Universidad de Córdoba

Escuela Politécnica Superior de Córdoba
Grado en Ingeniería informática

Práctica 4. Máquinas de Vectores Soporte

Introducción a los modelos computacionales
Cuarto curso de Computación

Por: Francisco Javier Pérez Castillo
Correo: i72pecaf@uco.es
DNI: 32734629V

Índice de la memoria

1. Representación 2D de las SVMs

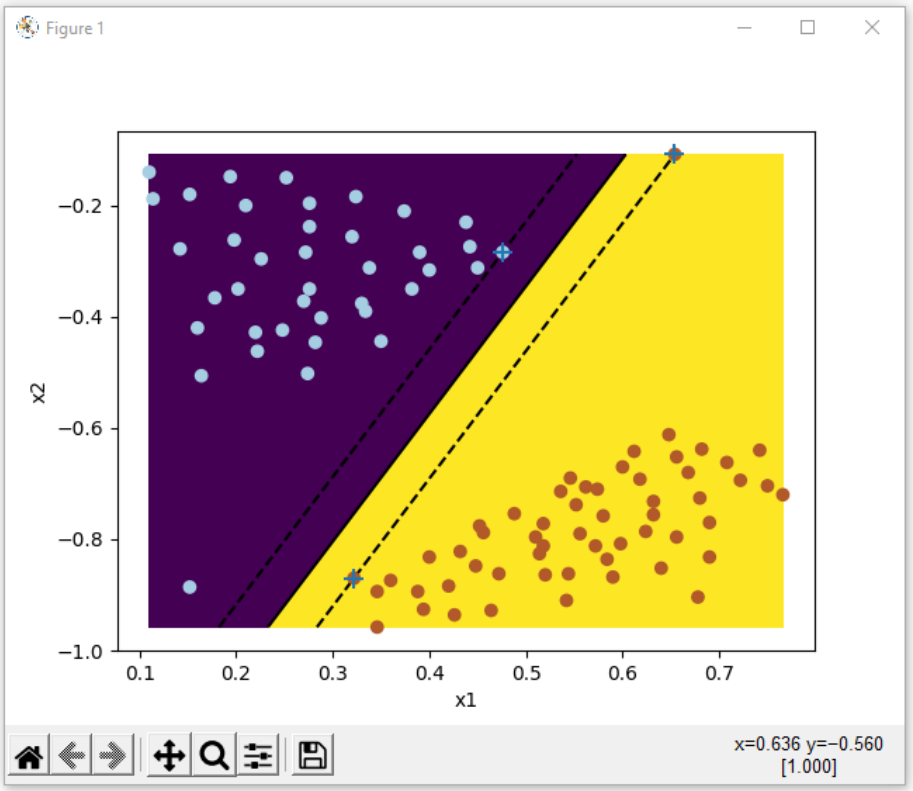
1.1 Pregunta [1]

El script utilizado comienza cargando el primer dataset de nuestro conjunto y guardando las dos primeras columnas en un variable, como características de nuestros patrones, y en otra variable las salidas de los mismos. Seguido de ello, se comienza a entrenar la máquina vector soporte utilizando para ello un kernel lineal:

$$K(\mathbf{x}_i,\mathbf{x}_j)= <\mathbf{x}_i,\mathbf{x}_j>$$

También se indica el parámetro *C*, conocido como parámetro de regularización, el cual actúa de forma inversa al término de regularización, es decir, cuanto más grande sea C, menor regularización y por tanto menor error, en caso contrario, cuanto menor es C, mayor regularización y mayor error.

Tras esto, el script continúa realizando operaciones para representar gráficamente el modelo anterior, usando para ello métodos de la librería *Matplot*. Si ejecutamos el script, se generará una gráfica representando el modelo anterior una vez entrenado.



