Práctica 4: Máquinas de vectores soporte

Asignatura: Introducción a los modelos computacionales, 4º Grado Ingeniería Informática (Escuela Politécnica Superior de Córdoba - Universidad de Córdoba)

Trabajo realizado por:

-Antonio Gómez Giménez (32730338G)
i72gogia@uco.es

Índice:

1. Pregunta [1]: 2
2. Pregunta [2]: 3
3. Pregunta [3]: 4
4. Pregunta [4]: 8
5. Pregunta [5]: 12
6. Pregunta [6]: 15
7. Pregunta [7]: 15
8. Pregunta [8]: 17
9. Pregunta [9]: 19
10. Pregunta [10]: 20
11. Pregunta [11]: 21
12. Pregunta [12]: 23
13. Pregunta [13]: 23
14. Pregunta [14]: 24
15. Pregunta [15]: 24
16. Pregunta [16]: 26
17. Pregunta [17]: 27

1. Pregunta [1]:

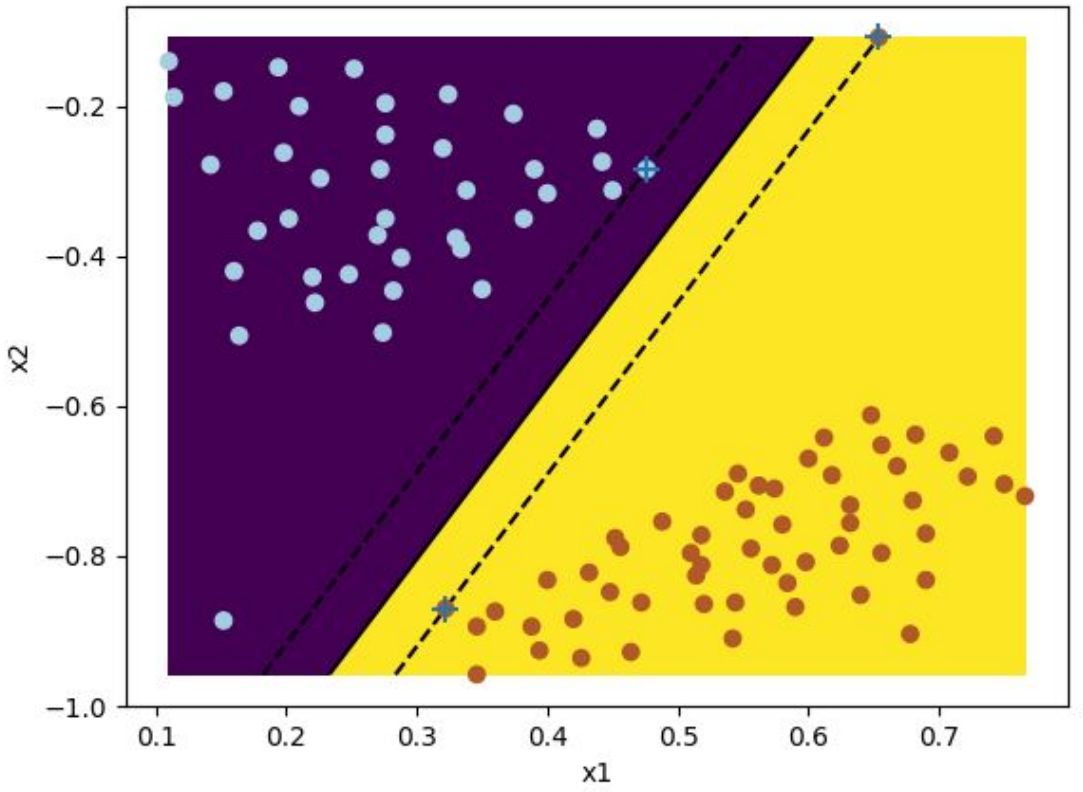
Abre este script y explica su contenido. Podrás ver que se utiliza el primer dataset de ejemplo y se realiza la representación gráfica del SVM. Comenta que tipo de kernel se está utilizando y cuáles son los parámetros de entrenamiento. Explica todos los elementos de la SVM que aparecen en la imagen, junto con los colores.

En este script podemos observar cómo se utiliza el primer dataset de ejemplo siendo el dataset1, posteriormente es necesario separar los datos en matriz de entrada y matriz de salida. Después creamos el SVM con el objeto SVC de SVM y le pasamos como parámetros el kernel que vamos a utilizar, en este caso el lineal, y el coste que va a ser de 1000, al hacer esto, por último lo entrenamos.
El kernel lineal trabaja en el propio espacio de características original, es decir, cuando tengamos la multiplicación entre los vectores soporte, se aplica directamente, en resumen, se aplica la suma del producto de todas las componentes (producto escalar).

Por último, lo representamos de forma gráfica con scatter, pasandole los respectivos valores.
Sacamos los dos vectores que tienen todos los puntos del plano para sacar el hiperplano separador, y tras ajustarlo, lo usamos para pintar la superficie del plano, una parte para la parte positiva y otro color para la parte negativa.
También se dibujan los márgenes con la función contour.
Finalmente se muestra.

Cabe destacar que la variable C, que se usa como parámetro de entrenamiento, nos da la importancia que le damos a los errores, entonces mientras mayor sea C, más importancia se le da a los errores, de tal forma que, los márgenes serán más pequeños.

La salida del programa es la siguiente:

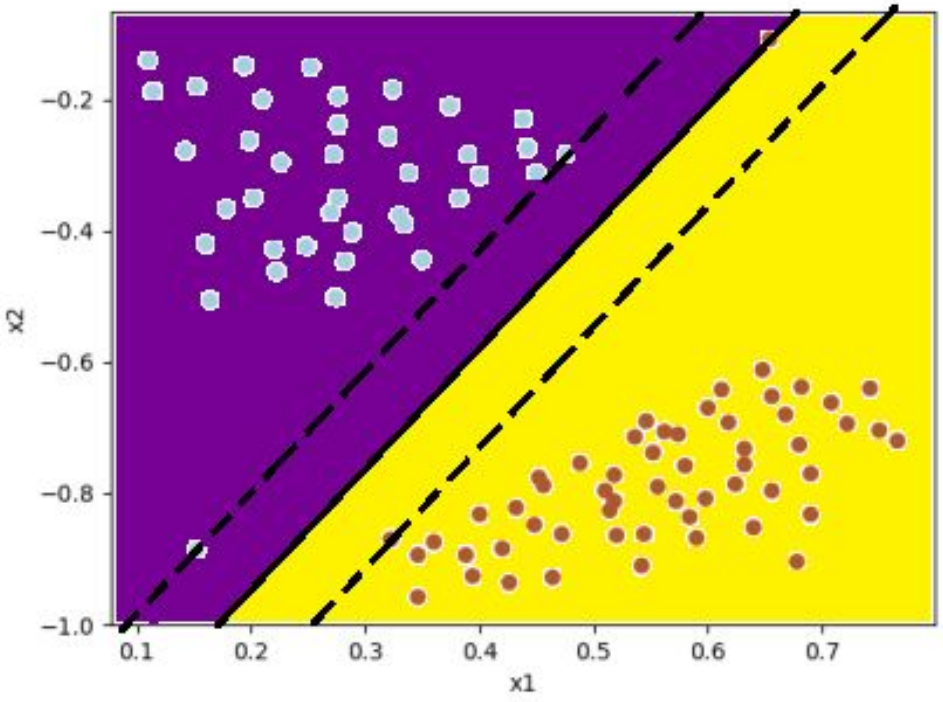


Como podemos observar, la línea negra continua es el hiperplano separador y las líneas discontinuas son los márgenes.
El area de color morado es una clase, en concreto la negativa y el area de color amarillo es la otra clase, en este caso la positiva. Los puntos que encontramos son los patrones, los de color azul pertenecen a la clase negativa y los marrones a la clase positiva. Aquellos patrones que salen con una cruceta, son los patrones que se usan como vectores soporte y se encuentran sobre el margen.

2. Pregunta [2]:

Intuitivamente, ¿qué hiperplano crees que incurriría en un menor error de test en la tarea de separar las dos clases de puntos?.

Creo que sería de la siguiente forma, es necesario clasificar mal un patrón para permitir así ampliar los márgenes.

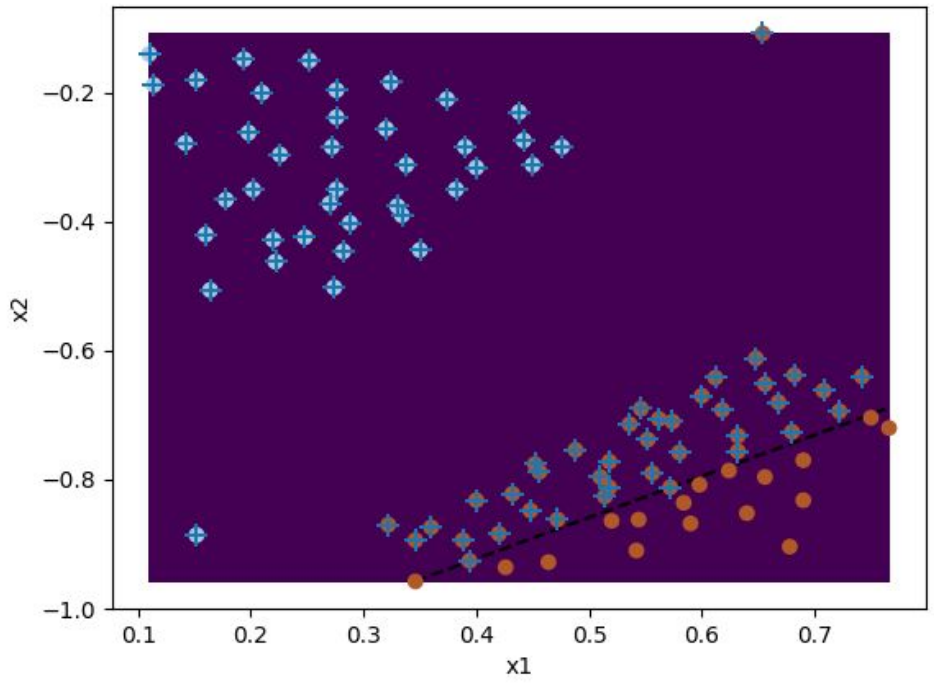


Como acabo de comprobar con la pregunta 1, esta combinación estaría mal, ya que se podría tomar como vectores soporte los dos puntos más externos de los marrones, de esta forma evitamos clasificar mal el patrón y además podemos ampliar de forma considerable el margen.

