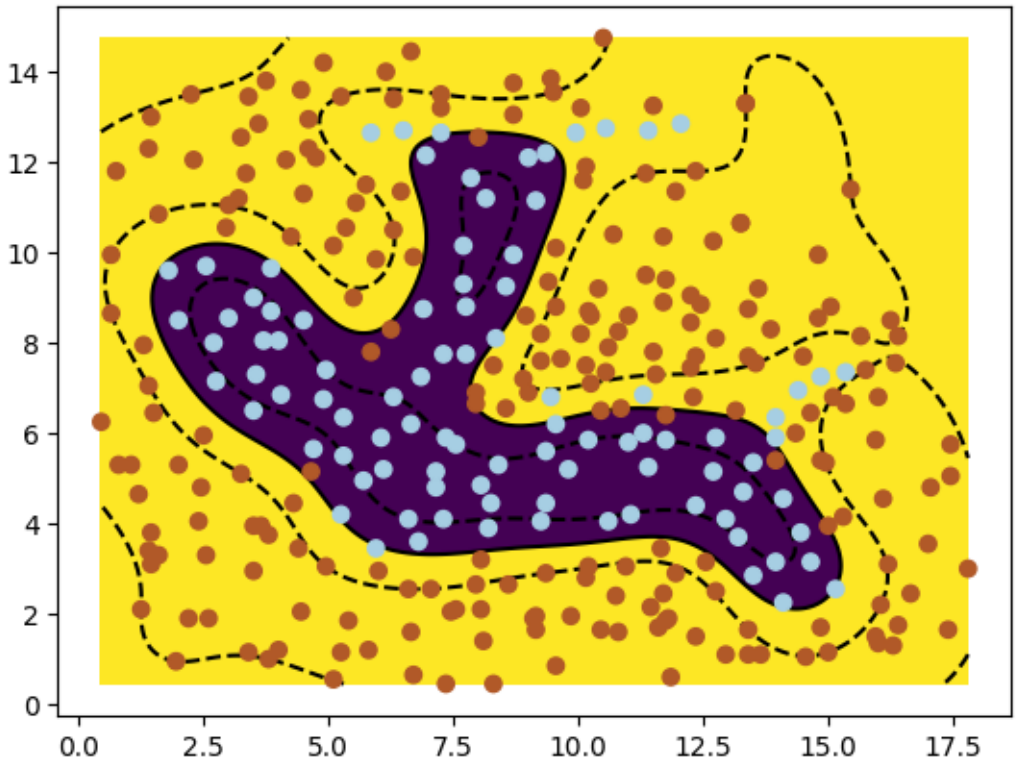
Ejemplo con sobreentrenamiento:

[58]:
```
libsvmScript('dataset2.csv', 'rbf', 1000, 5)
```



Ejemplo con infraentrenamiento:

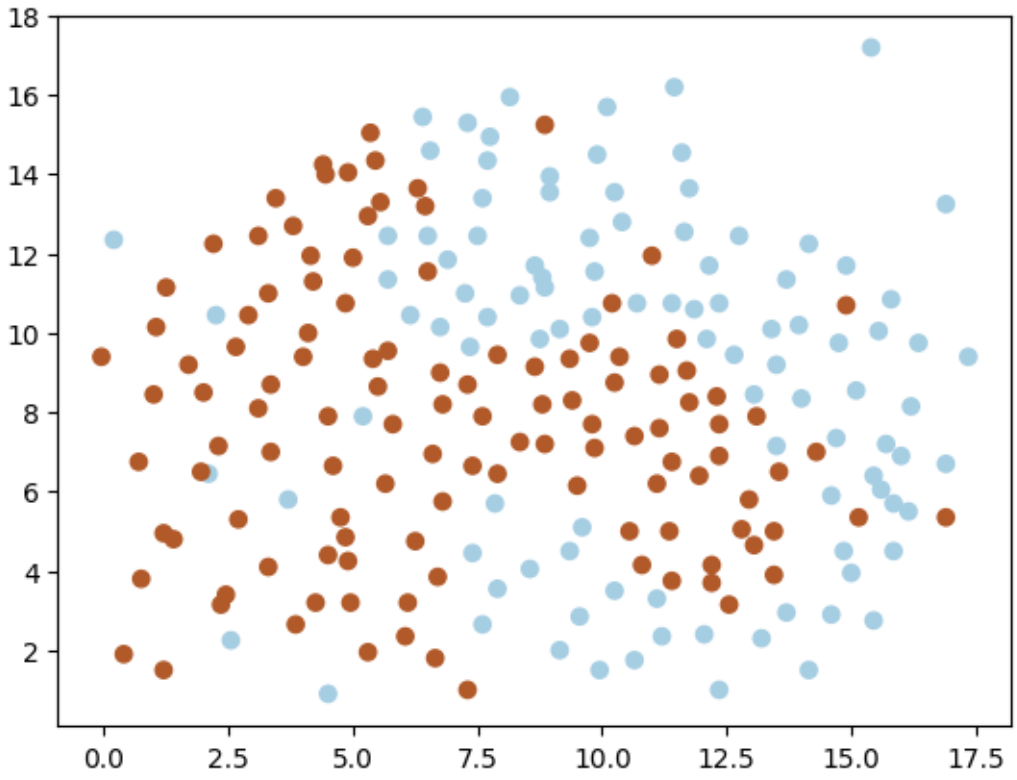
[60]:
```
libsvmScript('dataset2.csv', 'rbf', 1, 1/4)
```



1.0.3 Dataset 3

[Pregunta 6] En este caso, ¿es el dataset linealmente separable?. A primera vista, ¿detectas puntos que presumiblemente sean outliers?, ¿por que?