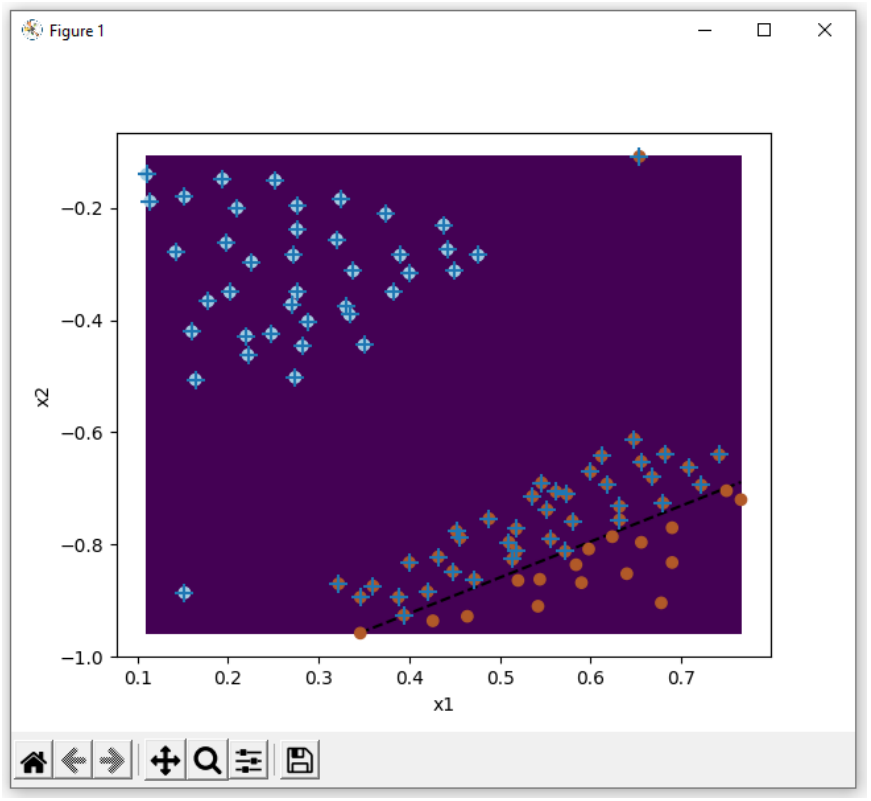
2. Primer dataset de ejemplo

2.1 Pregunta [2]

Hay varias formas razonables posibles, una de ellas es permitir trazar una recta pero tomando como mal clasificado el patron amarillo superior que está cercano a la nube de puntos azules, tambien tomar como mal clasificado al patron azul de la esquina inferiro izquierda. Esta idea resultaría en trazar una recta que permite un máximo margen pero teniendo en cuenta que se han cometido ciertos errores. Otra opción es tomar como vectores soporte dos puntos amarillos, siendo uno de ellos el punto cercano a la nube de puntos azul, junto con otro vector soporte de la clase azul, permitiendo trazar un hiperplano que divida bien el espacio sin suponer que se han clasificado erroneamente algunos patrones.

2.2 Pregunta [3]

Se mostrarán todas las gráficas para los valores indicados en el enunciado. La primera gráfica es el valor más bajo de C y la última el más alto, cada gráfica va incrementando el valor anterior de C siguiendo el mismo orden que el enunciado.



