UNIVERSIDAD
D
CÓRDOBA

INTRODUCCIÓN A LOS MODELOS COMPUTACIONALES

Práctica 4: Máquina de Vectores Soporte

4º Curso-Grado en Ingeniería Informática

UCO-Escuela Politécnica Superior de Córdoba

Manuel Casas Castro - 31875931R

i72cascm@uco.es

Pregunta [1]: **3**

Pregunta [2]: **3**

Pregunta [3]: **4**

Pregunta [4]: **6**

Pregunta [5]: **7**

Pregunta [6]: **8**

Pregunta [7]: **9**

Pregunta [8]: **10**

Pregunta [9]: **12**

Pregunta [10]: **13**

Pregunta [11]: **13**

Pregunta [12]: **14**

Pregunta [13]: **14**

Pregunta [14]: **15**

Pregunta [15]: **16**

Pregunta [16]: **17**

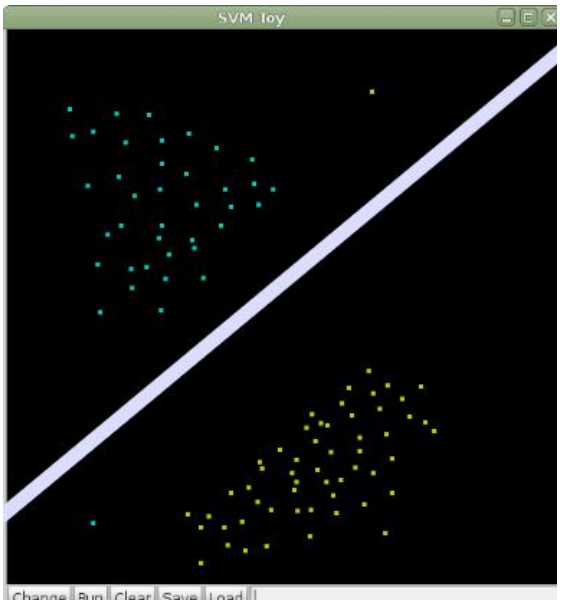
Pregunta [17]: **19**

Bibliografía **20**

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.

En este script de python, podemos observar una serie de pasos para realizar el algoritmo de máquinas de vectores de soporte. En primer lugar, se cargan los datos de los datasets utilizados en el script y son entrenados mediante la función “svm.SVC()” para obtener el modelo requerido. Tras esto, el script llama a unas funciones plt cuya función es representar de manera gráfica los puntos de los patrones, al mismo tiempo cuáles de estos patrones serán los vectores de soporte (marcados con una cruceta) y el hiperplano que separa las dos nubes de patrones clasificados.

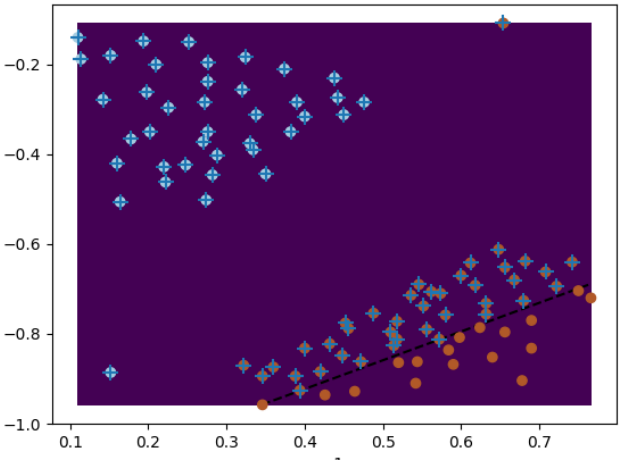
Pregunta [2]:
Intuitivamente, ¿qué hiperplano crees que incurrirá en un menor error de test en la tarea de separar las dos clases de puntos?.



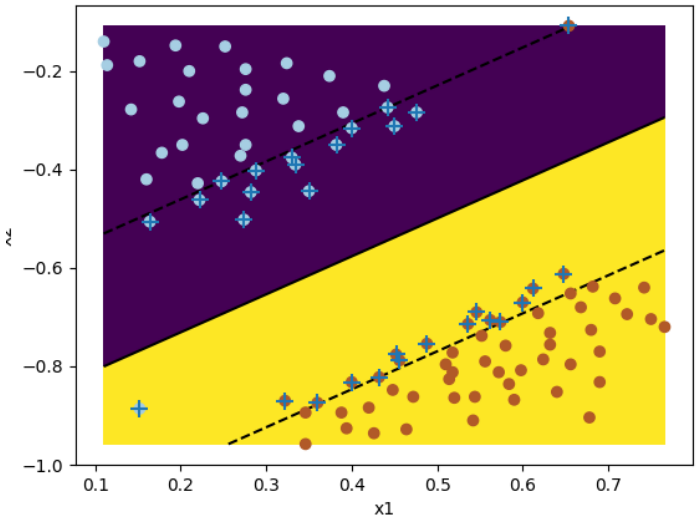
De manera intuitiva pienso que el hiperplano que otorgaría menor error de clasificación es aquel que separa las dos nubes de puntos y toma los dos patrones que están alejados de sus respectivas nubes de puntos como patrones mal clasificados.