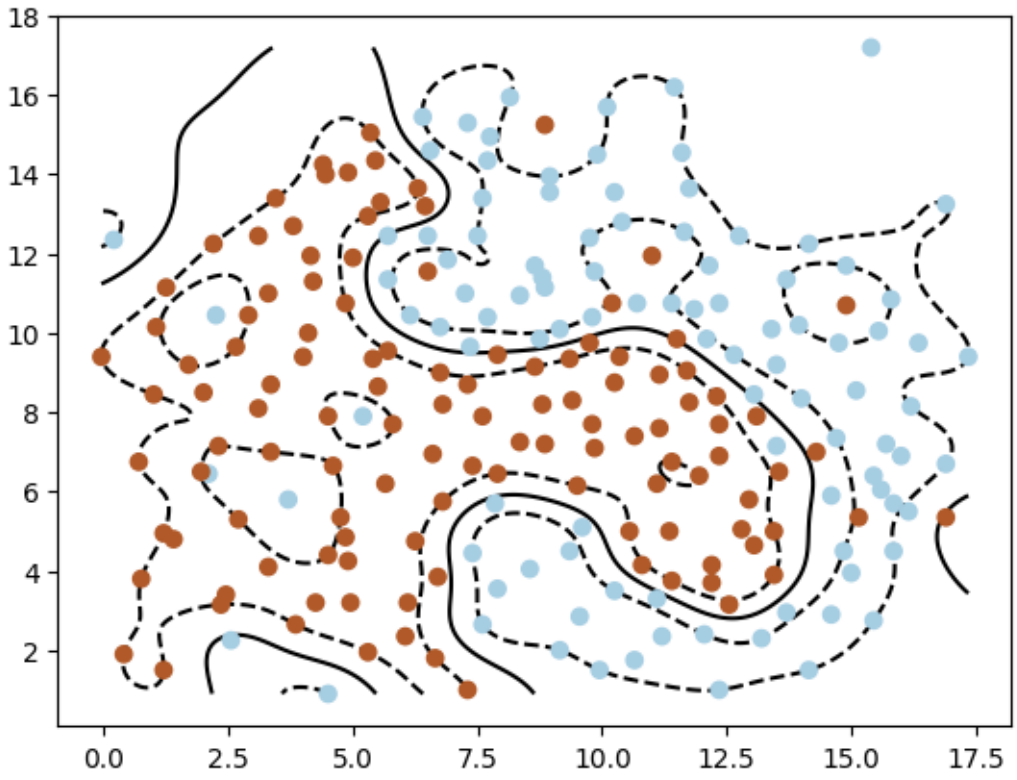
[62]:
```
showDataset('dataset3.csv')
```



No, no es linealmente separable.
Sí, ya que hay ciertos puntos muy separados de los grupos correspondientes a sus etiquetas.

[Pregunta 7] Lanza una SVM para clasificar los datos, con objeto de obtener un resultado lo mas parecido al de la Figura 5. Muevete en potencias de 2. Establece el valor de los parametros optimos. Ademas, incluye un ejemplo de una configuracion de parametros que produzca sobre-entrenamiento y otra que produzca infra-entrenamiento.

[91]:
```
libsvmScriptNoColor('dataset3.csv', 'rbf', 2, 1/2)
```



Los parámetros óptimos son `C=2` y `gamma = 1/2`

[Pregunta 8] Vamos a reproducir este proceso en Python. Divide el dataset sintetico dataset3.csv en dos subconjuntos aleatorios de forma estratificada, con un 75 % de patrones en train y un 25 % de patrones en test. Realiza el proceso de entrenamiento completo (estandarizacion, entrenamiento y prediccion), volviendo a optimizar los valores de C y que obtuviste en la ultima pregunta. Comprueba el porcentaje de buena clasificacion que se obtiene para el conjunto de test. Repite el proceso mas de una vez para comprobar que los resultados dependen mucho de la semilla utilizada para hacer la particion.

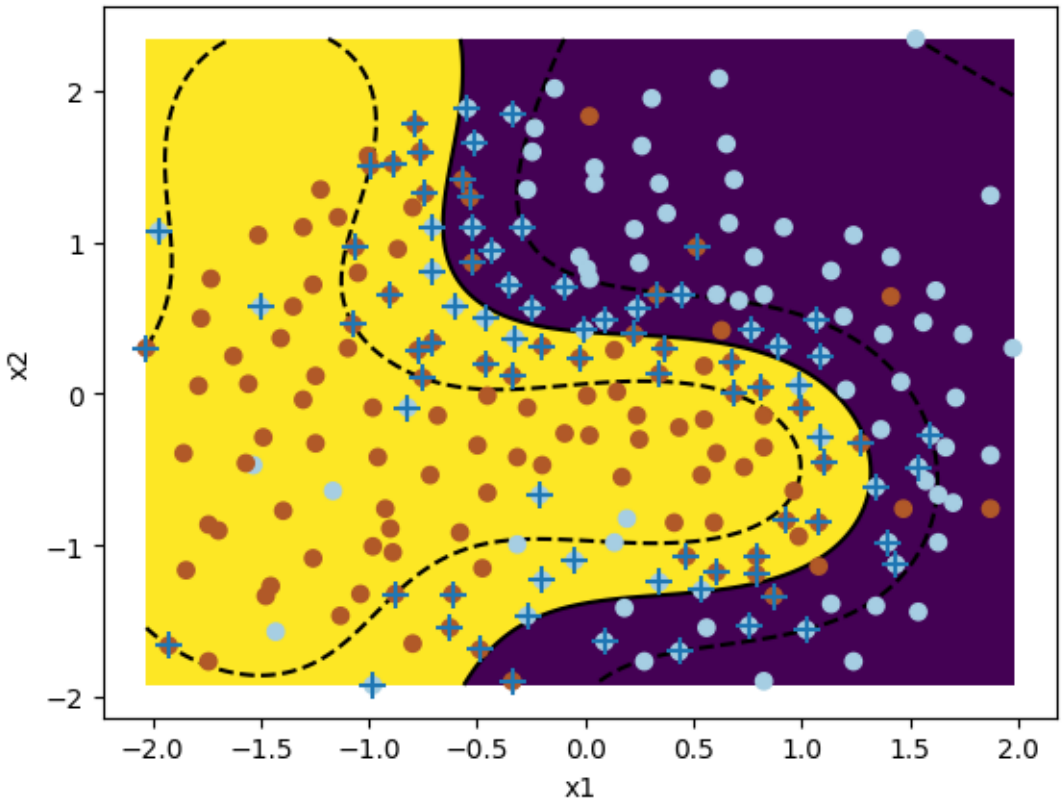
[94]:
```
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
```