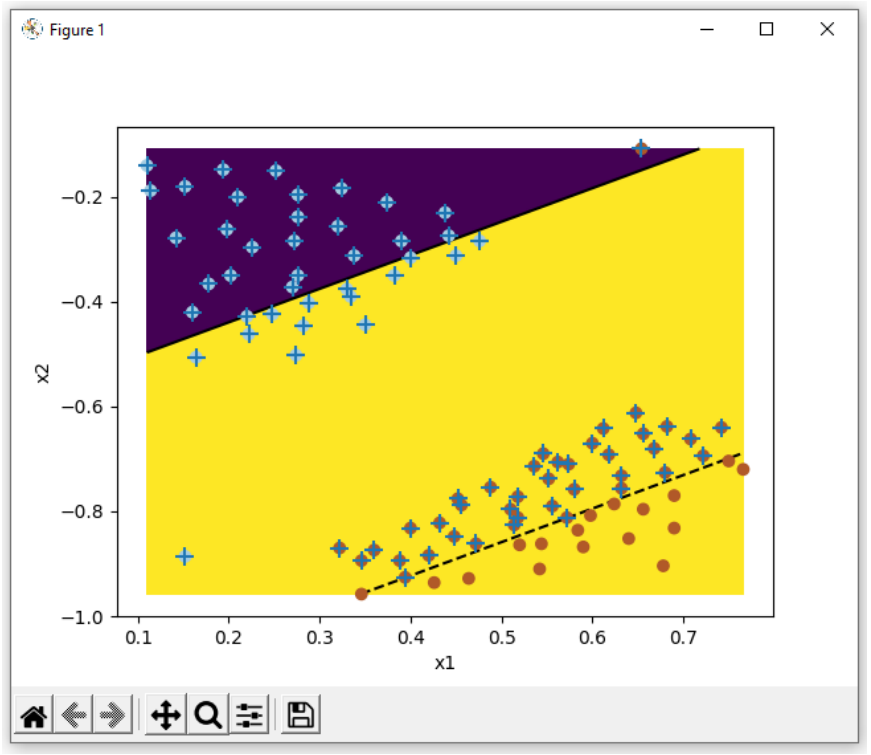


Figure 1
−0.2
−0.4
−0.6
−0.8
−1.0
x2
0.1
0.2
0.3
0.4
0.5
0.6
0.7
x1

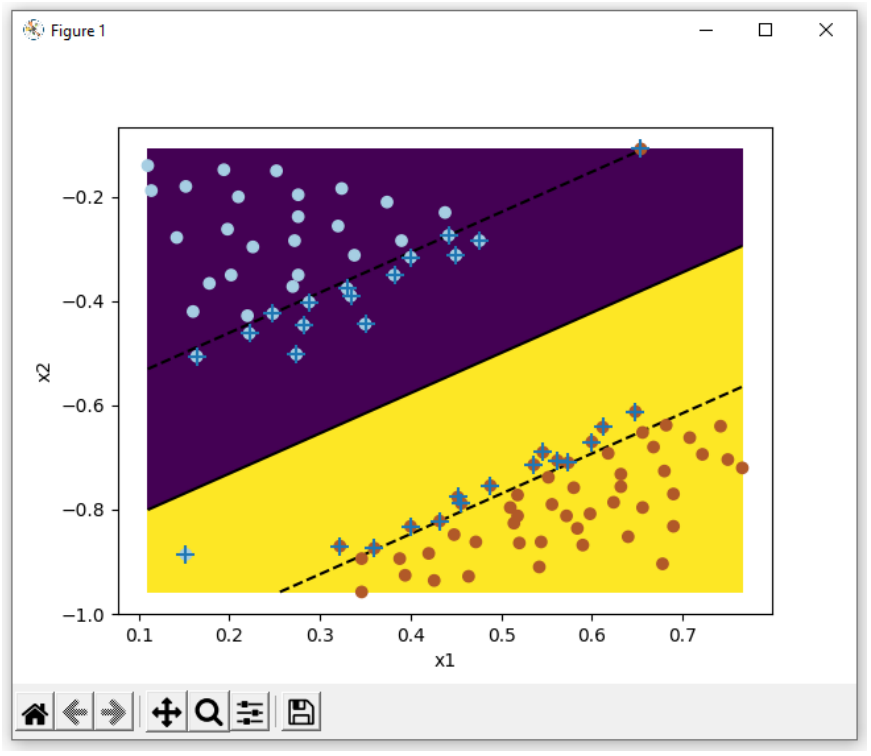


Figure 1
−0.2
−0.4
−0.6
−0.8
−1.0
x2
0.1
0.2
0.3
0.4
0.5
0.6
0.7
x1

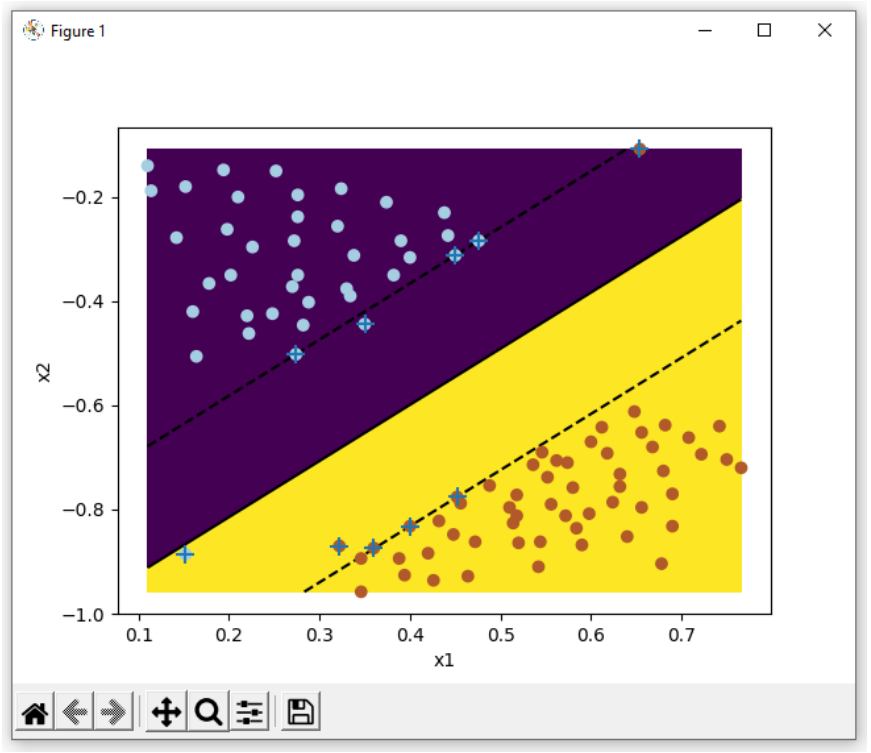


Figure 1
−0.2
−0.4
−0.6
−0.8
−1.0
x2
0.1
0.2
0.3
0.4
0.5
0.6
0.7
x1

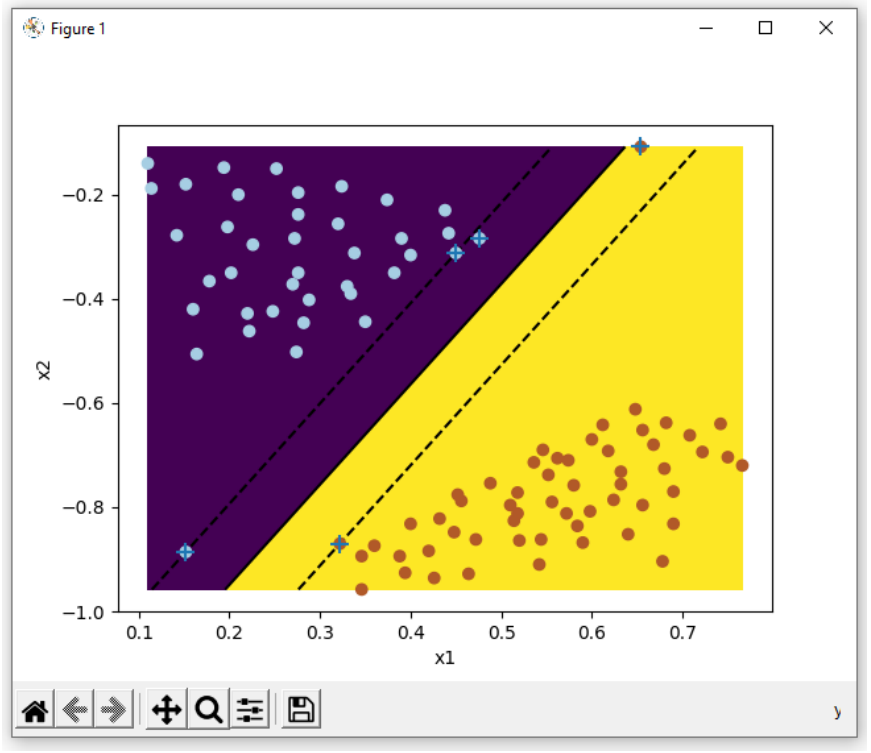


Figure 1
−0.2
−0.4
−0.6
−0.8
−1.0
x2
0.1
0.2
0.3
0.4
0.5
0.6
0.7
x1

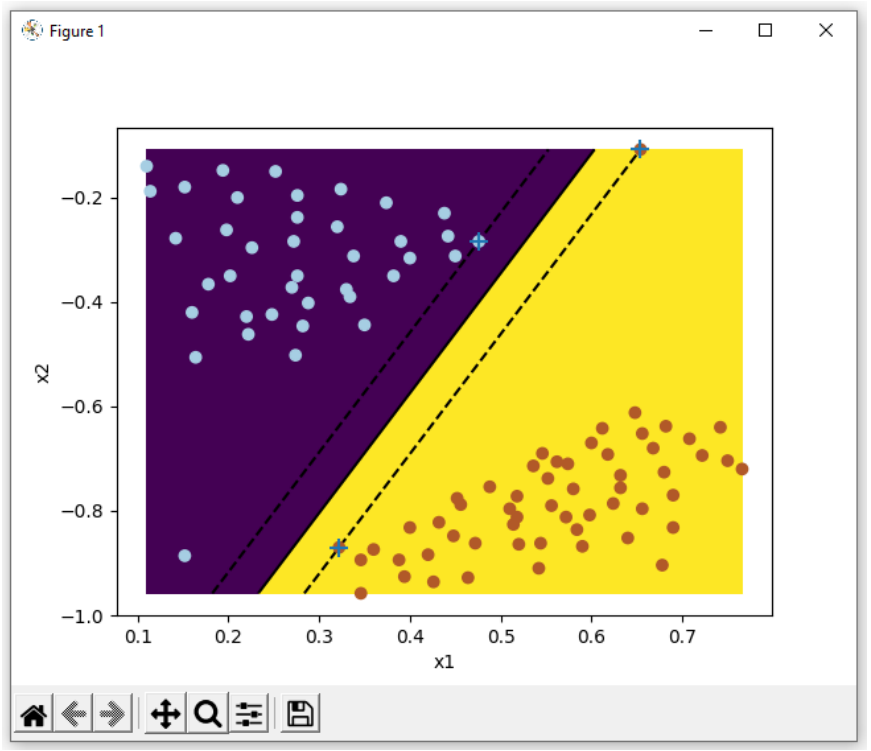