Pregunta [3]:
Modifica el script probando varios valores de C, en concreto, C ∈ {10−2, 10−1, 100, 101, 102, 103, 104}. Observa qué sucede, explica por qué y escoge el valor más adecuado.

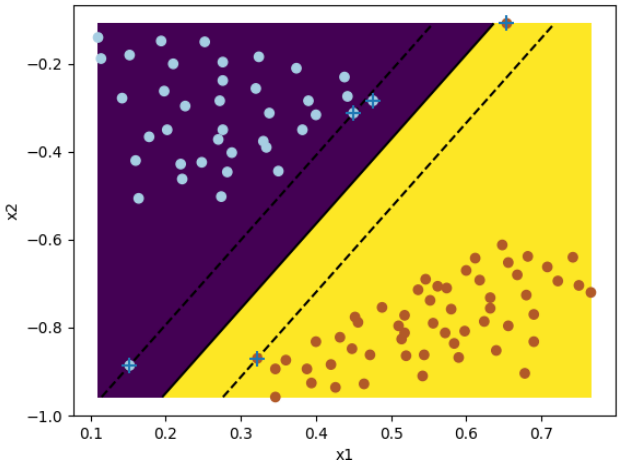
-0.01:



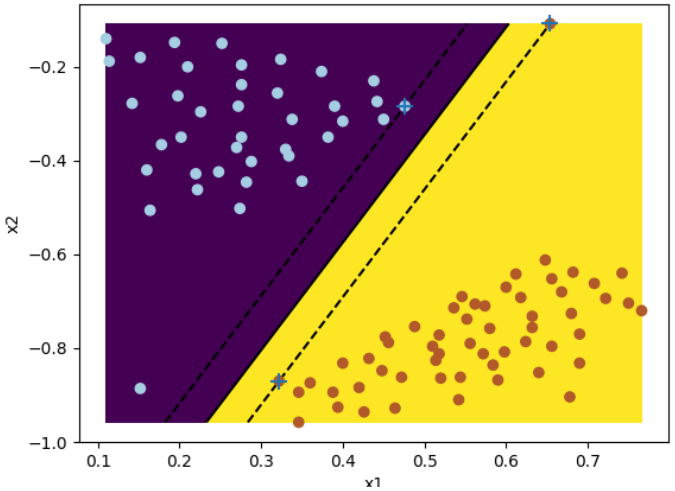
-1:



-100:



-10000:

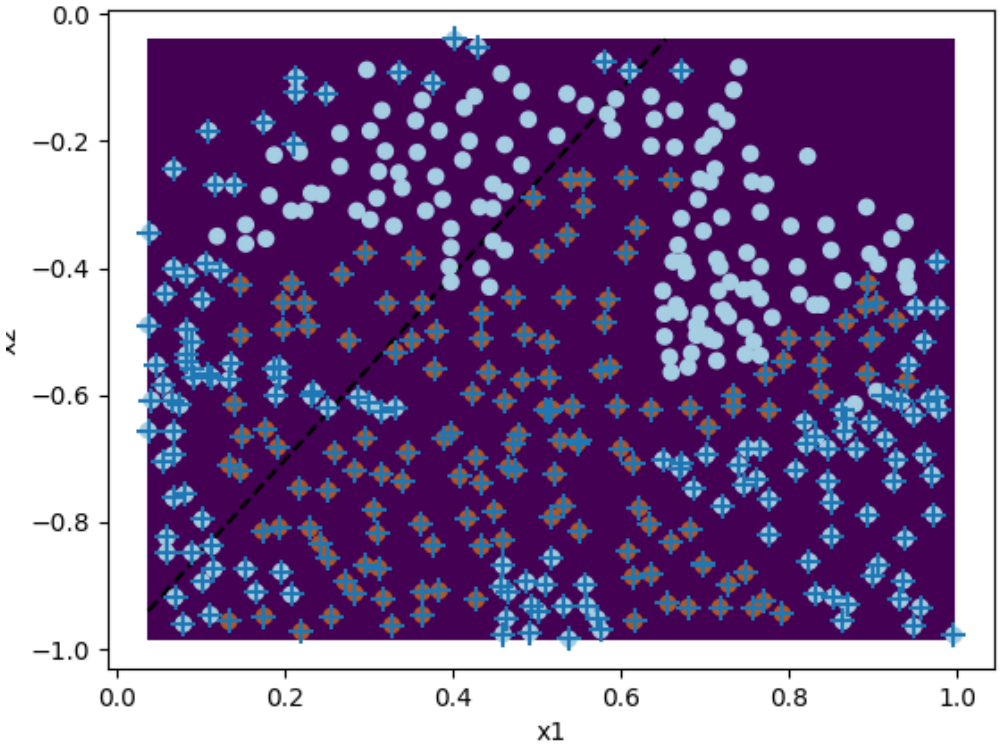


Como podemos observar en las gráficas, a menor coeficiente de entrenamiento, peores son los resultados obtenidos por el modelo.

Para este conjunto de valores de C, podemos observar que para el valor 10000 (valor más alto) es el modelo que mejores resultados genera sin llegar así a un sobreentrenamiento.