[120]:
```
def ej8():
  scaler = preprocessing.StandardScaler()
  # Load the dataset
  data = pd.read_csv("dataset3.csv",header=None)
  X = data.iloc[:,:-1].values
  y = data.iloc[:,-1].values
```

```
    X_train = scaler.fit_transform(X,y)
    # Entrenamiento estratificado con 75% de entrenamiento y 25% de test.
    x_train, x_test, y_train, y_test = train_test_split(X_train, y, stratify=y,␣
    ↪test_size=0.25)
    # Realizamos la predicción de test, aplicando los
    # parámetros C y gamma obtenidos en el ejercicio anterior
    svm_model = svm.SVC(kernel='rbf',C=2, gamma=1/2)
    svm_model.fit(x_train, y_train)
    print(svm_model.score(x_test,y_test))
    # Plot the points
    plt.figure(1)
    plt.clf()
    plt.scatter(X_train[:, 0], X_train[:, 1], c=y, zorder=10, cmap=plt.cm.Paired)
    # Plot the support vectors class regions, the separating hyperplane and the␣
    ↪margins
    plt.axis('tight')
    # |->Plot support vectors
    plt.scatter(svm_model.support_vectors_[:,0], svm_model.support_vectors_[:,1],
              marker='+',  s=100, zorder=10, cmap=plt.cm.Paired)
    # |-> Extract the limits
    x_min = X_train[:, 0].min()
    x_max = X_train[:, 0].max()
    y_min = X_train[:, 1].min()
    y_max = X_train[:, 1].max()
    # |-> Create a grid with all the points and then obtain the SVM
    #    score for all the points
    XX, YY = np.mgrid[x_min:x_max:500j, y_min:y_max:500j]
    Z = svm_model.decision_function(np.c_[XX.ravel(), YY.ravel()])
    # |-> Plot the results in a countour
    Z = Z.reshape(XX.shape)
    plt.pcolormesh(XX, YY, Z > 0)
    plt.contour(XX, YY, Z, colors=['k', 'k', 'k'],
              linestyles=['--', '-', '--'], levels=[-1, 0, 1])

    plt.xlabel('x1')
    plt.ylabel('x2')

    plt.show()
```

[121]:
```
ej8()
```

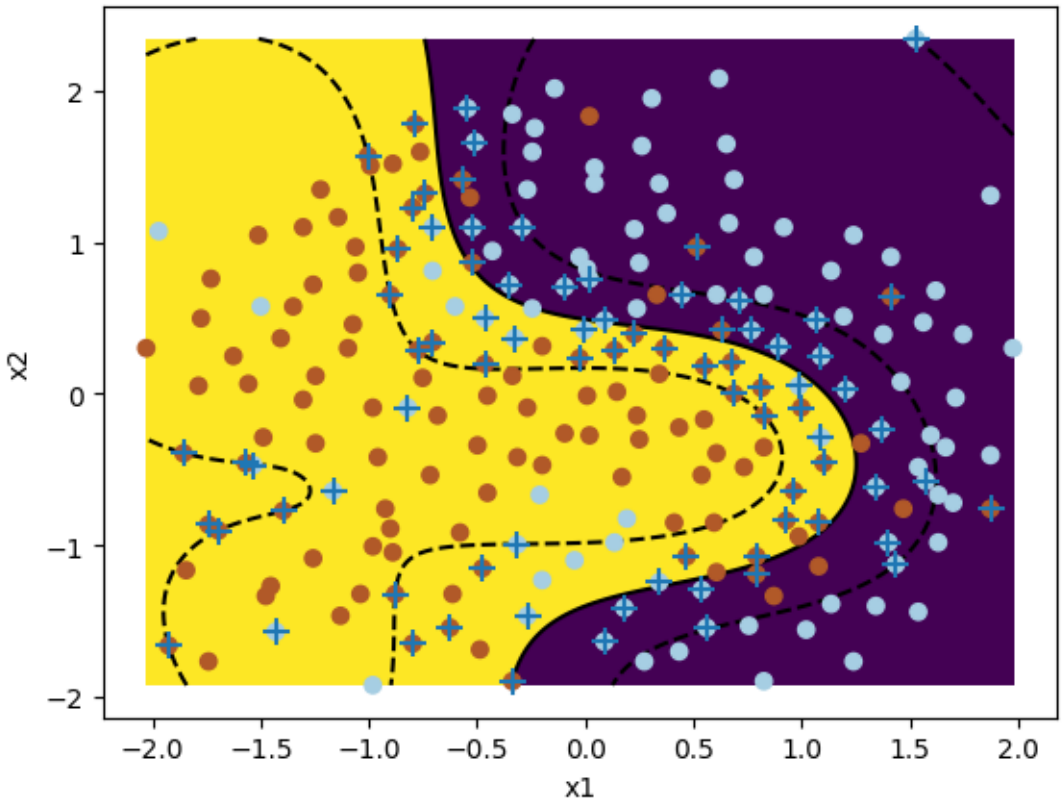
```
0.7857142857142857

C:\Users\Corne\AppData\Local\Temp\ipykernel_8388\2227253269.py:22: UserWarning:
No data for colormapping provided via 'c'. Parameters 'cmap' will be ignored
  plt.scatter(svm_model.support_vectors_[:,0], svm_model.support_vectors_[:,1],
```