Figure 1
−0.2
−0.4
−0.6
−0.8
−1.0
x2
0.1
0.2
0.3
0.4
0.5
0.6
0.7
x1

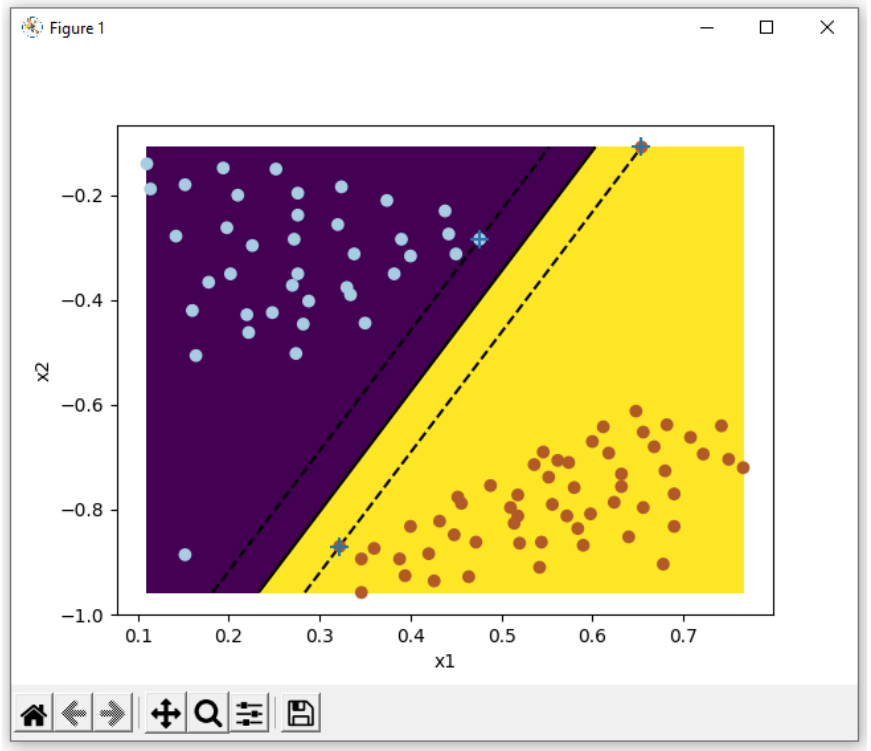


Figure 1
−0.2
−0.4
−0.6
−0.8
−1.0
x2
0.1
0.2
0.3
0.4
0.5
0.6
0.7
x1

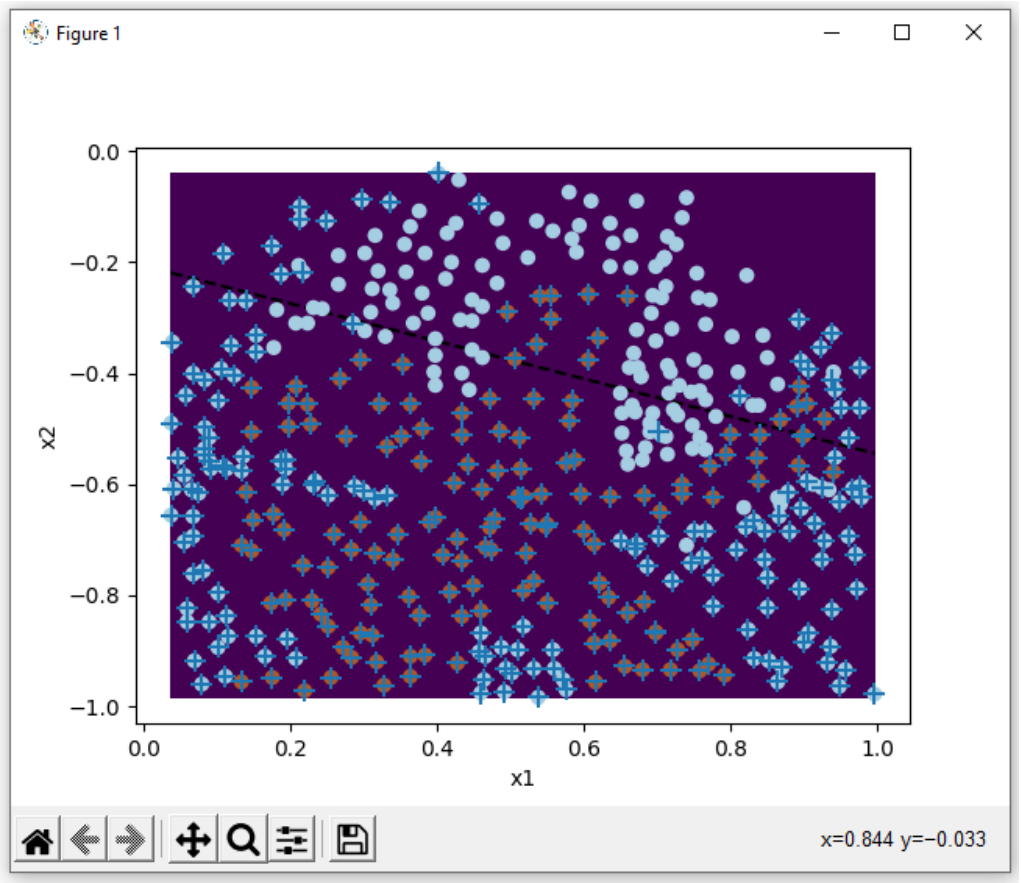
A medida que se va aumentando el valor *C* el número de patrones que se encuentran dentro del área creada por los márgenes disminuye, esto es, porque a medida que aumenta *C*, se va penalizando cada vez más los errores, esto deriva en que se vaya clasificando cada vez mejor. Valores de *C* muy pequeños permiten obtener un margen mayor a costa de clasificar mal algunos patrones. Si aumentamos enormemente C, podría producirse sobreentrenamiento. Por estos motivos, el valor de *C* depende enormemente de los datos con los que estemos trabajando, y en este caso, creo que un valor de $C = 10^{-2}$ (5º gráfico) permite obtener una correcta separación entre los datos sin la necesidad de presentar errores en los patrones ni derivar en un posible sobreentrenamiento.