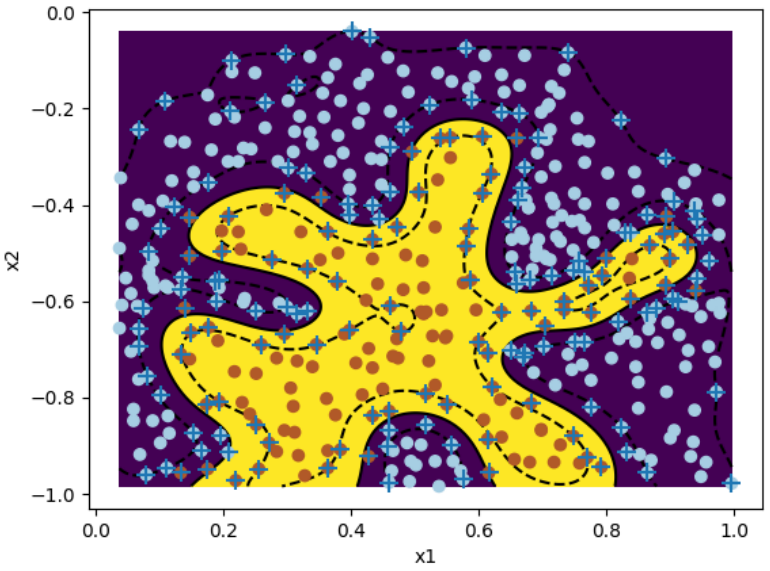
Pregunta [4]:
Prueba a lanzar una SVM lineal con los valores para C que se utilizaron en la pregunta anterior. ¿Consigues algún resultado satisfactorio en el sentido de que no haya errores en el conjunto de entrenamiento?. ¿Por qué?.

Para cualquier configuración de C utilizada, no se obtiene ningún resultado satisfactorio debido a que el conjunto de patrones de entrenamiento no puede ser separado linealmente debido a que la separación entre los dos conjuntos de patrones tiene forma estrellada, lo cual hace imposible su separación con una única línea.

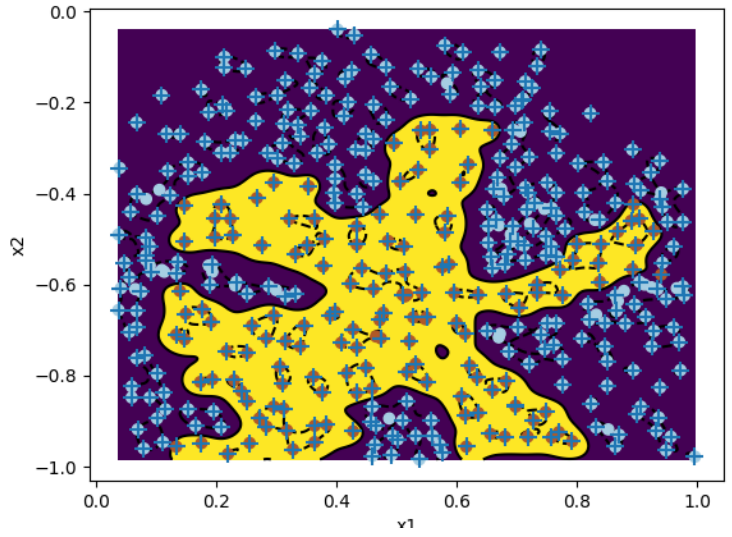


Pregunta [5]:
Propón una configuración de SVM no lineal (utilizando el kernel tipo RBF o Gaussiano) que resuelva el problema. El resultado debería ser similar al de la Figura 3. ¿Qué valores has considerado para C y para γ?. Además, incluye un ejemplo de una configuración de parámetros que produzca sobre-entrenamiento y otra que produzca infra-entrenamiento.

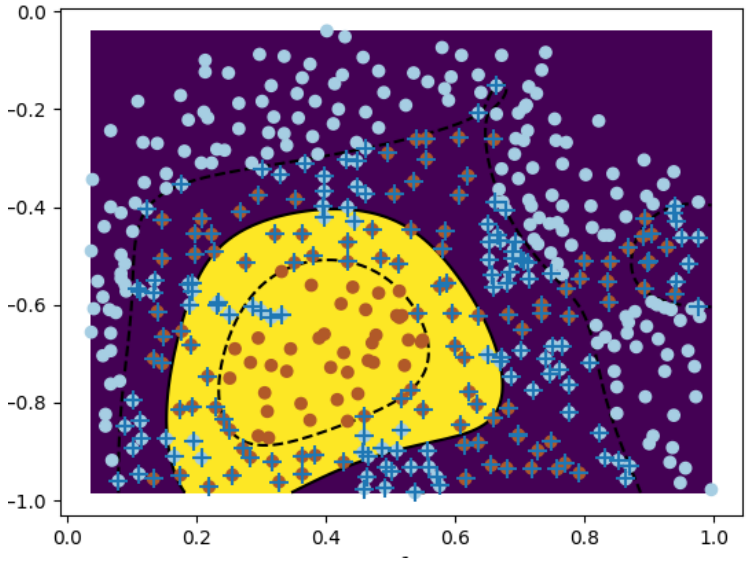
Para obtener un resultado satisfactorio, se utiliza la función “svm.SVC()” en modo “rbf” con los parámetros C=1 y gamma=100.