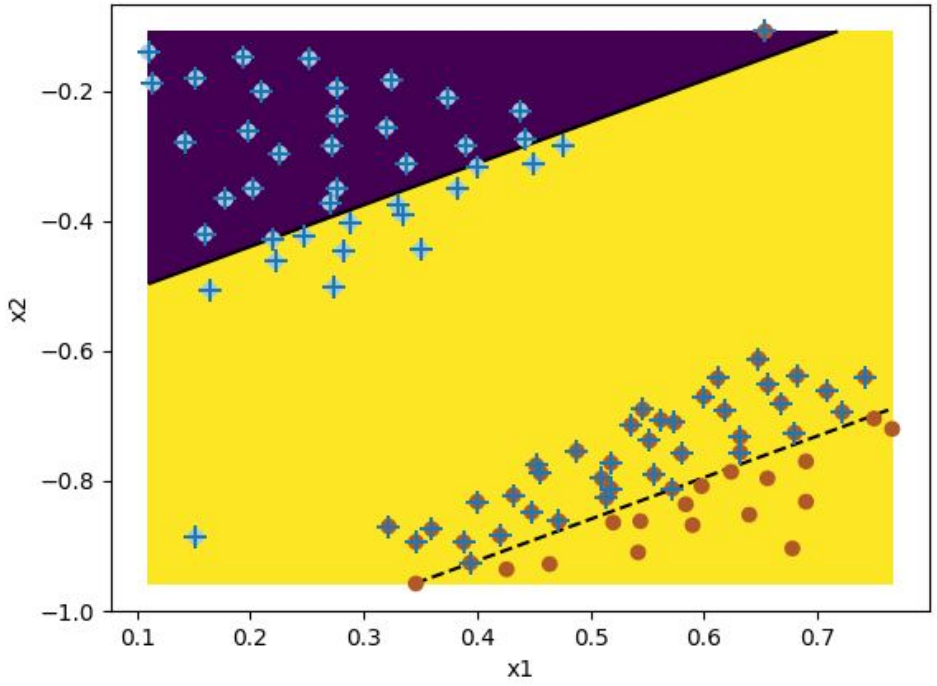
3. Pregunta [3]:

Modifica el script probando varios valores de C, en concreto, C ∈ {10^−2, 10^−1, 10^0, 10^1, 10^2, 10^3, 10^4}. Observa que sucede, explica por qué y escoge el valor más adecuado.