[97]:
```
ej8()
```

```
0.6785714285714286
```

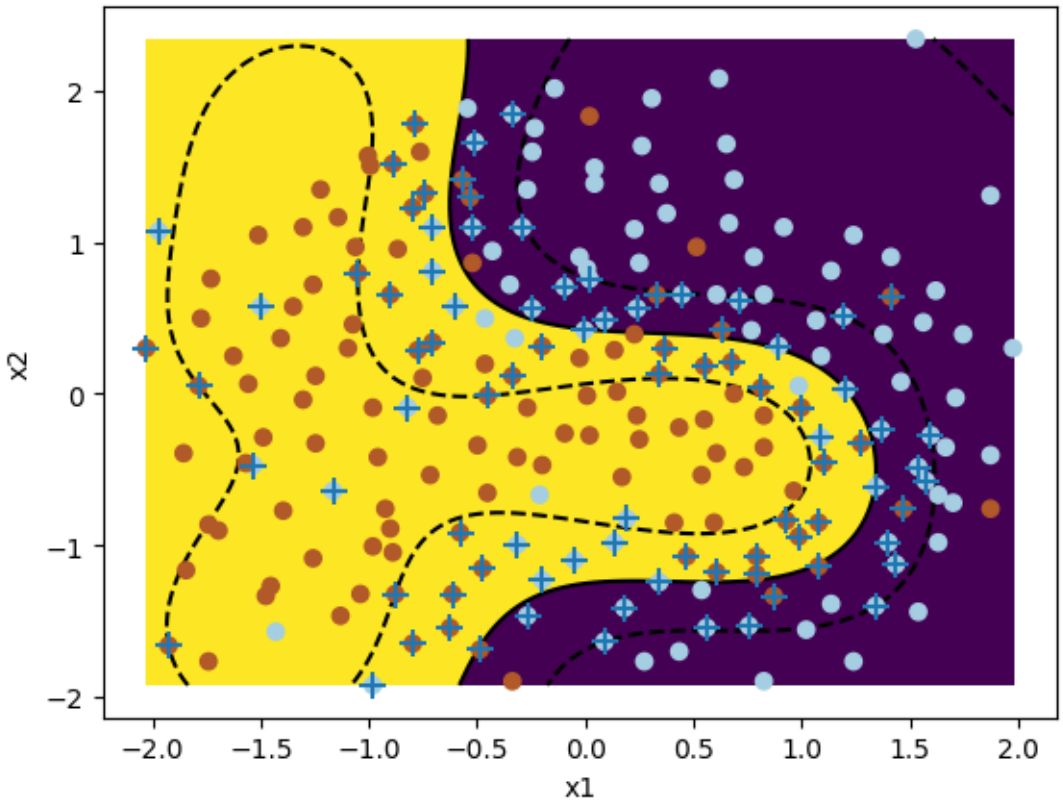
```
C:\Users\Corne\AppData\Local\Temp\ipykernel_8388\1784827145.py:22: UserWarning:
No data for colormapping provided via 'c'. Parameters 'cmap' will be ignored
  plt.scatter(svm_model.support_vectors_[:,0], svm_model.support_vectors_[:,1],
```



[98]:
```
ej8()
```

```
0.8214285714285714
```

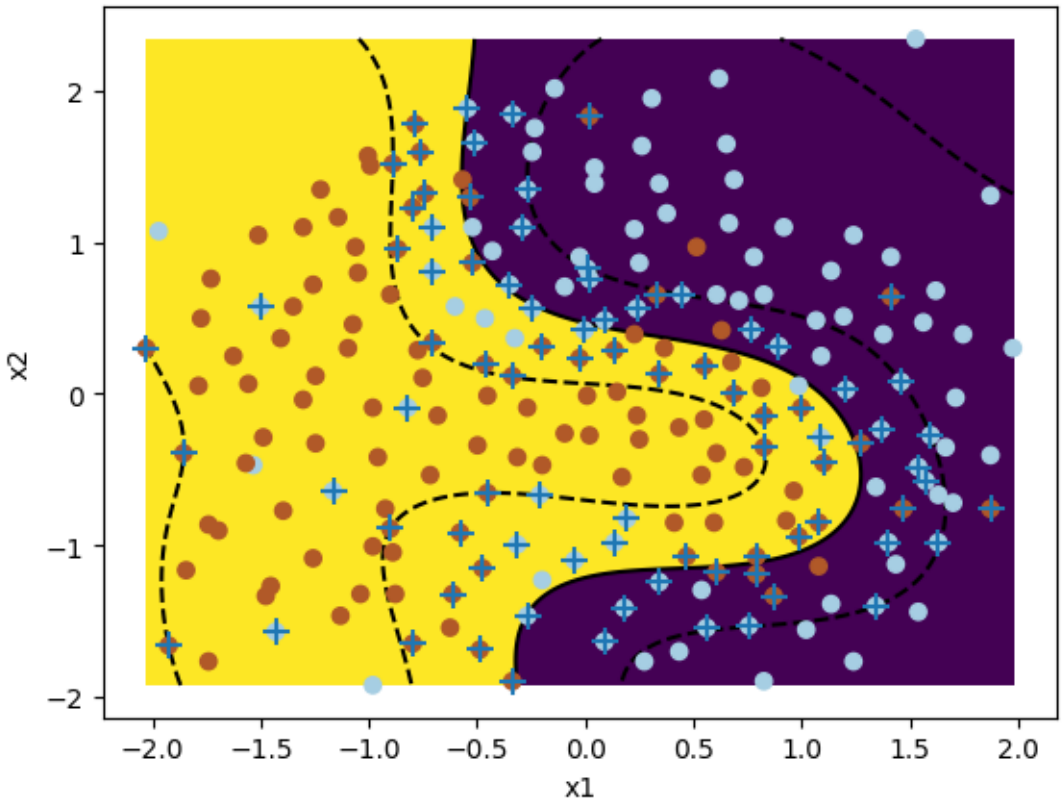
```
C:\Users\Corne\AppData\Local\Temp\ipykernel_8388\1784827145.py:22: UserWarning:
No data for colormapping provided via 'c'. Parameters 'cmap' will be ignored
  plt.scatter(svm_model.support_vectors_[:,0], svm_model.support_vectors_[:,1],
```