Con el parámetro gamma elevado, se tiende al sobreentrenamiento como se observa en la siguiente figura (gamma=10000):

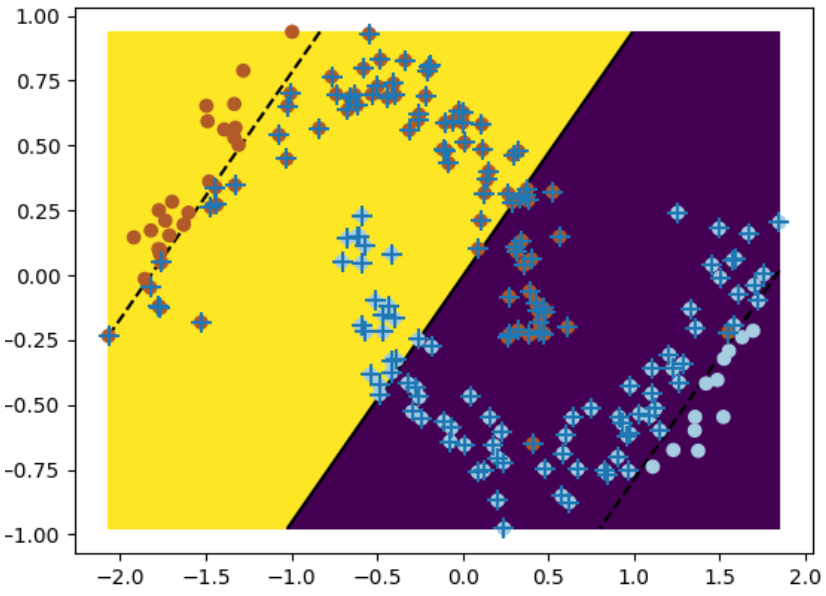


Mientras que con un gamma muy reducido se tiende al infra-entrenamiento como se aprecia en la siguiente figura (gamma=10):



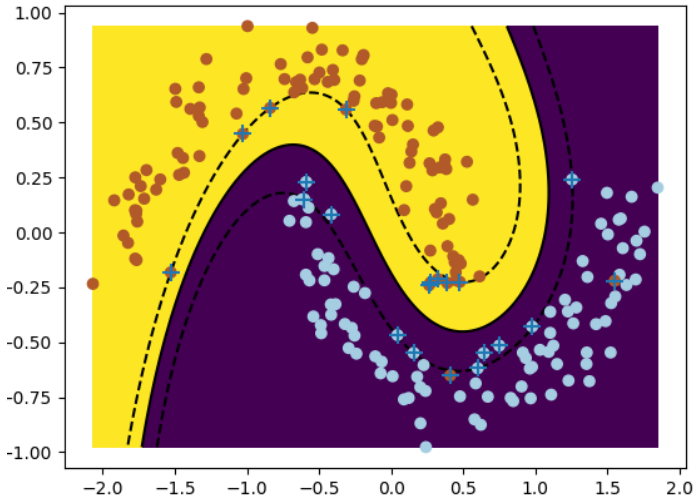
Pregunta [6]:
En este caso, ¿es el dataset linealmente separable?. A primera vista, ¿detectas puntos que presumiblemente sean outliers?, ¿por qué?

Este dataset no es linealmente separable debido a la forma circular que presentan las diferentes nubes de puntos.
No parecen existir puntos outliers debido a que todos los puntos están bien definidos en sus respectivas nubes de puntos y no hay patrones aislados.

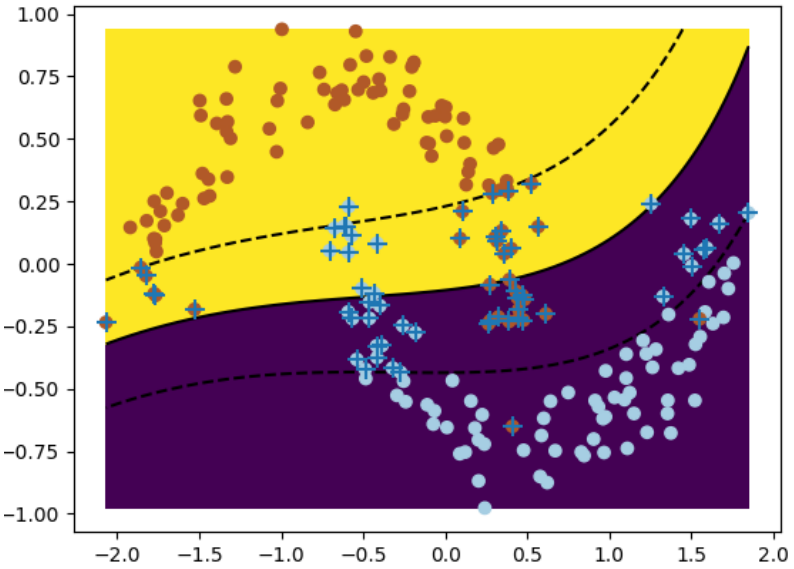


Pregunta [7]:
Lanza una SVM para clasificar los datos, con objeto de obtener un resultado lo más parecido al de la Figura 5. Ajusta el ancho del kernel en el intervalo γ ∈ {0,02, 0,2, 2, 200}. Ajusta el parámetro de coste en el intervalo C ∈ {0,02, 0,2, 2, 200}. Establece el valor de los parámetros óptimos. Además, incluye un ejemplo de una configuración de parámetros que produzca sobre-entrenamiento y otra que produzca infra-entrenamiento.