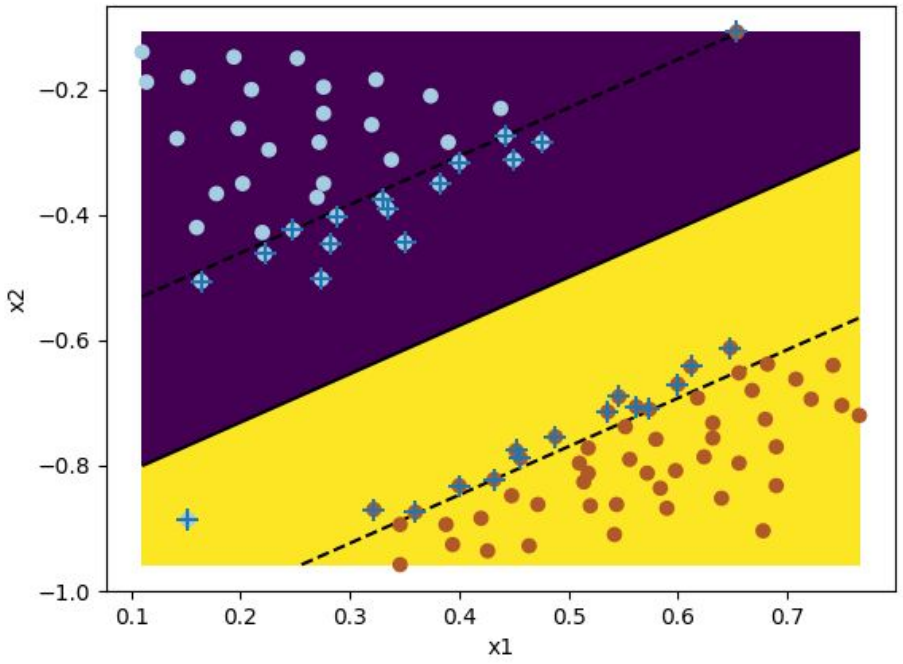
C=10^−2


−0.2
−0.4
−0.6
−0.8
−1.0
x2
0.1
0.2
0.3
0.4
0.5
0.6
0.7
x1

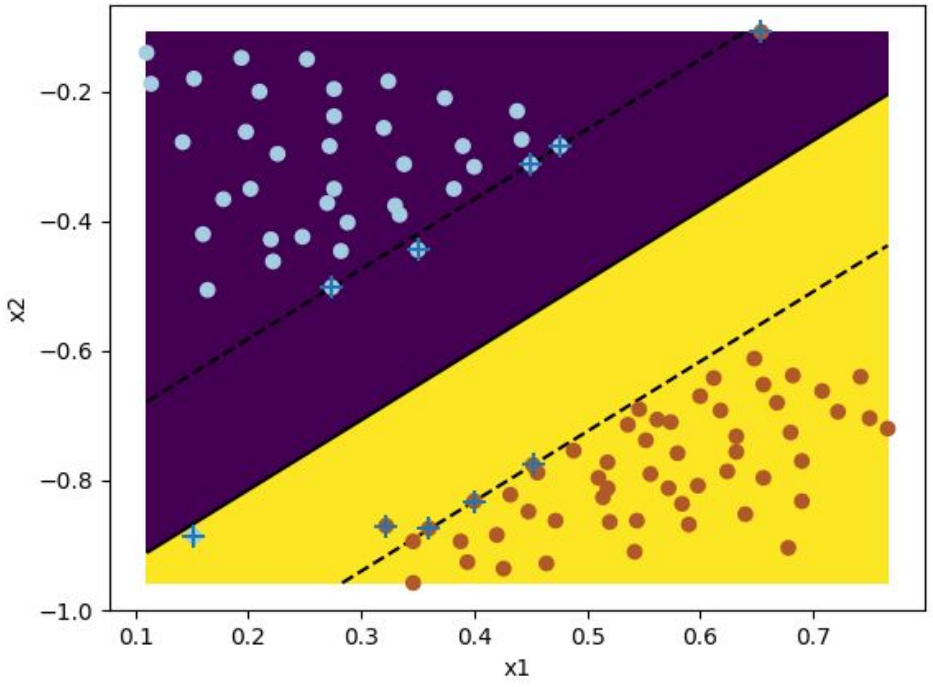
C=10^−1


−0.2
−0.4
−0.6
−0.8
−1.0
x2
0.1
0.2
0.3
0.4
0.5
0.6
0.7
x1

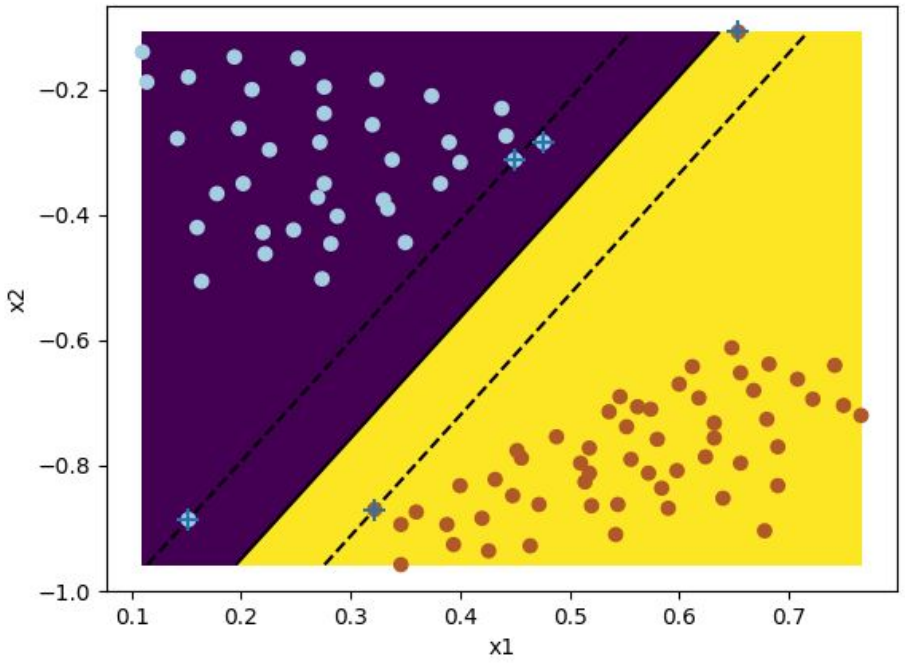
C=10^0


−0.2
−0.4
−0.6
−0.8
−1.0
x2
0.1
0.2
0.3
0.4
0.5
0.6
0.7
x1

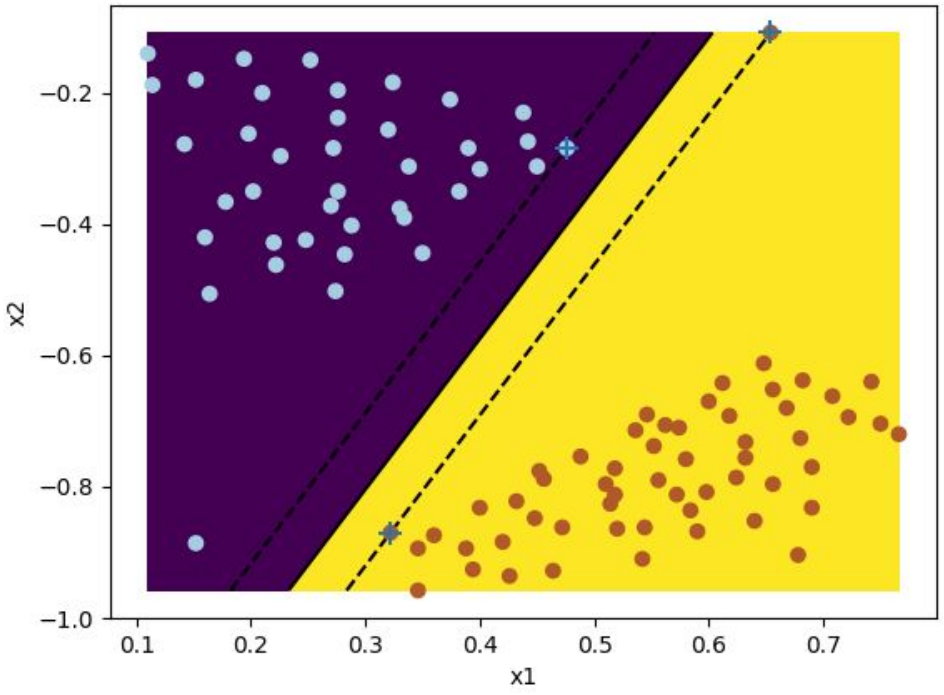
C=10^1


−0.2
−0.4
−0.6
−0.8
−1.0
x2
0.1
0.2
0.3
0.4
0.5
0.6
0.7
x1

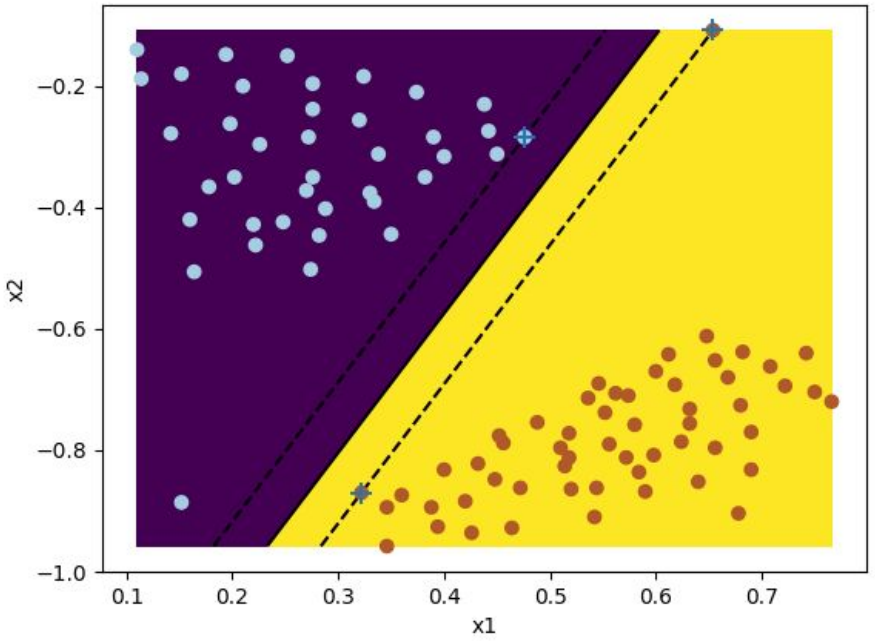
C=10^2


−0.2
−0.4
−0.6
−0.8
−1.0
x2
0.1
0.2
0.3
0.4
0.5
0.6
0.7
x1

C=10^3


−0.2
−0.4
−0.6
−0.8
−1.0
x2
0.1
0.2
0.3
0.4
0.5
0.6
0.7
x1

C=10^4

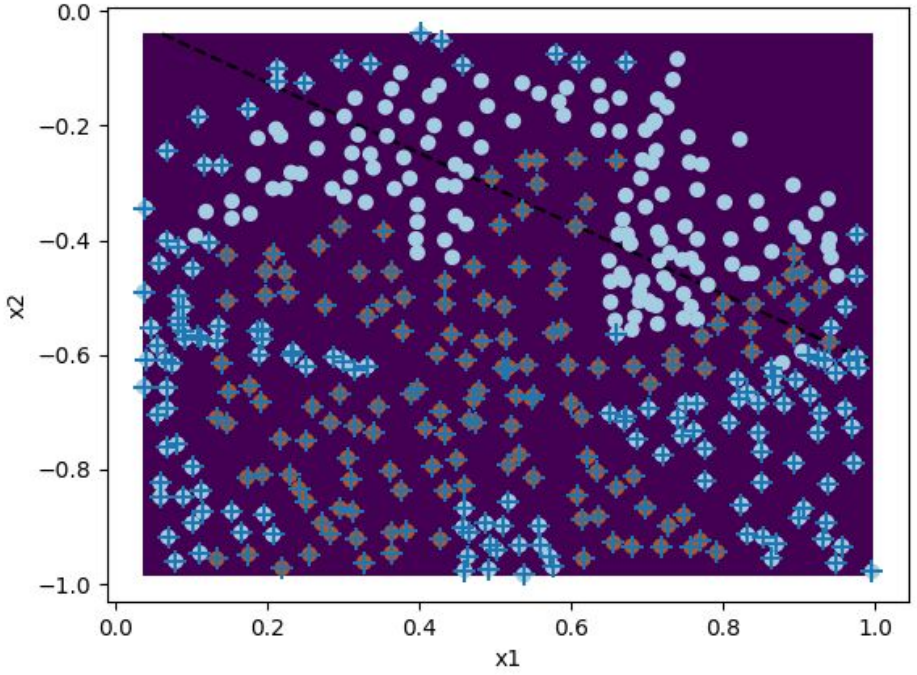


Como podemos observar, a medida que C es más bajo el número de patrones dentro del margen aumenta, esto se debe a que damos menos importancia al error aumentando así el margen. Al aumentar estos patrones se quedan en el margen, cosa que no debería suceder.
Cabe destacar que a partir de cierto valor, en concreto C=10^3, todos los patrones estarían fuera del margen y bien clasificados, ya que es el punto de mayor margen y mínimo error, aunque aumentemos C no va a variar porque no hay errores y el margen es el máximo.
Siempre se busca maximizar el margen para prevenir que los patrones de test tengan más probabilidad de estar bien clasificados.
Entonces el valor más adecuado es cualquier valor superior a C=10^3 ya que maximiza el margen sin errores.

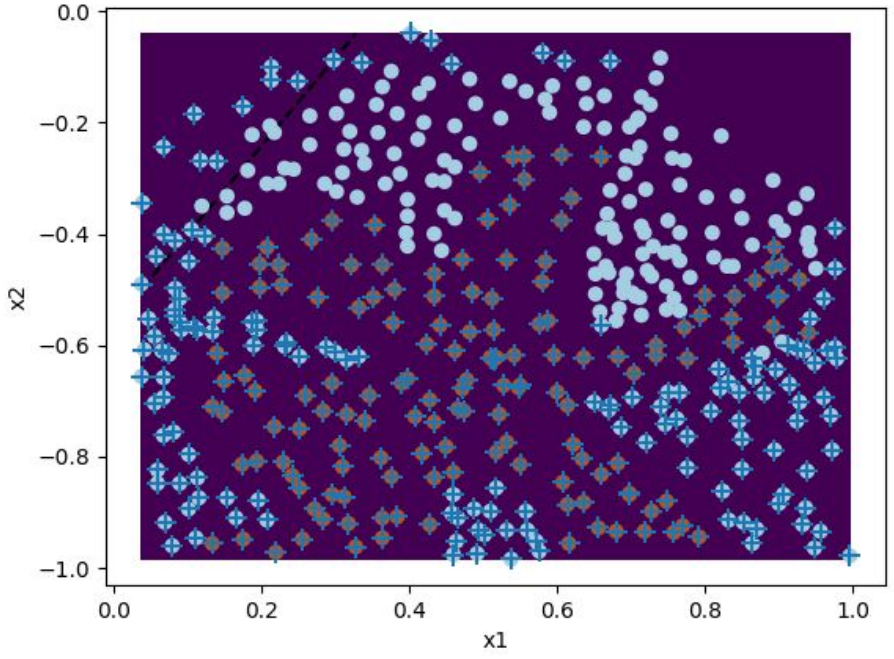
4. Pregunta [4]:

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

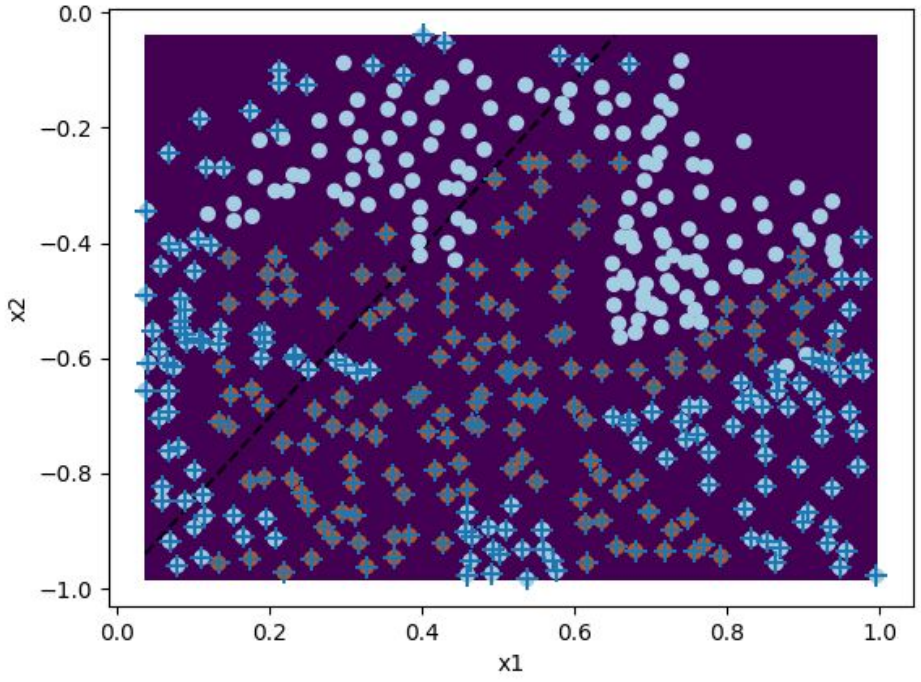
C=10^−2


0.0
−0.2
−0.4
−0.6
−0.8
−1.0
x2
0.0
0.2
0.4
0.6
0.8
1.0
x1

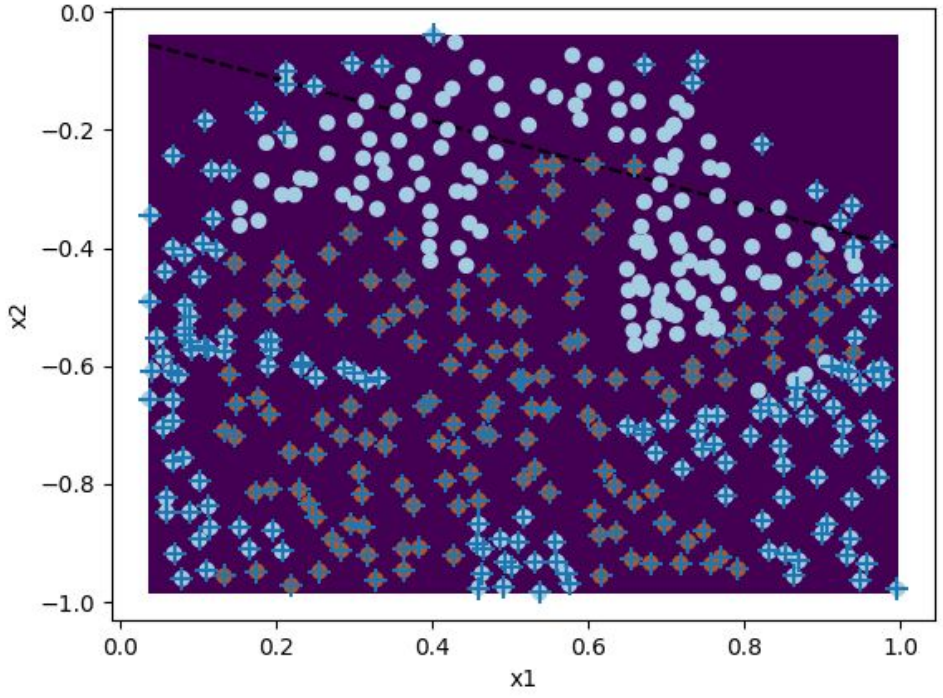
C=10^−1


0.0
−0.2
−0.4
−0.6
−0.8
−1.0
x2
0.0
0.2
0.4
0.6
0.8
1.0
x1

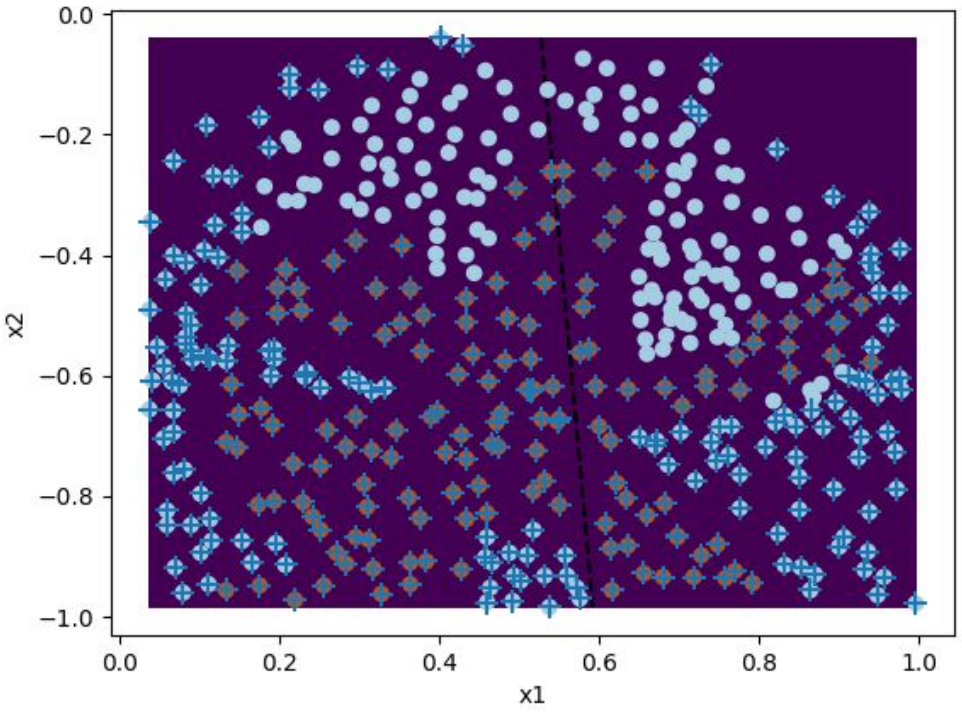
C=10^0


0.0
−0.2
−0.4
−0.6
−0.8
−1.0
x2
0.0
0.2
0.4
0.6
0.8
1.0
x1

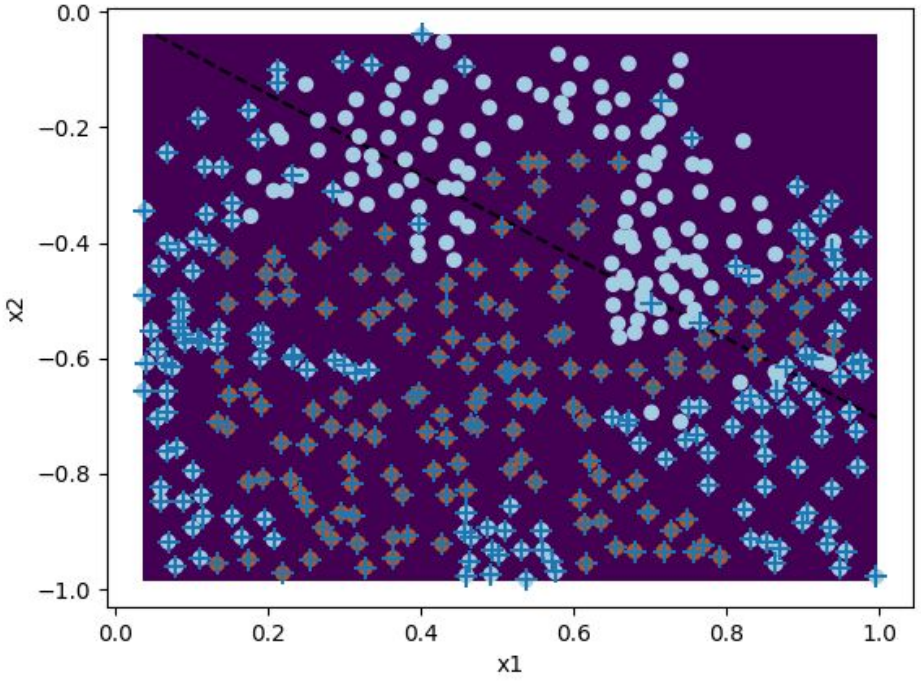
C=10^1


0.0
−0.2
−0.4
−0.6
−0.8
−1.0
x2
0.0
0.2
0.4
0.6
0.8
1.0
x1

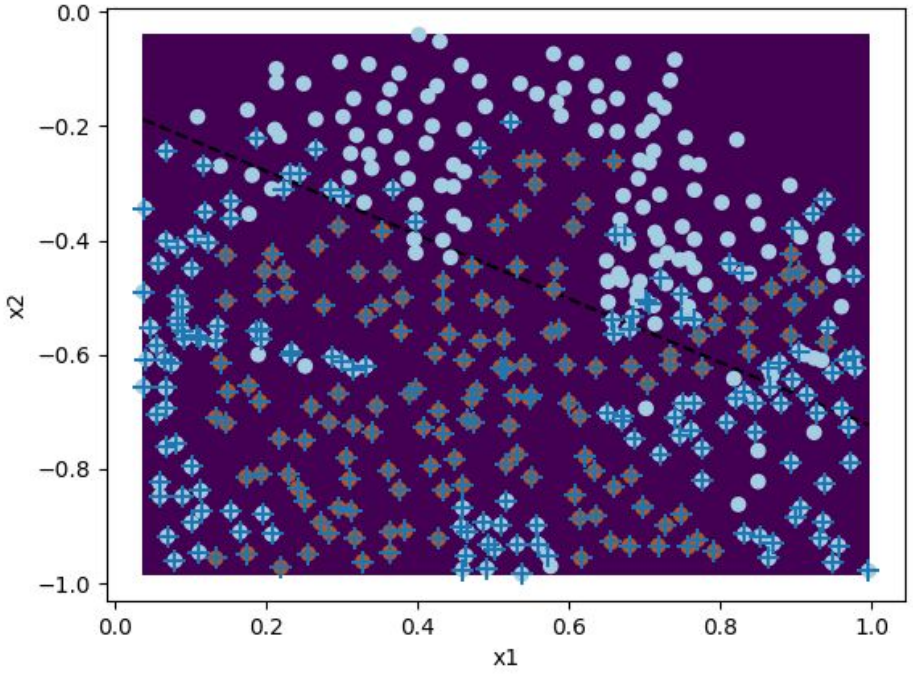
C=10^2


0.0
−0.2
−0.4
−0.6
−0.8
−1.0
x2
0.0
0.2
0.4
0.6
0.8
1.0
x1

C=10^3


0.0
−0.2
−0.4
−0.6
−0.8
−1.0
x2
0.0
0.2
0.4
0.6
0.8
1.0
x1

C=10^4

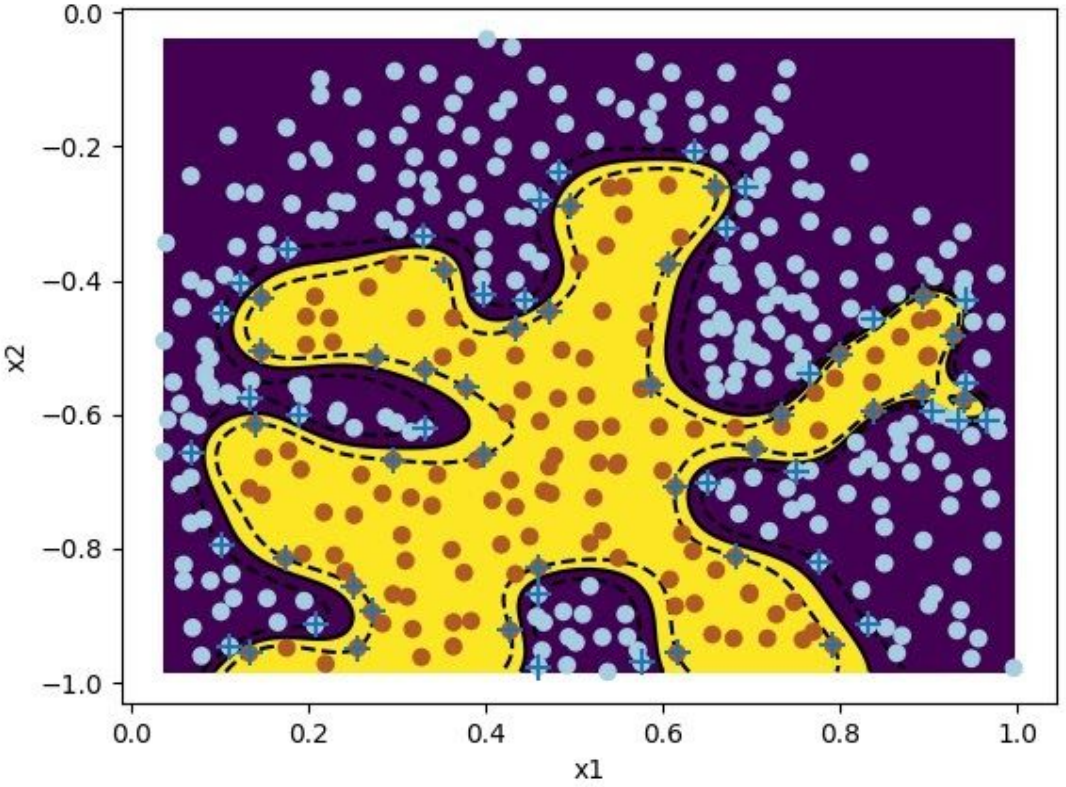


Como podemos observar, este problema no es linealmente separable ya que como podemos observar tiene una estrella(estos patrones serían una clase) la cual está rodeada(de patrones de otra la otra clase). Por lo tanto, es imposible separar los patrones. Será necesario utilizar otro tipo de kernel donde aumentemos la dimensión del espacio de características.

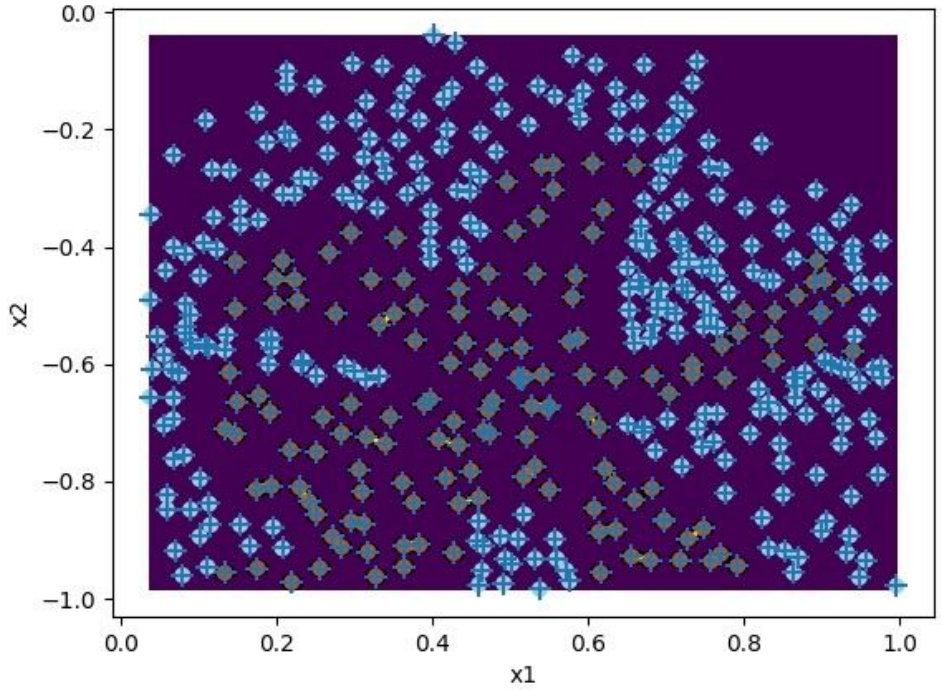
5. Pregunta [5]:

Propón una configuración de SVM no lineal (utilizando el kernel tipo RBF o Gaussiano) que resuelva el problema. Prueba a utilizar valores en el rango C, Gamma ∈ {10^−2 , 10^−1 , 10^0 , 10^1 , 10^2 , 10^3 , 10^4}. El resultado debería ser similar al de la Figura 3. ¿Qué valores has considerado para C y para Gamma?. Además, incluye un ejemplo de una configuración de parámetros que produzca sobre-entrenamiento y otra que produzca infra-entrenamiento.