[126]:
```
ej8()
```

```
0.7857142857142857
```

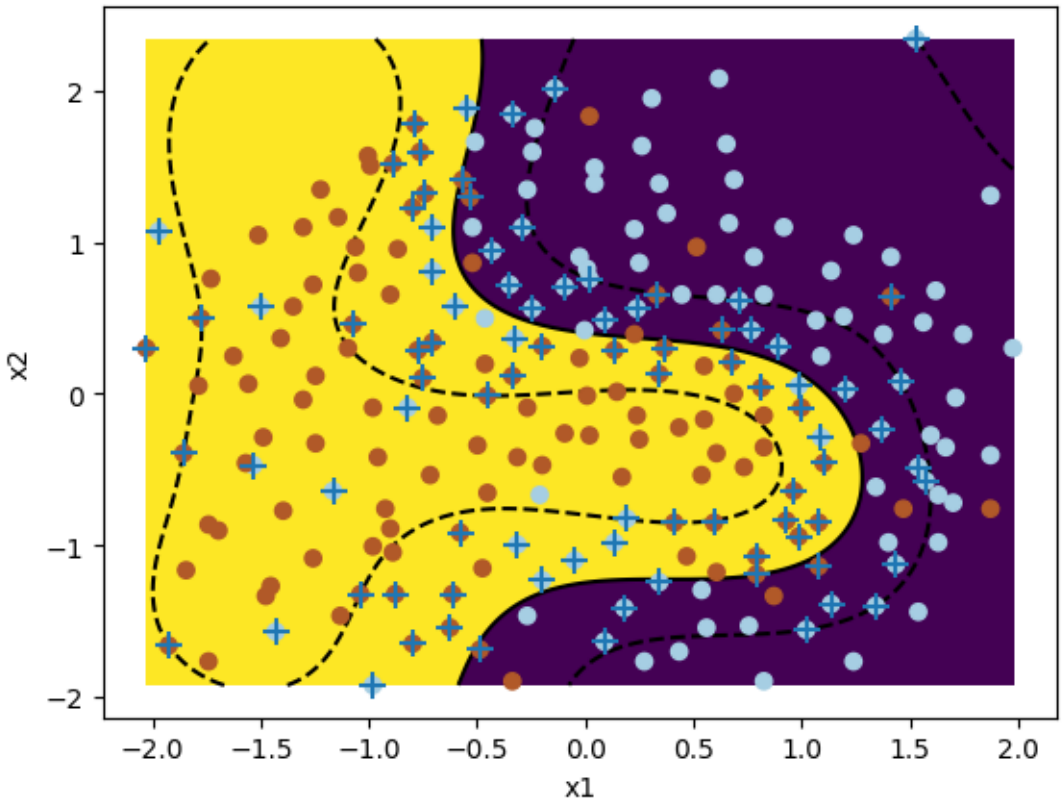
```
C:\Users\Corne\AppData\Local\Temp\ipykernel_8388\2227253269.py:22: UserWarning:
No data for colormapping provided via 'c'. Parameters 'cmap' will be ignored
  plt.scatter(svm_model.support_vectors_[:,0], svm_model.support_vectors_[:,1],
```



[125]:
```
ej8()
```

```
0.8035714285714286
```

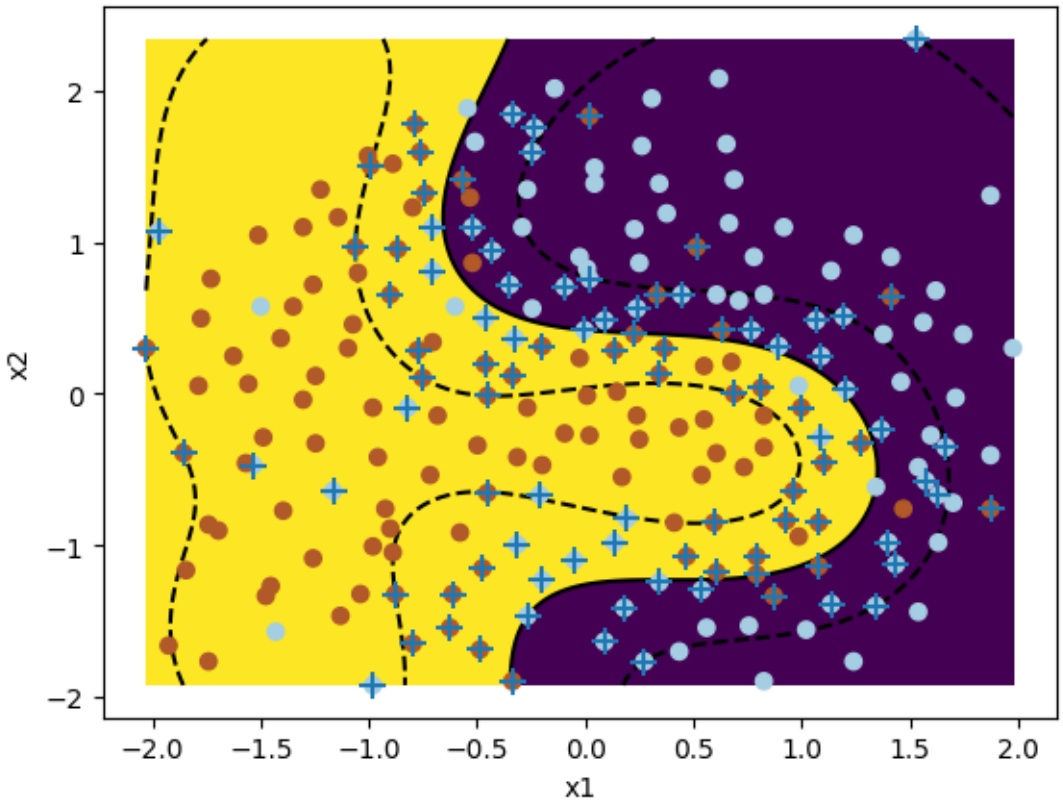
```
C:\Users\Corne\AppData\Local\Temp\ipykernel_8388\2227253269.py:22: UserWarning:
No data for colormapping provided via 'c'. Parameters 'cmap' will be ignored
  plt.scatter(svm_model.support_vectors_[:,0], svm_model.support_vectors_[:,1],
```