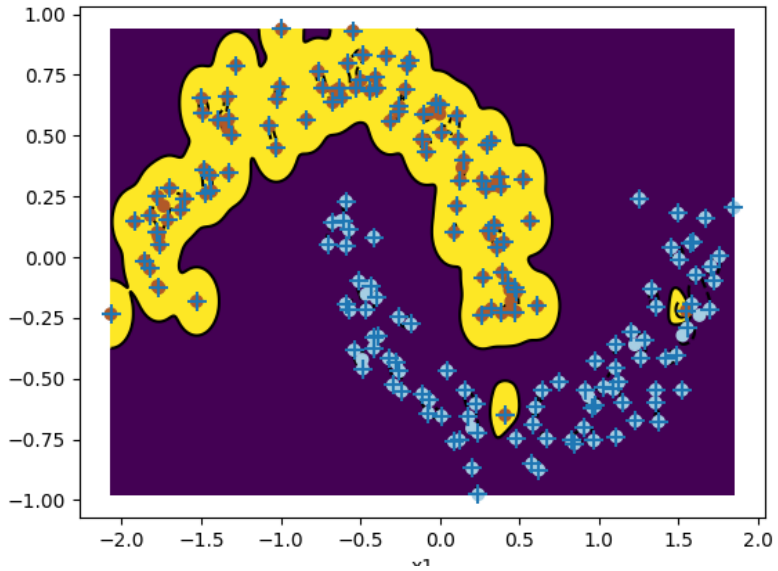
De manera similar al dataset 2, utilizaremos un modelo RBF con los parámetros C=200 y gamma=0.2 (mejor combinación de resultados obtenidos).



Infra-entrenamiento C=200, gamma=0,02:



Sobreentrenamiento C=200, gamma=200:

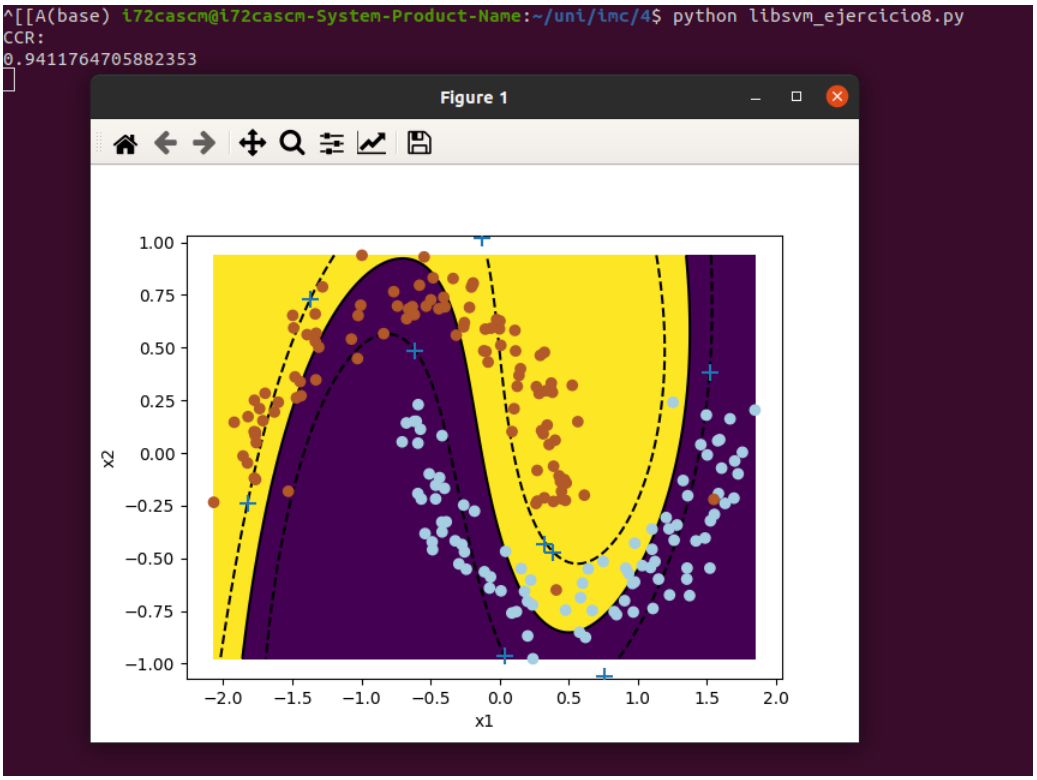


Pregunta [8]:
Vamos a reproducir este proceso en Python. Divide el dataset sintético dataset3.csv en dos subconjuntos aleatorios de forma estratificada, con un 75 % de patrones en train y un 25 % de patrones en test. Realiza el proceso de entrenamiento completo (estandarización, entrenamiento y predicción), 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.