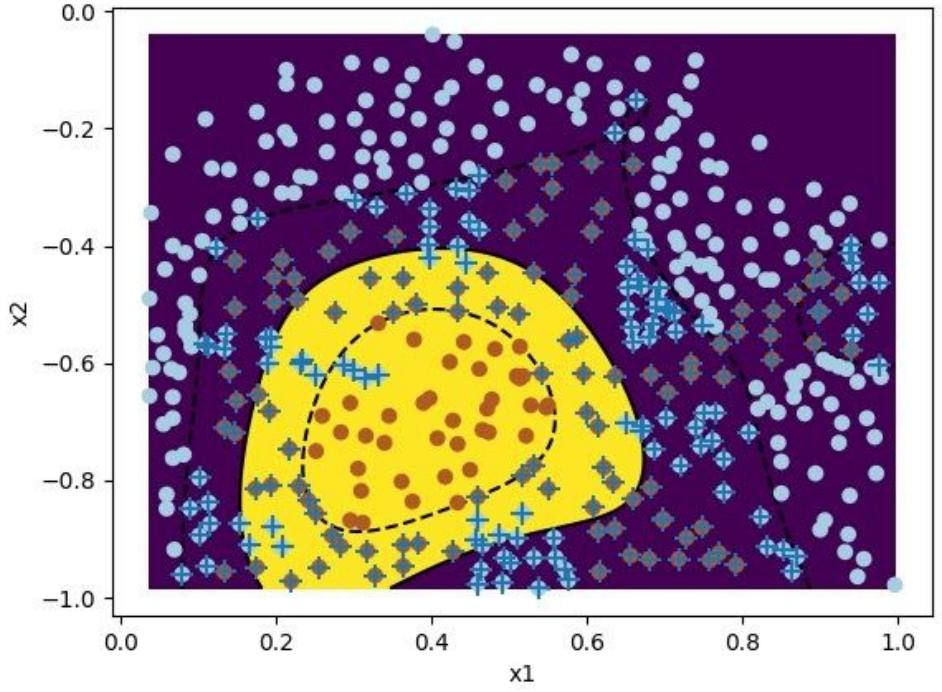
Si consideramos como C=10000 y Gamma=100 podemos observar la siguiente figura donde se clasifican correctamente todos los patrones y se intenta maximizar el margen.



Si consideramos como C=10000 y Gamma=10000 podemos observar que ocurre sobre-entrenamiento ya que para cada patrón se genera una rbf ya que el radio es muy bajo de tal forma que no se puede englobar a patrones cercanos. Todos los patrones estarían bien clasificados pero no se podría generalizar este modelo ya que con que se varíe un poco los patrones la mayoría se clasificaron mal.



Si consideramos como C=1 y Gamma=10 podemos observar que ocurre infra-entrenamiento, ya que genera un círculo y hay muchísimos patrones mal clasificados.



Cabe destacar que a mayor gama menor radio en las rbf, y por tanto reducimos la capacidad de agrupar patrones.
En resumen, si utilizamos valores muy altos tanto de gamma como de C, se produce sobre-entrenamiento y si usamos valores muy bajos se produce infra-entrenamiento. Por ello debemos buscar valores que generen mayor margen si cometer muchos errores y ello se consigue con valores intermedios de C y Gamma.

6. Pregunta [6]:

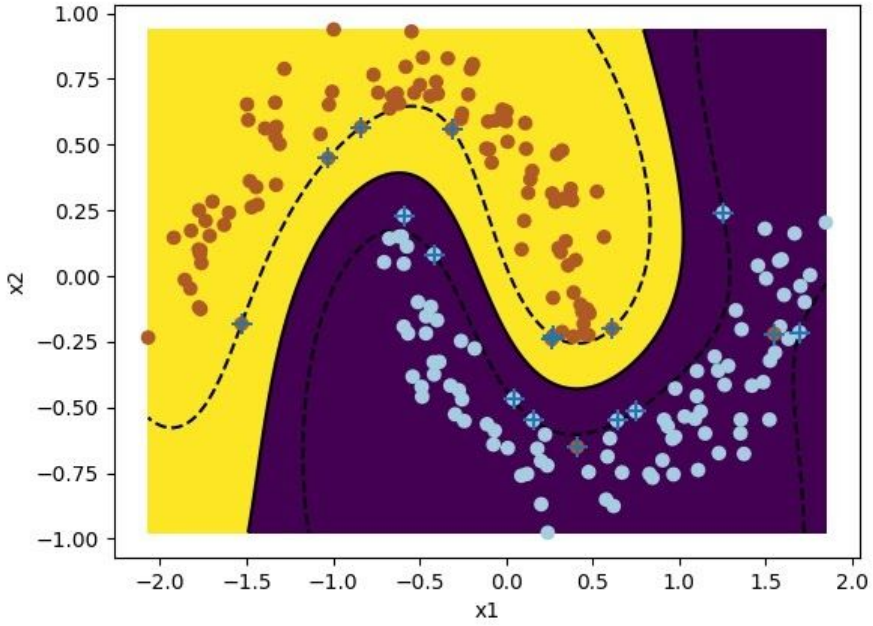
En este caso, ¿es el dataset linealmente separable?. A primera vista, ¿detectamos puntos que presumiblemente sean outliers?, ¿por qué?.