3. Segundo dataset de ejemplo

3.1 Pregunta [4]

En este punto, utilizaremos el segundo dataset. Si lanzamos el script para este nuevo dataset, independientemente del valor de *C*, obtendremos unos malos resultados. El motivo de esto es que la distribución de los datos de este dataset impide que los datos puedan ser separados linealmente. Si intentamos utilizar el mismo modelo anterior obtendremos que la mayoría de puntos se van a una misma clase.

El resultado obtenido usando el modelo anterior para un $C = 10^{-2}$ es el siguiente:

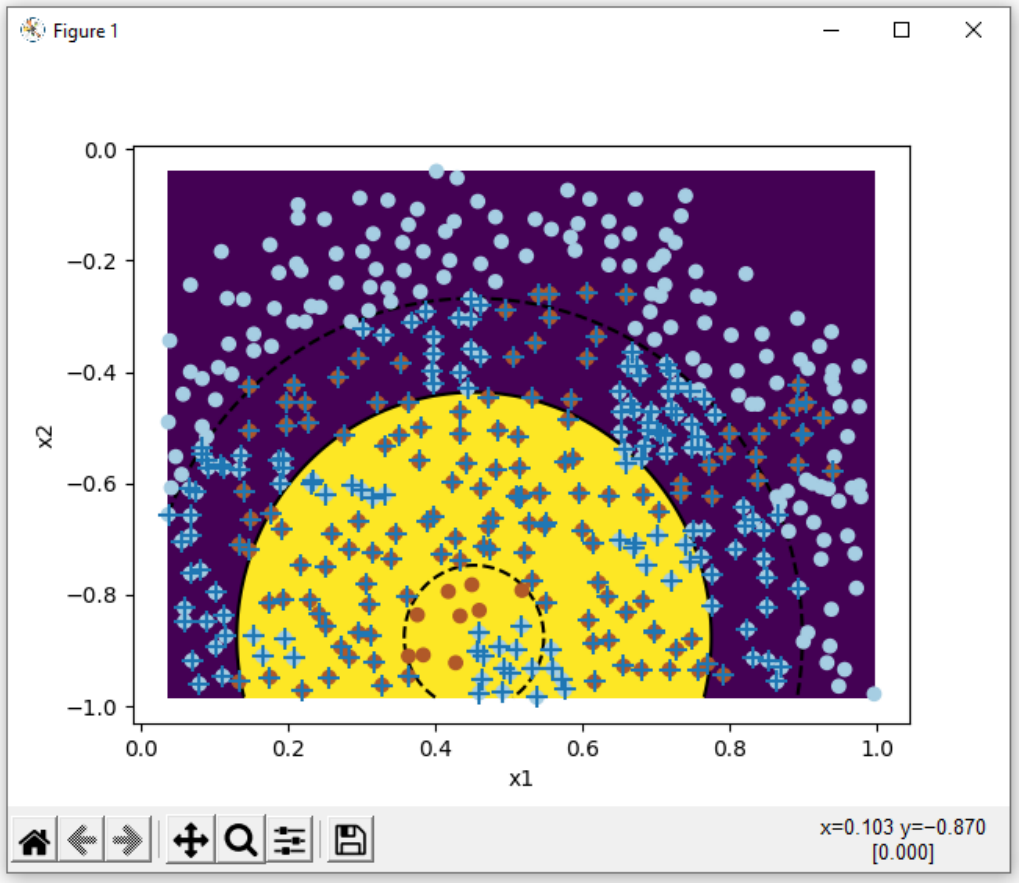


Como se puede ver, los datos no pueden ser separados correctamente por una línea.