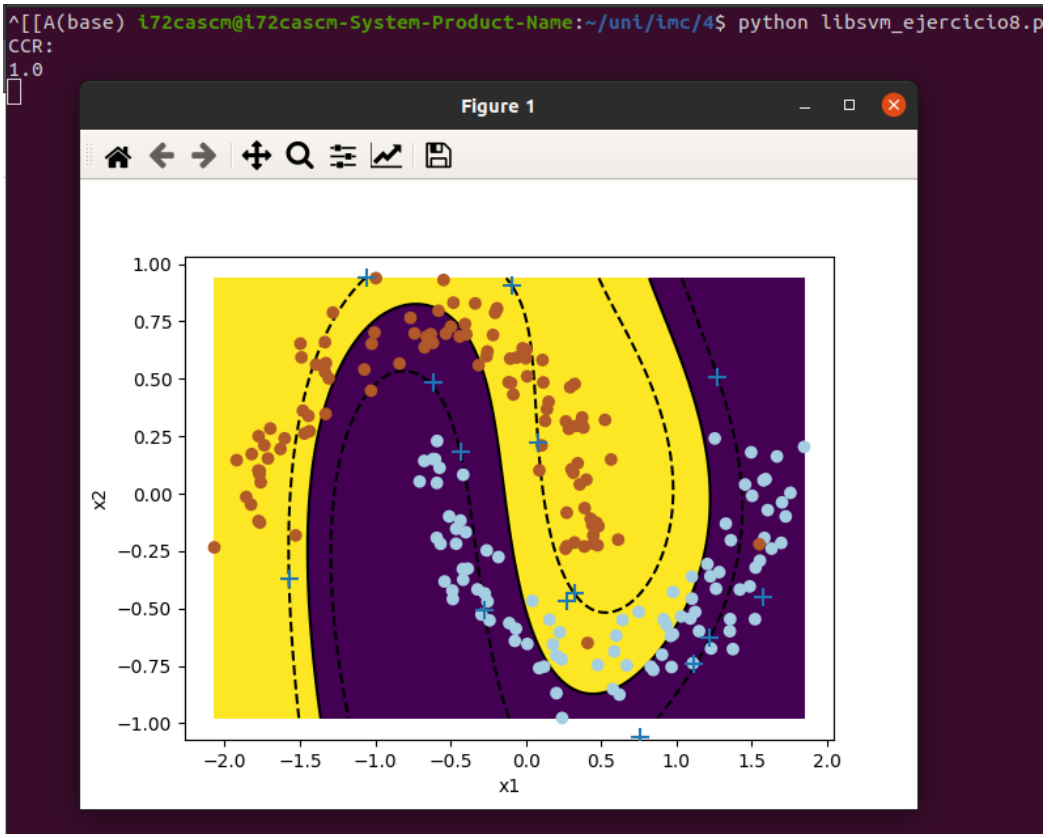
Para esta pregunta, utilizaremos el script “libsvm_ejercicio8.py”, donde se ha modificado el código para que el dataset se divida en dos subconjuntos aleatorios de manera estratificada con un 75% de los patrones en entrenamiento y el 25% de los patrones en test.

Los datos son procesados con diferentes semillas y su CCR es medido para observar las diferencias que existen entre ejecuciones. Todas las ejecuciones utilizan los parámetros C y gamma del ejercicio anterior (C=200, gamma=0.2):

Para semilla=1:



Para semilla=5:



Podemos observar que, dependiendo de la semilla el CCR varía debido a cómo se dividen en entrenamiento y test el conjunto de patrones. Además, podemos observar un valor alto de CCR, por lo que podemos concluir que el modelo clasifica de manera correcta, llegando a CCR=1 en ciertas semillas como es la segunda ejecución (randon_state= 5).

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.

El código se ha modificado en el script “libsvm_ejercicio9.py”, donde se implementa la búsqueda automática de los parámetros C y gamma mediante “GridSearch”.

Se utilizarán las mismas semillas que en el ejercicio anterior:
Para semilla=1:

```
(base) i72cascm@i72cascm-System-Product-Name:~/uni/imc/4$ python libsvm_ejercicio9.py
CCR:
1.0
(base) i72cascm@i72cascm-System-Product-Name:~/uni/imc/4$
```

Para semilla=5:

```
(base) i72cascm@i72cascm-System-Product-Name:~/uni/imc/4$ python libsvm_ejercicio9.py
CCR:
0.9803921568627451
```

Podemos observar que tanto para la semilla 1 como la 5 los resultados han mejorado, llegando a obtener en la semilla 5 un CRR de 100% (al igual que el ejercicio anterior) y en la primera semilla un CCR que mejora al del ejercicio anterior (0.94<0.98).

Pregunta [10]:
¿Qué inconvenientes observas en ajustar el valor de los parámetros "a mano", viendo el porcentaje de buena clasificación en el conjunto de test (lo que se hizo en la pregunta 8)?.

El principal inconveniente de introducir los valores de C y gamma a mano es la cantidad de tiempo empleada en buscar los valores más óptimos para estos parámetros y la posibilidad de quedarnos con valores que no sean tan óptimos como los generados automáticamente por este método.

Pregunta [11]:
Para estar seguros de que has entendido cómo se realiza la búsqueda de parámetros, implementa de forma manual (sin usar Grid Search CV) 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 Grid Search CV.

Para realizar este ejercicio, ejecutamos el script "libsvm_ejercicio11.py" donde se realiza el algoritmo GridSearchCV de manera manual.