[124]:
```
ej8()
```

```
0.8571428571428571
```

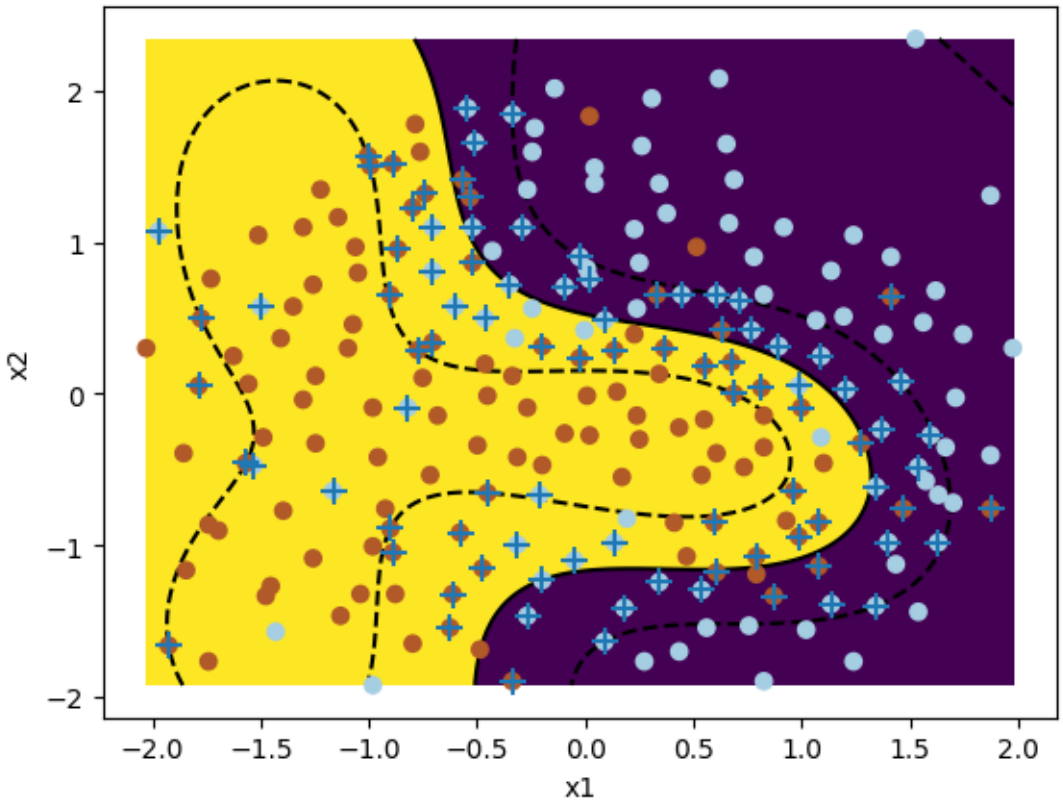
```
C:\Users\Corne\AppData\Local\Temp\ipykernel_8388\2227253269.py:22: UserWarning:
No data for colormapping provided via 'c'. Parameters 'cmap' will be ignored
  plt.scatter(svm_model.support_vectors_[:,0], svm_model.support_vectors_[:,1],
```



[123]:
```
ej8()
```

```
0.8214285714285714
```

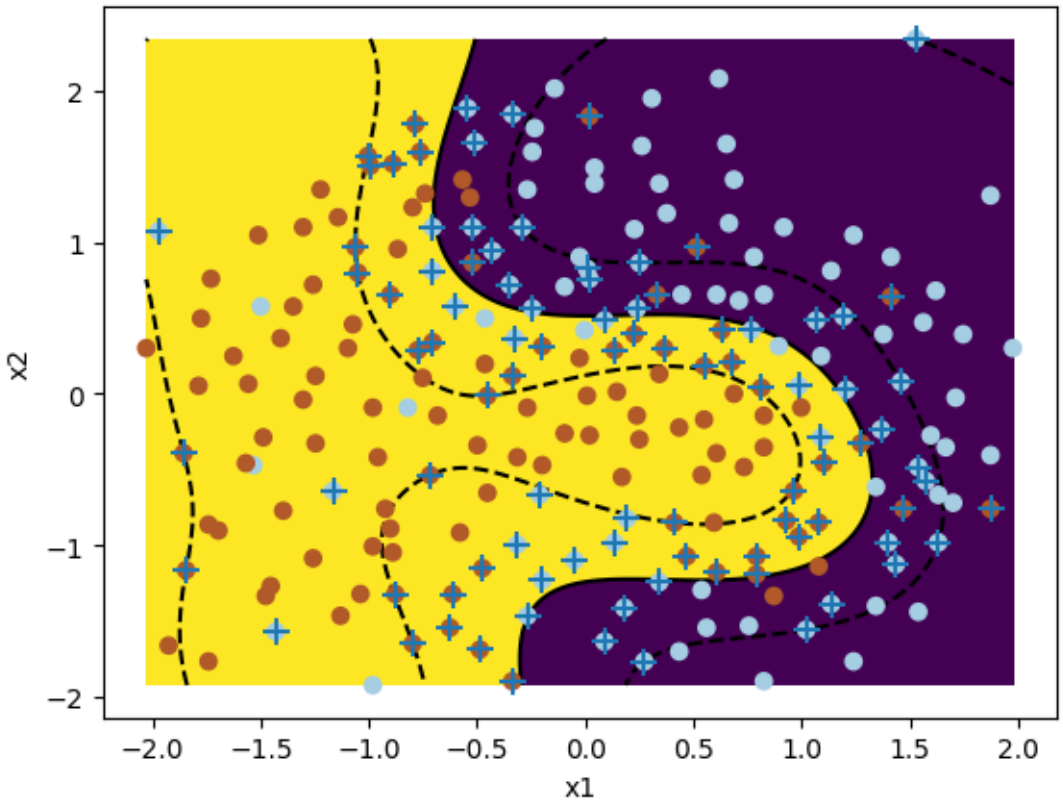
```
C:\Users\Corne\AppData\Local\Temp\ipykernel_8388\2227253269.py:22: UserWarning:
No data for colormapping provided via 'c'. Parameters 'cmap' will be ignored
  plt.scatter(svm_model.support_vectors_[:,0], svm_model.support_vectors_[:,1],
```



[122]:
```
ej8()
```

```
0.8035714285714286
```

```
C:\Users\Corne\AppData\Local\Temp\ipykernel_8388\2227253269.py:22: UserWarning:
No data for colormapping provided via 'c'. Parameters 'cmap' will be ignored
  plt.scatter(svm_model.support_vectors_[:,0], svm_model.support_vectors_[:,1],
```



Tras ejecutarlo 8 veces, vemos una notable variacion debido a las diferencias en las semillas.