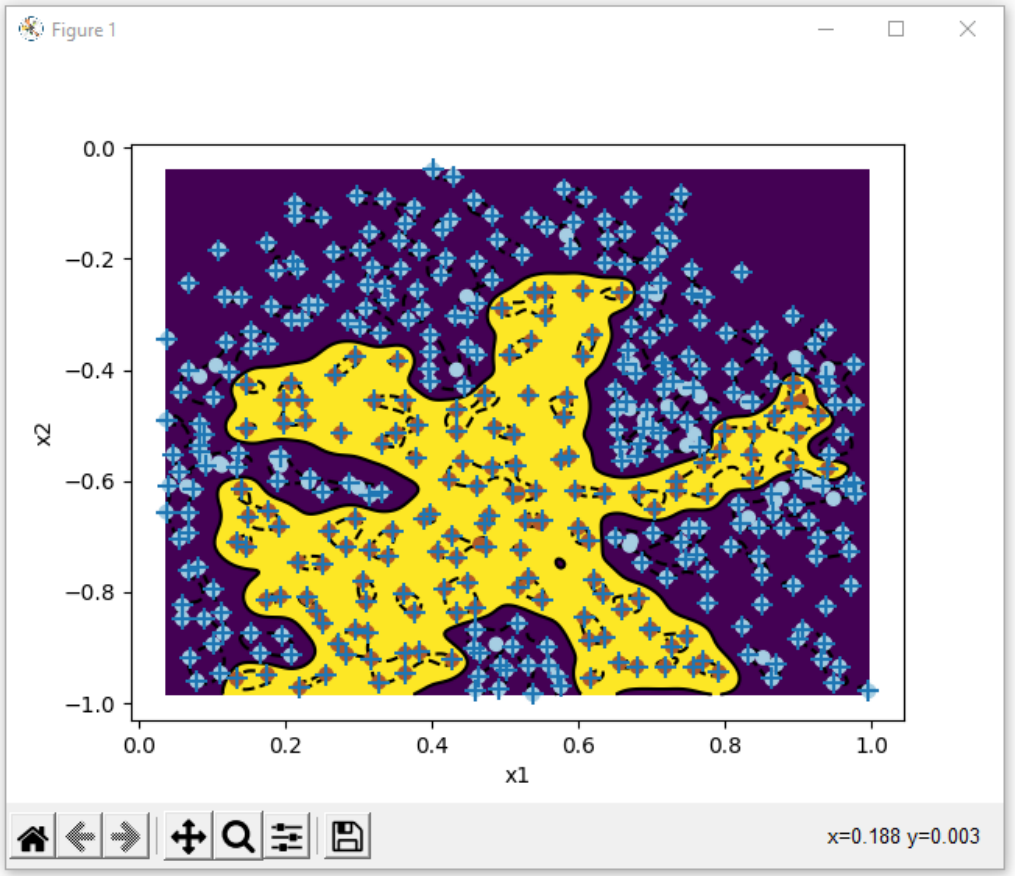
3.2 Pregunta [5]

Para solventar el problema anterior, debemos de modificar el kernel utilizado. Hasta ahora hemos usado un kernel lineal para separar el espacio en el que se encuentran nuestros datos, pero ahora utilizaremos distintas funciones kernel que, a través del *Truco del Kernel*, se podrá obtener una separación correcta de los datos.

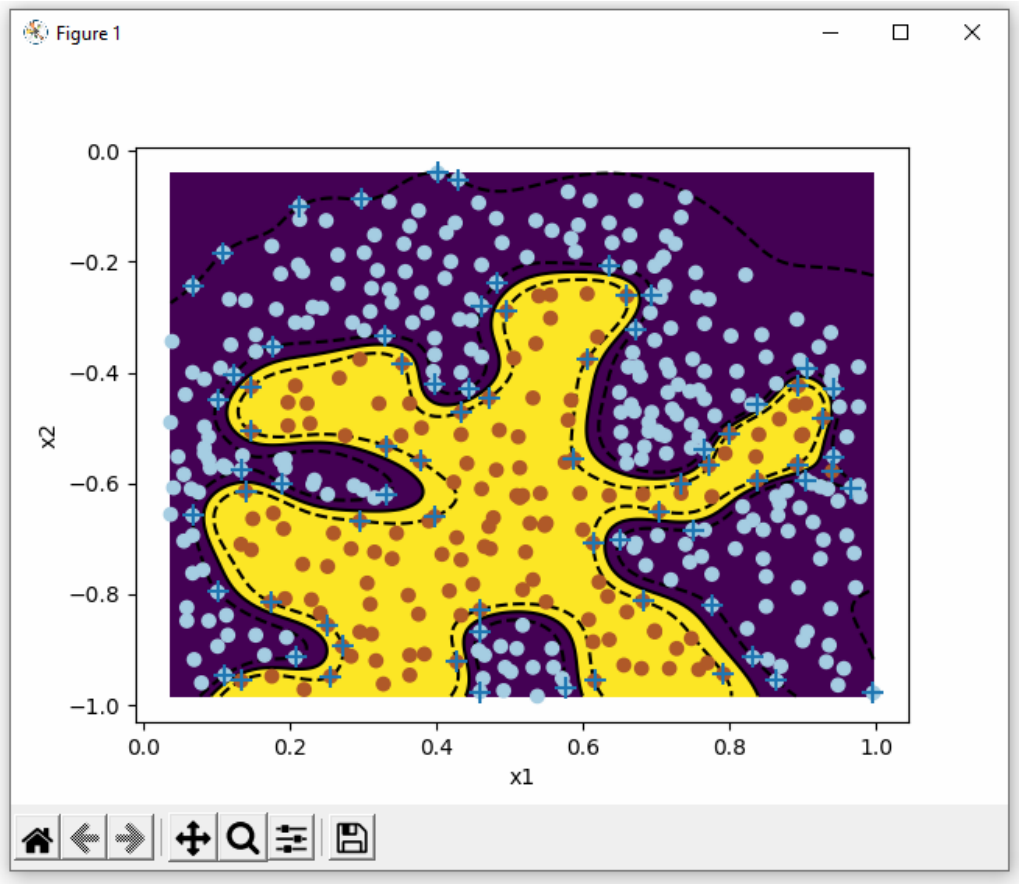
En nuestro script, esto se obtiene modificando el parámetro que recibe el método *svm.SVC*, cambiando de "*lineal*" a "*rbf*". Además, hay que añadir el parámetro γ (gamma), el cual según la documentación de este método, es la inversa del radio de la función gaussiana. Si para este parámetro tomamos valores muy grandes, tendemos a sufrir sobreentrenamiento, mientras que valores más bajos, resultará en soluciones más "suaves".