En este caso, el dataset no es linealmente separable ya que una función polinómica no aumenta las probabilidades de que en generalización funcione correctamente, ya que no aumentamos el espacio de dimensiones.
Podemos encontrar outliers, ya que observamos patrones positivos mezclados entre patrones negativos.

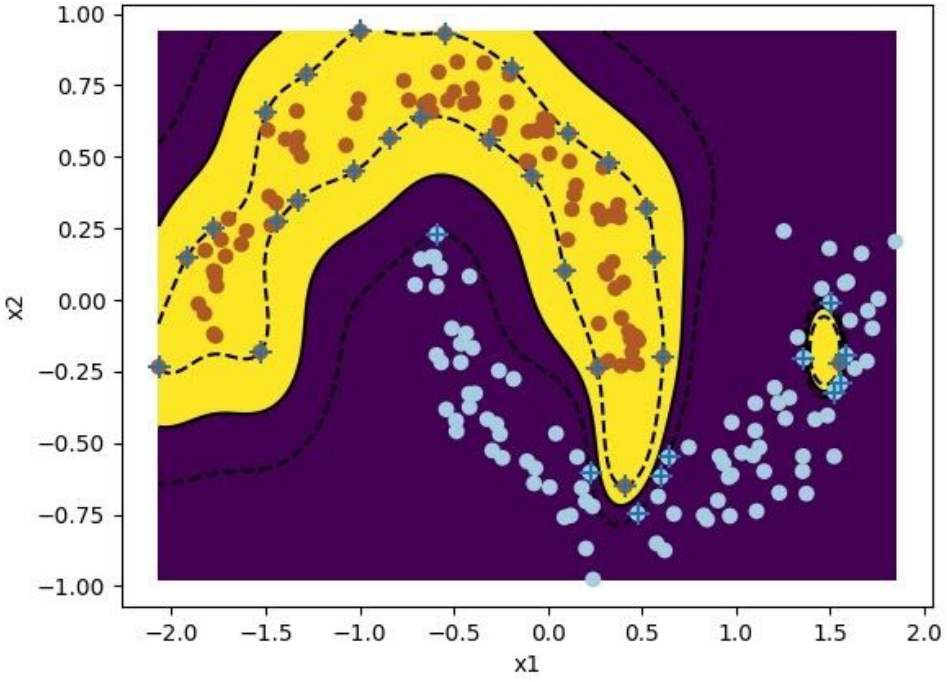
7. Pregunta [7]:

Propón una configuración de SVM no lineal (utilizando el kernel tipo RBF o Gaussiano) que resuelva el problema. Prueba a utilizar valores en el rango C, Gamma ∈ {10^−2 , 10^−1 , 10^0 , 10^1 , 10^2 , 10^3 , 10^4}. El resultado debería ser similar al de la Figura 3. ¿Qué valores has considerado para C y para Gamma?. Además, incluye un ejemplo de una configuración de parámetros que produzca sobre-entrenamiento y otra que produzca infra-entrenamiento.

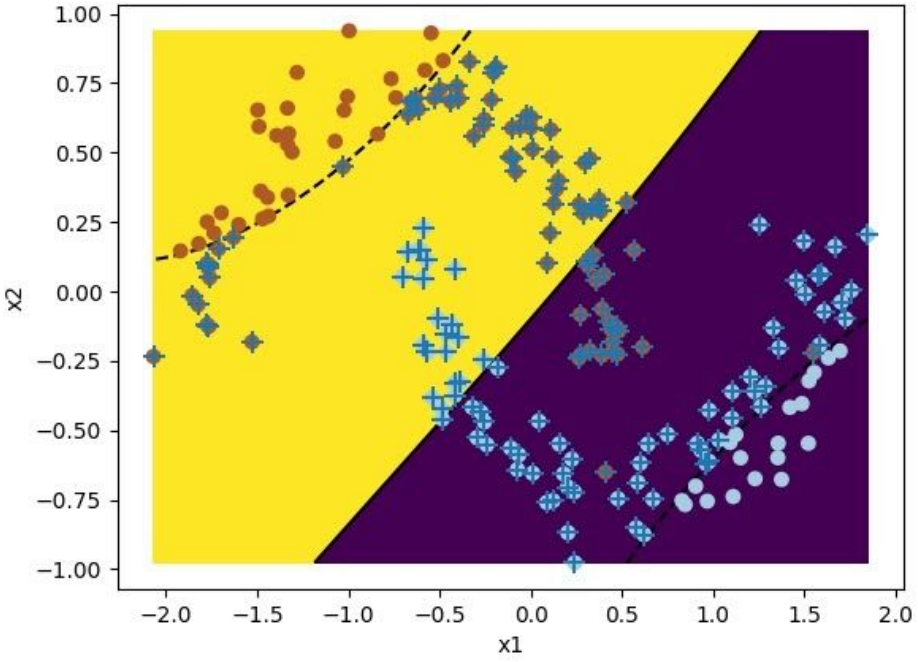
En este caso como hay outliers, no nos renta tener un C grande, ya que hay patrones que necesitamos que se detecte como error, por ello una buena configuración sería con C=10 y Gamma=1.



Si consideramos como C=1000 y Gamma=10 podemos observar que ocurre sobre-entrenamiento ya que para el outlier genera una región de clase positiva cuando eso carece de sentido y en generalización dará por tanto muchos errores .



Para el caso de infra-entrenamiento, se ve bastante claro si escogemos la configuración donde C=0.1 y Gamma=0.1, ya que es similar a usar un kernel lineal donde hay muchos patrones mal clasificados.



8. Pregunta [8]:

Vamos a reproducir este proceso en Python. Divide el dataset sintético 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 (estandarización, entrenamiento y predicción), volviendo a optimizar los valores de C y γ que obtuviste en la última pregunta. Comprueba el porcentaje de buena clasificación que se obtiene para el conjunto de test. Repite el proceso más de una vez para comprobar que los resultados dependen mucho de la semilla utilizada para hacer la partición.

La primera vez que realizamos este experimento, obtenemos los siguientes resultados:

```
C= 0.01  y Gamma= 0.01  -> CCR= 52.94117647058824
C= 0.01  y Gamma= 0.1  -> CCR= 52.94117647058824
C= 0.01  y Gamma= 1.0  -> CCR= 52.94117647058824
C= 0.01  y Gamma= 10.0  -> CCR= 52.94117647058824
C= 0.01  y Gamma= 100.0  -> CCR= 52.94117647058824
C= 0.01  y Gamma= 1000.0  -> CCR= 52.94117647058824
C= 0.01  y Gamma= 10000.0  -> CCR= 52.94117647058824
C= 0.1  y Gamma= 0.01  -> CCR= 52.94117647058824
C= 0.1  y Gamma= 0.1  -> CCR= 86.27450980392157
C= 0.1  y Gamma= 1.0  -> CCR= 94.11764705882352
C= 0.1  y Gamma= 10.0  -> CCR= 98.0392156862745
C= 0.1  y Gamma= 100.0  -> CCR= 52.94117647058824
C= 0.1  y Gamma= 1000.0  -> CCR= 52.94117647058824
C= 0.1  y Gamma= 10000.0  -> CCR= 52.94117647058824
C= 1.0  y Gamma= 0.01  -> CCR= 84.31372549019608
C= 1.0  y Gamma= 0.1  -> CCR= 86.27450980392157
C= 1.0  y Gamma= 1.0  -> CCR= 100.0
C= 1.0  y Gamma= 10.0  -> CCR= 100.0
C= 1.0  y Gamma= 100.0  -> CCR= 96.07843137254902
C= 1.0  y Gamma= 1000.0  -> CCR= 52.94117647058824
C= 1.0  y Gamma= 10000.0  -> CCR= 52.94117647058824
C= 10.0  y Gamma= 0.01  -> CCR= 86.27450980392157
C= 10.0  y Gamma= 0.1  -> CCR= 90.19607843137256
C= 10.0  y Gamma= 1.0  -> CCR= 100.0
C= 10.0  y Gamma= 10.0  -> CCR= 100.0
C= 10.0  y Gamma= 100.0  -> CCR= 96.07843137254902
C= 10.0  y Gamma= 1000.0  -> CCR= 54.90196078431373
C= 10.0  y Gamma= 10000.0  -> CCR= 52.94117647058824
C= 100.0  y Gamma= 0.01  -> CCR= 84.31372549019608
C= 100.0  y Gamma= 0.1  -> CCR= 98.0392156862745
C= 100.0  y Gamma= 1.0  -> CCR= 100.0
C= 100.0  y Gamma= 10.0  -> CCR= 96.07843137254902
C= 100.0  y Gamma= 100.0  -> CCR= 96.07843137254902
C= 100.0  y Gamma= 1000.0  -> CCR= 54.90196078431373
C= 100.0  y Gamma= 10000.0  -> CCR= 52.94117647058824
C= 1000.0  y Gamma= 0.01  -> CCR= 86.27450980392157
C= 1000.0  y Gamma= 0.1  -> CCR= 98.0392156862745
C= 1000.0  y Gamma= 1.0  -> CCR= 98.0392156862745
C= 1000.0  y Gamma= 10.0  -> CCR= 96.07843137254902
C= 1000.0  y Gamma= 100.0  -> CCR= 96.07843137254902
C= 1000.0  y Gamma= 1000.0  -> CCR= 54.90196078431373
C= 1000.0  y Gamma= 10000.0  -> CCR= 52.94117647058824
C= 10000.0  y Gamma= 0.01  -> CCR= 90.19607843137256
C= 10000.0  y Gamma= 0.1  -> CCR= 100.0
C= 10000.0  y Gamma= 1.0  -> CCR= 96.07843137254902
C= 10000.0  y Gamma= 10.0  -> CCR= 96.07843137254902
C= 10000.0  y Gamma= 100.0  -> CCR= 96.07843137254902
C= 10000.0  y Gamma= 1000.0  -> CCR= 54.90196078431373
C= 10000.0  y Gamma= 10000.0  -> CCR= 52.94117647058824
```