[Pregunta 9] Amplia el codigo anterior para realizar el entrenamiento de la pregunta 8 sin necesidad de especificar los valores de C y . Compara los valores optimos obtenidos para ambos parametros con los que obtuviste a mano. Extiende el rango de valores a explorar, si es que lo consideras necesario.

[138]:
```
def ej9():
  scaler = preprocessing.StandardScaler()
  # Load the dataset
  data = pd.read_csv("dataset3.csv",header=None)
  X = data.iloc[:,:-1].values
  y = data.iloc[:,-1].values
  X_train = scaler.fit_transform(X,y)
  Cs = np.logspace(-2, 4, num=7, base=10)
  Gs = np.logspace(-2, 2, num=5, base=2)
  # Entrenamiento estratificado con 75% de entrenamiento y 25% de test.
  x_train, x_test, y_train, y_test = train_test_split(X_train, y, stratify=y,␣
↪test_size=0.25)
  # Realizamos la predicción de test, aplicando los parámetros C y gamma␣
↪obtenidos anteriormente
  svm_model = svm.SVC(kernel='rbf',C=2, gamma=1/2)
```

```
  svm_model.fit(x_train, y_train)
  paramOpt = GridSearchCV(estimator=svm_model, param_grid=dict(C=Cs,gamma=Gs),␣
↪n_jobs=-1, cv=5)
  paramOpt.fit(x_train, y_train)
  print(paramOpt.score(x_test, y_test))
```

[147]:
```
for i in range (1,6):
    ej9()
```

```
0.8392857142857143
0.875
0.8214285714285714
0.9107142857142857
0.8928571428571429
```

Por lo general, los resultados de utilizar validacion cruzada son mejores que los resultados de haberlos ajustado a mano.

[Pregunta 10] ¿Que inconvenientes observas en ajustar el valor de los parametros "a mano", viendo el porcentaje de buena clasificacion en el conjunto de test (lo que se hizo en la pregunta 8)? Los inconvenientes son que es más laborioso que ajustarlo a mano, y que los resultados son, por lo general, peores.

[Pregunta 11] Para estar seguros de que has entendido como se realiza la busqueda de parametros, implementa de forma manual (sin usar GridSearchCV) la validacion cruzada anidada tipo K-fold expuesta en esta seccion. Te puede ser util el uso de listas por compresion y la clase StratifiedKFold. Compara los resultados con los que obtienes usando GridSearchCV

[148]:
```
def ej11():
  Cs = np.logspace(-2, 4, num=7, base=10)
  Gs = np.logspace(-2, 2, num=5, base=2)
  skf = StratifiedKFold(n_splits=5)
  results=np.zeros((len(Cs)*len(Gs),7))
  data = pd.read_csv("dataset3.csv",header=None)
  X = data.iloc[:,:-1].values
  y = data.iloc[:,-1].values
  i=2
  for train_index, test_index in skf.split(X, y):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    j=0

    for c in Cs:
        for g in Gs:
            train = svm.SVC(kernel='rbf', C=c, gamma=g).fit(X_train, y_train)

            if i == 2:
```

```
                results[j,0]=c
                results[j,1]=g

            results[j,i]=train.score(X_test, y_test)
            j = j+1
    i= i+1
  maxJ=0
  maxMean=np.mean(results[0,2:7])
  # Buscamos la mejor combinacion de parametros
  for x in range(1, (len(Cs)*len(Gs))):
    mean = np.mean(results[x,2:7])
    if mean > maxMean:
        maxMean = mean
        maxJ = x
  # Utilizamos los parámetros encontrados
  optimo = svm.SVC(kernel='rbf', C=results[maxJ,0], gamma=results[maxJ,1])
  x_train, x_test, y_train, y_test = train_test_split(X, y, stratify=y,␣
 ↪test_size=0.25)
  optimo.fit(x_train, y_train)
  print(optimo.score(x_test, y_test))
```

[150]:
```
for i in range (1,6):
    ej11()
```

```
0.875
0.8571428571428571
0.8571428571428571
0.8571428571428571
0.9107142857142857
```

Nos ha ofrecido valores similares que utilizando GridSearchCV

1.0.4 Bases de datos reales

[Pregunta 12] Utiliza el script que desarrollaste en la pregunta 9 para entrenar esta base de datos, pero sin dividir el dataset ni estandarizarlo. Utiliza como rangos C, {0.001, 0.01, ..., 1000}. Observa el valor de accuracy obtenido para el conjunto de generalizacion. Al finalizar, toma nota de los valores optimos obtenidos para los parametros

[158]:
```
def ej12():
    scaler = preprocessing.StandardScaler()
    # Load the dataset
    data_train = pd.read_csv('train_compas.csv',header=None)
    data_test = pd.read_csv('test_compas.csv',header=None)
    x_train = data_train.iloc[:,:-1].values
    y_train = data_train.iloc[:,-1].values
    x_test = data_test.iloc[:,:-1].values
    y_test = data_test.iloc[:,-1].values
```

```
    Cs = np.logspace(-3, 3, num=7, base=10)
    Gs = np.logspace(-3, 3, num=7, base=2)

    svm_model = svm.SVC(kernel='rbf',C=2, gamma=1/2)
    svm_model.fit(x_train, y_train)

    paramOpt = GridSearchCV(estimator=svm_model,␣
↪param_grid=dict(C=Cs,gamma=Gs), n_jobs=-1, cv=3)
    paramOpt.fit(x_train, y_train)

    print(paramOpt.score(x_test, y_test))
```

[ ]:
```
def ej9():
  scaler = preprocessing.StandardScaler()
  # Load the dataset
  data = pd.read_csv("dataset3.csv",header=None)
  X = data.iloc[:,:-1].values
  y = data.iloc[:,-1].values
  X_train = scaler.fit_transform(X,y)
  Cs = np.logspace(-2, 4, num=7, base=10)
  Gs = np.logspace(-2, 2, num=5, base=2)
  # Entrenamiento estratificado con 75% de entrenamiento y 25% de test.
  x_train, x_test, y_train, y_test = train_test_split(X_train, y, stratify=y,␣
↪test_size=0.25)
  # Realizamos la predicción de test, aplicando los parámetros C y gamma␣
↪obtenidos anteriormente
  svm_model = svm.SVC(kernel='rbf',C=2, gamma=1/2)
  svm_model.fit(x_train, y_train)
  paramOpt = GridSearchCV(estimator=svm_model, param_grid=dict(C=Cs,gamma=Gs),␣
↪n_jobs=-1, cv=5)
  paramOpt.fit(x_train, y_train)
  print(paramOpt.score(x_test, y_test))
```

[159]:
```
ej12()
```

```
0.6590909090909091
```

Este modelo da un resultado de 65,9%, lo cual es un modelo decente, pero tampoco demasiado bueno.