Los resultados obtenidos de ejecutar el script una serie de veces es el siguiente:

```
^[[A^[[A(base) i72cascm@i72cascm-System-Product-Name:~/uni/imc/4$ python libsvm_ejercicio11
CCR:
0.9803921568627451
(base) i72cascm@i72cascm-System-Product-Name:~/uni/imc/4$ python libsvm_ejercicio11.py
CCR:
1.0
(base) i72cascm@i72cascm-System-Product-Name:~/uni/imc/4$ python libsvm_ejercicio11.py
CCR:
1.0
(base) i72cascm@i72cascm-System-Product-Name:~/uni/imc/4$ python libsvm_ejercicio11.py
CCR:
1.0
(base) i72cascm@i72cascm-System-Product-Name:~/uni/imc/4$ python libsvm_ejercicio11.py
CCR:
1.0
(base) i72cascm@i72cascm-System-Product-Name:~/uni/imc/4$ python libsvm_ejercicio11.py
CCR:
1.0
(base) i72cascm@i72cascm-System-Product-Name:~/uni/imc/4$ python libsvm_ejercicio11.py
CCR:
0.9607843137254902
(base) i72cascm@i72cascm-System-Product-Name:~/uni/imc/4$
```

Los resultados son similares a los del ejercicio número 9, por lo que podríamos concluir que el algoritmo funciona adecuadamente.

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.

Ejecutando el script "libsvm_ejercicio12.py" (modificación del script del ejercicio 9 preparado para leer los datos de la base de datos nomnist train/test) obtenemos el siguiente resultado:

```
(base) i72cascm@i72cascm-System-Product-Name:~/uni/imc/4$ python libsvm_ejercicio12.py
CCR:
0.8966666666666666
(base) i72cascm@i72cascm-System-Product-Name:~/uni/imc/4$ python libsvm_ejercicio12.py
CCR:
0.8966666666666666
(base) i72cascm@i72cascm-System-Product-Name:~/uni/imc/4$
```

Podemos observar que es un CCR bastante bueno pero es levemente peor al CCR obtenido por ejercicios anteriores lo cual podría deberse al nuevo conjunto de datos.

Pregunta [13]:
Localiza dónde 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.

Para k=3:

```
(base) nanopoke@nanopoke-System-Product-Name:~/uni/imc/4$ python libsvm_ejer
cicio12.py
CCR:
0.9
Tiempo empleado:
15.208694696426392
```

Para k=5:

```
(base) nanopoke@nanopoke-System-Product-Name:~/uni/imc/4$ python libsvm_ejer
cicio12.py
CCR:
0.8966666666666666
Tiempo empleado:
30.43335485458374
```

Para k=10:

```
(base) nanopoke@nanopoke-System-Product-Name:~/uni/imc/4$ python libsvm_ejer
cicio12.py
CCR:
0.92
Tiempo empleado:
81.89279389381409
```

Podemos observar que para este dataset, aumentar o disminuir el valor de k en estos rangos no supone una gran diferencia en el CCR obtenido, pero si podemos notar que para valores pequeños de k, el tiempo necesario para ejecutar el script es bastante menor con respecto al resto de valores de k, por lo que para este dataset es más eficiente emplear un valor pequeño de K (cv).

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.

Ejecutamos el script “libsvm_ejercicio14.py” para realizar este ejercicio:
-C=0.01:

```
(base) i72cascm@i72cascm-System-Product-Name:~/uni/imc/4$ python libsvm_ejercicio14.py
CCR:
0.98
```

-C=0.1:

```
CCR:
0.989
(base) i72cascm@i72cascm-System-Product-Name:~/uni/imc/4$
```

-C=1:

```
CCR:
0.978
(base) i72cascm@i72cascm-System-Product-Name:~/uni/imc/4$
```

-C=10:

```
CCR:
0.975
(base) i72cascm@i72cascm-System-Product-Name:~/uni/imc/4$
```

La mejor configuración es C=0.1 donde C corresponde al margen utilizado en el modelo SVM.

Pregunta [15]:
Para la mejor configuración, construye la matriz de confusión y establece cuáles 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 x_i es igual a 1 quiere decir que la palabra i- ésima del vocabulario aparece, al menos una vez, en el correo.

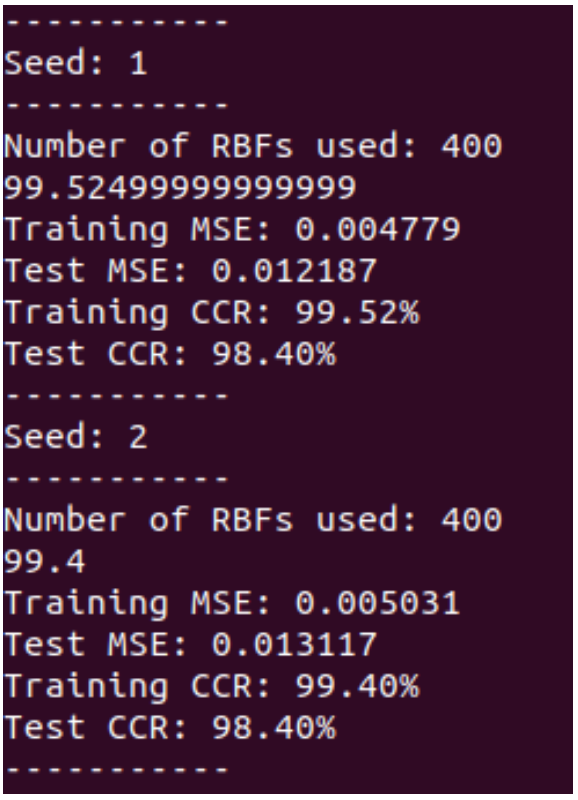
Utilizando el scrpit “libsvm_ejercicio15”, y utilizando la mejor configuración encontrada para el ejercicio anterior (C = 0.1), entrenamos un modelo lineal y predecimos las salidas del test para posteriormente emplearlo en la matriz de confusión.