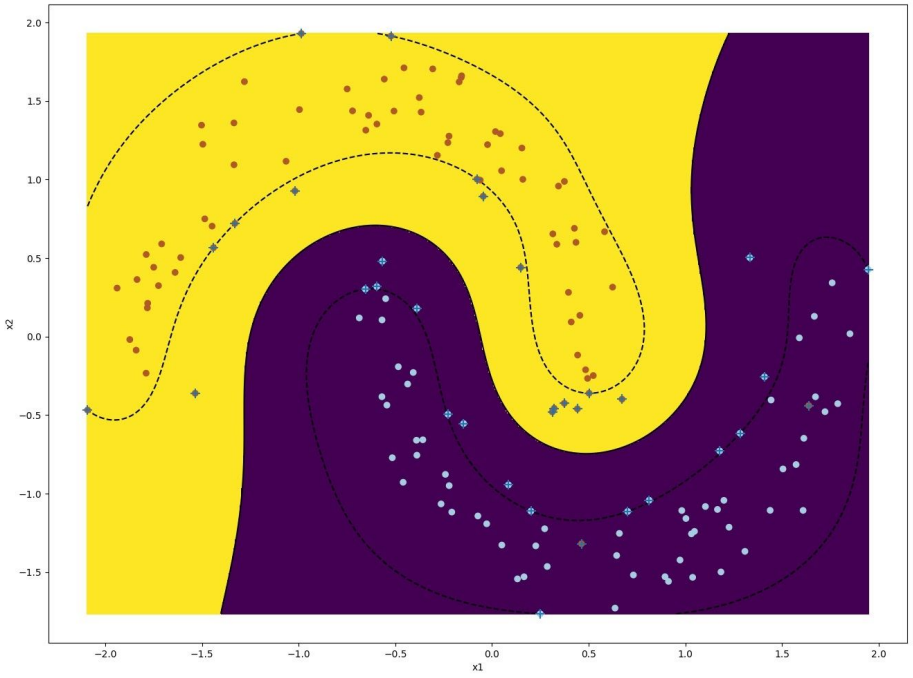
Tras realizar este experimento varias veces, se puede llegar a la conclusión de que se ve afectado en gran medida por la aleatoriedad de la semilla, esto se puede apreciar claramente si nos fijamos en los outliers.
Si el outlier se encuentra en entrenamiento, ciertas configuraciones se adaptarán a este patrón siendo capaz de clasificarlo correctamente si apareciera de nuevo en el test. Pero en el caso de que el outlier se encuentre en la segmentación de test, sería prácticamente imposible tener un CCR de 100 ya que no ha existido ningún patrón en entrenamiento capaz de entrenar ese tipo de patrón.
Cabe destacar que este es solo un ejemplo pero pueden darse otros casos donde se elija para entrenamiento una la parte izquierda y todos los patrones de test sean de la derecha, de esta forma fallaría de forma considerable el test.

Las mejores configuraciones son aquellas con un porcentaje del 100% pero cabe destacar que depende en gran medida de la semilla ya que al ejecutar de nuevo el programa puede ser que baje este porcentaje o que aumenten otros que eran de un porcentaje más bajo.

9. Pregunta [9]:

Amplía el código anterior para realizar el entrenamiento de la pregunta 8 sin necesidad de especificar los valores de C y γ. Compara los valores óptimos obtenidos para ambos parámetros con los que obtuviste a mano. Extiende el rango de valores a explorar, si es que lo consideras necesario.

Tras realizar las pruebas, el valor óptimo dado es de C=1 y Gamma=1, siendo el CCR de 100.
Aun así he probado ha volver a ejecutar y a veces salía CCR de 98, esto ocurre por lo explicado anteriormente, referente a los outliers.
Si comparamos esta configuración con la del ejercicio anterior es igual, dando un valor de CCR de 100. Probablemente haya también otras configuraciones de parámetros que den CCR de 100 pero con esta función nos quedamos con la primera.
Visualmente, el resultado de la mejor combinación sería el siguiente:

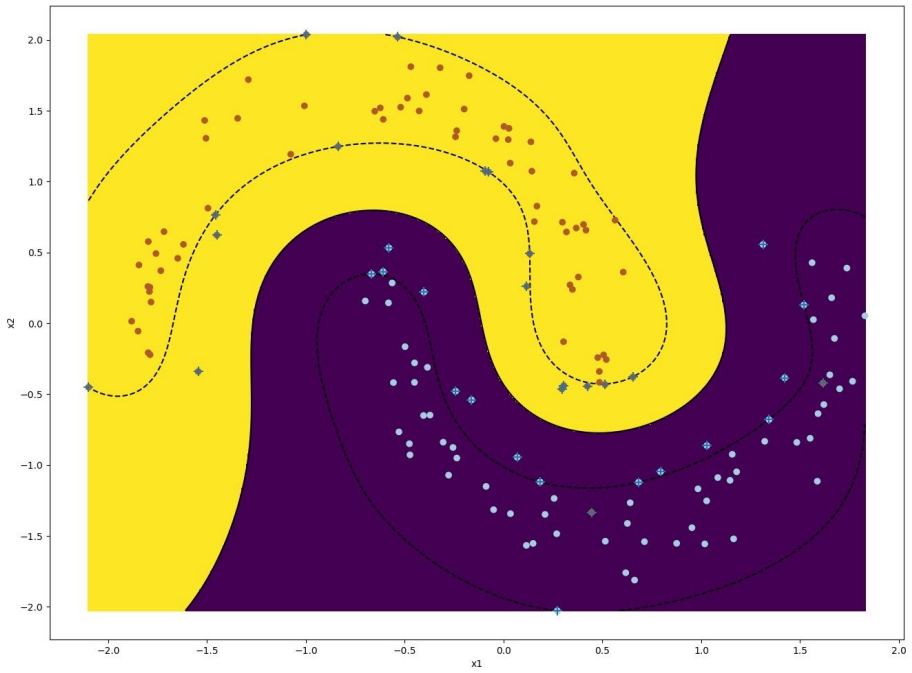


Como se puede observar, todos los parámetros están correctamente clasificados aunque hay patrones en el margen. Por tanto, cuanto se generalice no será tan robusto.

10. Pregunta [10]:

¿Qué inconvenientes observas en ajustar el valor de los parámetros "á mano", viendo el porcentaje de buena clasificación en el conjunto de test (lo que se hizo en la pregunta 8)?.

No sería lo correcto, estaríamos haciendo trampa, por lo tanto a la hora de generalizar, los resultados serían bastante malos. Esto ocurre ya que aunque el CCR que obtenemos es muy bueno se debe a que hemos entrenado con los patrones de test, esto hace que a la verdadera hora de generalizar pueda fallar.
También cabe destacar que al hacerlo a mano perdemos la capacidad de paralelización, por tanto a la hora de ejecutar el programa irá muchísimo más lento.
Se puede paralelizar en este ejercicio, ya que las pruebas se pueden hacer por separado porque no tienen relación entre sí.

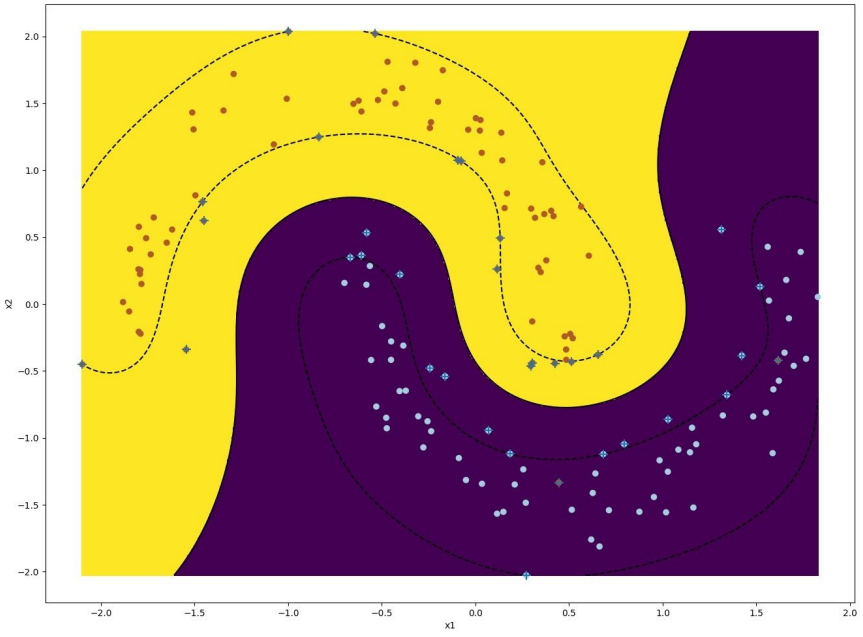


11. Pregunta [11]:

Para estar seguros de que has entendido cómo se realiza la búsqueda de parámetros, implementa de forma manual (sin usar GridSearchCV) la validación cruzada anidada tipo K-fold expuesta en esta sección. Te puede ser útil el uso de listas por comprensión y la clase StratifiedKFold. Compara los resultados con los que obtienes usando GridSearchCV.

```
C= 100.0  y Gamma= 0.01  -> CCR= 86.08602150537635
C= 100.0  y Gamma= 0.1  -> CCR= 96.70967741935485
C= 100.0  y Gamma= 1.0  -> CCR= 100.0
C= 100.0  y Gamma= 10.0  -> CCR= 100.0
C= 100.0  y Gamma= 100.0  -> CCR= 96.70967741935485
C= 100.0  y Gamma= 1000.0  -> CCR= 68.88172043010753
C= 100.0  y Gamma= 10000.0  -> CCR= 56.27956989247313
C= 1000.0  y Gamma= 0.01  -> CCR= 85.41935483870968
C= 1000.0  y Gamma= 0.1  -> CCR= 98.70967741935485
C= 1000.0  y Gamma= 1.0  -> CCR= 100.0
C= 1000.0  y Gamma= 10.0  -> CCR= 100.0
C= 1000.0  y Gamma= 100.0  -> CCR= 96.70967741935485
C= 1000.0  y Gamma= 1000.0  -> CCR= 68.88172043010753
C= 1000.0  y Gamma= 10000.0  -> CCR= 56.27956989247313
C= 10000.0  y Gamma= 0.01  -> CCR= 88.73118279569893
C= 10000.0  y Gamma= 0.1  -> CCR= 98.70967741935485
C= 10000.0  y Gamma= 1.0  -> CCR= 100.0
C= 10000.0  y Gamma= 10.0  -> CCR= 100.0
C= 10000.0  y Gamma= 100.0  -> CCR= 96.70967741935485
C= 10000.0  y Gamma= 1000.0  -> CCR= 68.88172043010753
C= 10000.0  y Gamma= 10000.0  -> CCR= 56.27956989247313
Final
C= 1.0  y Gamma= 1.0  -> CCR= 100.0
```