[Pregunta 13] Localiza donde se especifica el valor de K para la validacion cruzada interna y el rango de valores que se han utilizado para los parametros C y . ¿Como podrıas reducir el tiempo computacional necesario para realizar el experimento? Prueba a establecer K = 3, K = 5 y K = 10 y compara, utilizando una tabla, los tiempos computacionales obtenidos y los resultados de CCR en test.

[160]:
```
import time
```

[161]:
```
def ej13(k):
    scaler = preprocessing.StandardScaler()
    # Load the dataset
    data_train = pd.read_csv('train_compas.csv',header=None)
    data_test = pd.read_csv('test_compas.csv',header=None)
    x_train = data_train.iloc[:,:-1].values
    y_train = data_train.iloc[:,-1].values
    x_test = data_test.iloc[:,:-1].values
    y_test = data_test.iloc[:,-1].values

    Cs = np.logspace(-3, 3, num=7, base=10)
    Gs = np.logspace(-3, 3, num=7, base=2)

    svm_model = svm.SVC(kernel='rbf',C=2, gamma=1/2)
    svm_model.fit(x_train, y_train)

    paramOpt = GridSearchCV(estimator=svm_model,␣
 ↪param_grid=dict(C=Cs,gamma=Gs), n_jobs=-1, cv=k)
    paramOpt.fit(x_train, y_train)

    return paramOpt.score(x_test, y_test)
```

[163]:
```
k = [3,5,10]
print('K : CCR : Tiempo')
for i in k:
  start = time.time()
  CCR=ej13(i)
  end = time.time()
  totalTime = end - start
  print(i,' : ',CCR,' : ',totalTime)
```

```
K : CCR : Tiempo
3  :  0.6590909090909091  :  118.04176664352417
5  :  0.6590909090909091  :  302.90902400016785
10  :  0.6590909090909091  :  755.0333852767944
```

Los resultados son los mismos, pero los tiempos computacionales son mucho mayores

1.0.5 Clasificación de Spam

[Pregunta 14] Debes entrenar un modelo lineal de SVM con valores C = 10^−2, C = 10^−1, C = 10^0 y C = 10^1. Para ello utiliza un script similar al que usaste para la pregunta 9. Compara los resultados y establece la mejor configuracion.

[ ]:
```
def ej14(c_value):
  # Load the dataset
  data_train = pd.read_csv('train_spam.csv',header=None)
  data_test = pd.read_csv('test_spam.csv',header=None)
  x_train = data_train.iloc[:,:-1].values
```

```
  y_train = data_train.iloc[:,-1].values
  x_test = data_test.iloc[:,:-1].values
  y_test = data_test.iloc[:,-1].values
  Cs = np.logspace(-2, 4, num=7, base=10)
  Gs = np.logspace(-2, 2, num=5, base=2)
  # Realizamos la predicción de test, aplicando los parámetros C y gamma␣
 ↪obtenidos anteriormente
  svm_model = svm.SVC(kernel='linear',C=c_value)
  svm_model.fit(x_train, y_train)
  paramOpt = GridSearchCV(estimator=svm_model, param_grid=dict(C=Cs,gamma=Gs),␣
 ↪n_jobs=-1, cv=5)
  paramOpt.fit(x_train, y_train)
  print(paramOpt.score(x_test, y_test))
```

[ ]:
```
c = [0.01, 0.1, 1, 10]

for i in c:
    ej14(i)
```

[Pregunta 15] Para la mejor configuracion, construye la matriz de confusion y establece cuales son los correos en los que la SVM se equivoca. Consulta las variables de entrada para los correos que no se clasifican correctamente y razona el motivo. Ten en cuenta que para cada patron, cuando Xi es igual a 1 quiere decir que la palabra i-esima del vocabulario aparece, al menos una vez, en el correo.

[ ]:
```
from sklearn.metrics import confusion_matrix
```

[ ]:
```
# Realiza la predicción en el conjunto de test
y_pred = svm_model.predict(x_test)

# Calcula la matriz de confusión
cm = confusion_matrix(y_test, y_pred)

# Imprime la matriz de confusión
print(cm)
```

[Pregunta 16] Entrena una SVM no lineal y compara los resultados.

[ ]:
```
def ej16():
  # Load the dataset
  data_train = pd.read_csv('train_spam.csv',header=None)
  data_test = pd.read_csv('test_spam.csv',header=None)
  x_train = data_train.iloc[:,:-1].values
  y_train = data_train.iloc[:,-1].values
  x_test = data_test.iloc[:,:-1].values
  y_test = data_test.iloc[:,-1].values
  Cs = np.logspace(-2, 4, num=7, base=10)
```

```
Gs = np.logspace(-2, 2, num=5, base=2)
# Entrenamiento estratificado con 75% de entrenamiento y 25% de test.
x_train, x_test, y_train, y_test = train_test_split(X_train, y, stratify=y,␣
↪test_size=0.25)
# Realizamos la predicción de test, aplicando los parámetros C y gamma␣
↪obtenidos anteriormente
svm_model = svm.SVC(kernel='rbf',C=1, gamma = 1/2)
svm_model.fit(x_train, y_train)
paramOpt = GridSearchCV(estimator=svm_model, param_grid=dict(C=Cs,gamma=Gs),␣
↪n_jobs=-1, cv=5)
paramOpt.fit(x_train, y_train)
print(paramOpt.score(x_test, y_test))
```