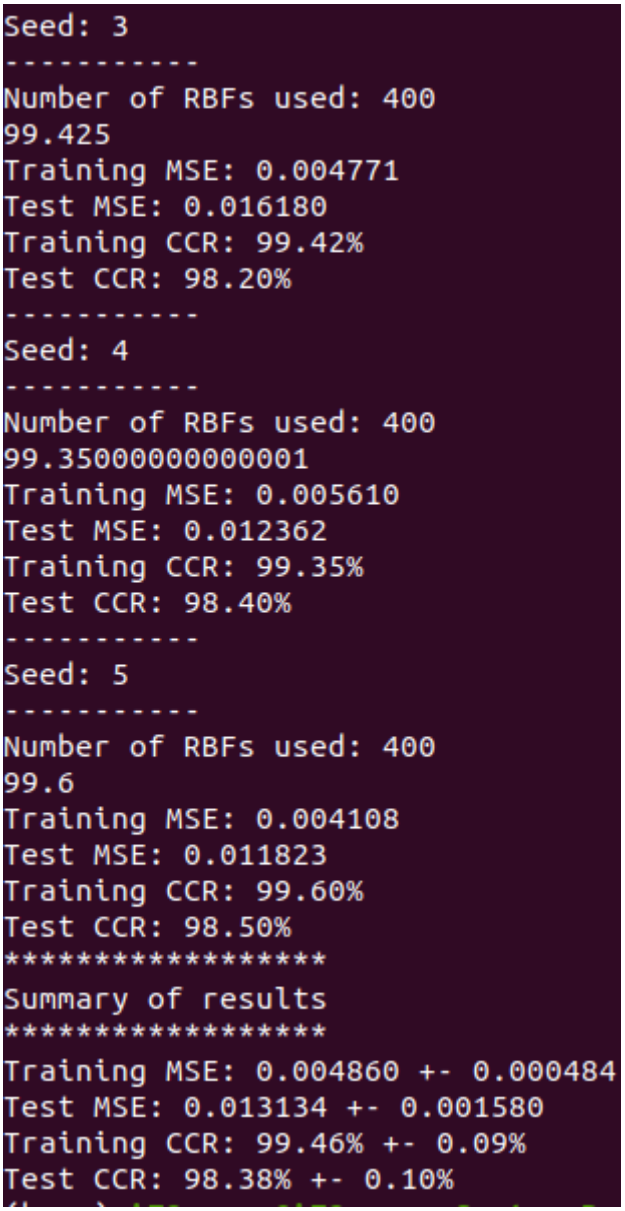
A continuación se muestra aquellos correos donde no se predice bien la clasificación:

```
(base) nanopoke@nanopoke-System-Product-Name:~/uni/imc/4$ python libsvm_ejercicio15.py
Position [ 9 ]: Predicción = [ 0 ] Valor real = [ 1 ]
Position [ 21 ]: Predicción = [ 1 ] Valor real = [ 0 ]
Position [ 58 ]: Predicción = [ 1 ] Valor real = [ 0 ]
Position [ 73 ]: Predicción = [ 1 ] Valor real = [ 0 ]
Position [ 147 ]: Predicción = [ 1 ] Valor real = [ 0 ]
Position [ 328 ]: Predicción = [ 1 ] Valor real = [ 0 ]
Position [ 407 ]: Predicción = [ 0 ] Valor real = [ 1 ]
Position [ 526 ]: Predicción = [ 1 ] Valor real = [ 0 ]
Position [ 560 ]: Predicción = [ 1 ] Valor real = [ 0 ]
Position [ 842 ]: Predicción = [ 1 ] Valor real = [ 0 ]
Position [ 881 ]: Predicción = [ 0 ] Valor real = [ 1 ]
```

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).

Ejecución con el script de la práctica 3:





Para todas las semillas, podemos observar un CCR cercano al 100%, por lo que podemos concluir que utilizando una red RBF para este dataset podemos clasificar los datos correctamente con muy pocos errores.

Pregunta [17]:
Entrenar una SVM no lineal y compara los resultados:

Utilizando un modelo SVM (“libsvm_ejercicio17.py”) no lineal con el valor de c ajustado a 0.1 (mejor valor de C para los ejercicios anteriores), obtenemos el siguiente CCR:

```
(base) nanopoke@nanopoke-System-Product-Name:~/uni/imc/4$ python libsvm_ejer
cicio17.py
CCR:
0.94
```

Podemos apreciar que es un CCR muy bueno, pero no consigue tan buenos resultados como la red RBF utilizada en la práctica anterior, la cual tiene CCR muy cercanos al 100%.

Bibliografía

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html