Como podemos observar, la mejor configuración es C=1 y Gamma=1, por lo tanto los resultados con respecto al uso de **GridSearchCV** son idénticos.
La gráfica resultante sería la siguiente:



12. Pregunta [12]:

Utiliza el script que desarrollaste en la pregunta 9 para entrenar esta base de datos. Observe el valor de CCR obtenido para el conjunto de generalización y compáralo con el obtenido en prácticas anteriores.
El proceso puede demorarse bastante. Al finalizar, toma nota de los valores óptimos obtenidos para los parámetros.

Los valores óptimos que hemos obtenidos son C=10 y Gamma=0.01, siendo el CCR obtenido de 78.33%.
Con la mejor configuración de red rbf, el CCR era de 88.67% y con la mejor configuración para perceptrón multicapa para problemas de clasificación, el CCR era de 84,33%.
Por tanto, si comparamos el CCR obtenido por el SVM, podemos observar que es peor que los anteriores CCR obtenidos en otros métodos para la base de datos nomists.
Esto se debe a que SVM funciona mejor para problemas que son biclase, y la base de datos nomist consta de 6 clases, por ello da peores resultados.

13. Pregunta [13]:

Localiza donde se especifica el valor de K para la validación cruzada interna y el rango de valores que se han utilizado para los parámetros C y γ. ¿Cómo podrías 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.

Vamos a comprobar cuánto tarda en ejecutarse el programa para distintos K y ver qué CCR obtenemos:

k=3->tiempo=2m34s, C=10, Gamma=0.01, CCR=78.33%
k=5->tiempo=5m7s, C=10, Gamma=0.01, CCR=78.33%
k=10->tiempo=8m13s, C10=, Gamma=0,01, CCR=78.33%

Como podemos observar, a mayor k, mayor tiempo computacional pero aunque aumente este valor, el CCR permanece constante. Por tanto, si queremos reducir el tiempo computacional, simplemente debemos realizar un k-fold menor.

Esto ocurre porque al aumentar K, realizamos más particiones y muchos más entrenamiento. Sin embargo, ya sea con K=3 o K=10, todos llegan a encontrar la misma configuración donde el CCR es 78.33%.

14. 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 configuración.

```
antoniogg@antoniogg-SATELLITE-L50D-C:~/Escritorio/IMC/Practica4/Datasets/csv$ ti
me python3 ./libsvm.py
{'C': 0.1}
-> CCR= 98.9 -> Score= 0.989

real    1m41,991s
user    4m30,173s
sys     0m7,498s
```

Como podemos observar la configuración óptima sería con un C=0.1, dando como resultado un CCR de 98.9%.
En este caso no podemos observar los distintos resultados de C porque con la función **GridSearchCV,** los compara internamente devolviendonos la mejor configuración en este caso la anteriormente especificada.

15. Pregunta [15]:

Para la mejor configuración, construye la matriz de confusión 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 patrón, cuando xi es igual a 1 quiere decir que la palabra i-ésima del vocabulario aparece, al menos una vez, en el correo.

La matriz de confusión de sería la siguiente:

```
[[684   8]
 [  3 305]]
```

Como podemos observar nuestro clasificador nos da 684 patrones bien clasificados positivos y 305 patrones bien clasificados negativos, es decir, 684 patrones son spam y 305 no lo son.
Nuestro clasificador tiene 3 patrones que no considera spam y que si lo son y 8 patrones que si considera spam y no lo son.

Vamos a observar algunos de los patrones que fallan:

Patrón 10 --> 1 | 0, se esperaba que fuera spam y nuestro programa nos da que no lo es. Las palabras usadas en el correo son la siguientes:
['emailaddr' 'for' 'group' 'httpaddr' 'inform' 'irish' 'linux' 'list' 'maintain' 'subscript' 'un' 'user']

Si nos fijamos, hay un par de palabras que podríamos entender como palabras de un mensaje de spam, estas serían: 'emailaddr', 'subscript' y 'subscript'. Aun así, nuestro clasificador lo clasifica mal, quizás se debe a que a la hora de entrenar, no teníamos patrones similares a este tipo patrón del test.

Patrón 22 --> 0 | 1, se esperaba que no fuera spam y nuestro programa nos da que sí lo es. Las palabras usadas en el correo son las siguientes:
['an' 'and' 'at' 'be' 'by' 'call' 'can' 'contact' 'email' 'emailaddr' 'geek' 'heaven' 'httpaddr' 'if' 'immedi' 'is' 'list' 'mail' 'messag' 'need' 'net' 'not' 'number' 'of' 'offic' 'our' 'out' 'repres' 'respond' 'return' 'sf' 'spamassassin' 'sponsor' 'start' 'talk' 'the' 'there' 'thi' 'thinkgeek' 'to' 'until' 'welcom' 'when' 'will' 'you' 'your']

Si nos fijamos hay muchas palabras que se podrían usar en un correo típico de spam, por ello, es normal que lo clasifique mal. Al parecer aquellos correos donde se habla sobre spam, los suele clasificar como spam, hay varios casos donde sucede, este es uno de ellos y se puede observar con la palabra 'spamassassin' .

Patrón 527 --> 0 | 1, se esperaba que no fuera spam y nuestro programa nos da que sí lo es. Las palabras usadas en el correo son las siguientes:
['about' 'ad' 'advertis' 'after' 'am' 'an' 'and' 'as' 'at' 'been' 'below' 'bodi' 'but' 'by' 'click' 'contain' 'dai' 'direct' 'doc' 'email' 'emailaddr' 'except' 'extra' 'fals' 'final' 'geek' 'get' 'ha' 'have' 'heaven' 'here' 'httpaddr' 'is' 'it' 'list' 'look' 'mail' 'mark' 'net' 'next' 'no' 'number' 'off' 'on' 'other' 'posit' 'razor' 'see' 'sender' 'sf' 'signatur' 'so' 'spam' 'sponsor' 'such' 'talk' 'the' 'thi' 'thinkgeek' 'to' 'turn' 'us' 'user' 'welcom' 'where' 'which' 'with' 'word']

Si nos fijamos, es el caso exactamente igual al patrón anterior, se puede ver perfectamente el uso de la palabra 'spam'.

De hecho, parece que este patrón es ruido ya que hay demasiadas palabras que son normales en un correo de spam como click, sponsor, etc. Este patrón debería pertenecer a la clase positiva.

16. Pregunta [16]:

Compara los resultados obtenidos con los resultados utilizando una red RBF. Para ello, haz uso del programa desarrollado en la práctica anterior. Utiliza solo una semilla (la que mejor resultado obtenga).

Si usamos el programa de la práctica anterior, utilizando como parámetros ratio=0.05 y una tasa de aprendizaje de 0.1, la seed que da mejores resultados es la cuatro:

```
-----------
Seed: 4
-----------
Number of RBFs used: 200
Confusion Matrix:
[[687   5]
 [ 11 297]]
Training MSE: 0.020250
Test MSE: 0.016000
Training CCR: 97.97%
Test CCR: 98.40%
Coeficientes: 111.000000
```

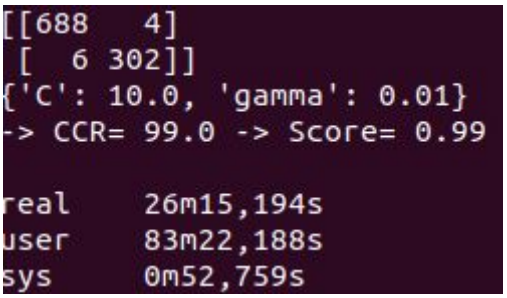
```
real    1m51,858s
user    2m18,696s
sys     0m18,421s
```

Como podemos observar, el tiempo de cómputo es similar, y el CCR es prácticamente el mismo pero el clasificador que usa SVM da un mejor CCR por poco. La diferencia es prácticamente nula.
Puede ser que con otras configuraciones mejores, por ejemplo, he probado con r=0.5 y el CCR aumenta a 99%, pero también aumenta el tiempo de ejecución, en concreto, 4m51.

17. Pregunta [17]:

Entrena una SVM no lineal y compara los resultados.

Si usamos una SVM no lineal, los resultados son los siguientes:



Como podemos observar, el tiempo computacional se dispara y esto es lógico, ya que estamos optimizando dos parámetros, C y Gamma, aparte de realizar los k-fold propuestos (5 igual que los usados para el lineal).
En total sería 4x4x5 entrenamientos para toda la base de datos, por eso es normal que tarde tanto.

Respecto al CCR, aumenta y es lógico ya que para un no lineal es más fácil separar los patrones entre sí ya que el espacio de dimensiones es mayor, aun así, la diferencia no es muy significativa y en proporción al tiempo computacional no es rentable.