La foto anterior es un caso donde establecemos $C = 10^3$ y un gamma muy pequeño, de un 0,01, esto deriva en un modelo que clasifica muy mal, presenta un modelo que apenas delimita bien las clases. Si aumentamos el gamma drásticamente obtendremos el siguiente resultado, el cual presenta sobreentrenamiento. Esto se puede ver por la forma en la que traza los márgenes, además de los patrones utilizados.



Tras diversas pruebas, he tomado una $C = 10^3$ y un valor de gamma de 100. El resultado obtenido es el siguiente y creo que presenta una buena clasificación, ajustandose el espacio amarillo al mismo espacio que el de la *Figura 3* del guión de la práctica.



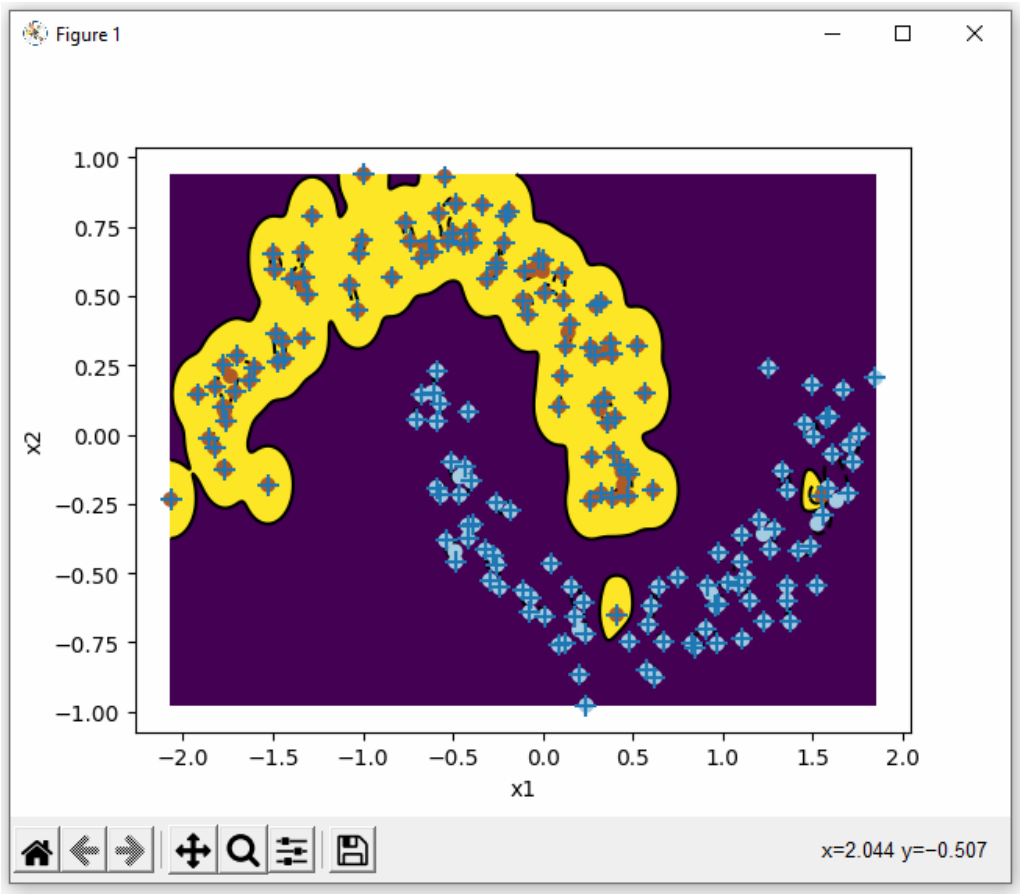
4. Tercer dataset de ejemplo

4.1 Pregunta [6]

No, el dataset no es posible que sea separado linealmente a través de un kernel lineal. En las gráficas de las preguntas siguientes se puede observar como hay dos patrones de la clase naranja que se encuentran en las nubes de puntos azules, estos son los *outliers*, dado que representan una clase distinta a la que debería de representar en base a sus valores.

4.2 Pregunta [7]

Ahora, se aplicará un kernel RBF a este datasat probando diferentes valores en los intervalos indicados. Primero se mostrará un ejemplo de sobreentrenamiento, seguido de un ejemplo de infra entrenamiento y finalmente la opción que más se asemeje a la dada en el guión de la práctica. El modelo que mejor se ajusta es $C = 20$ y gamma con valor 2.



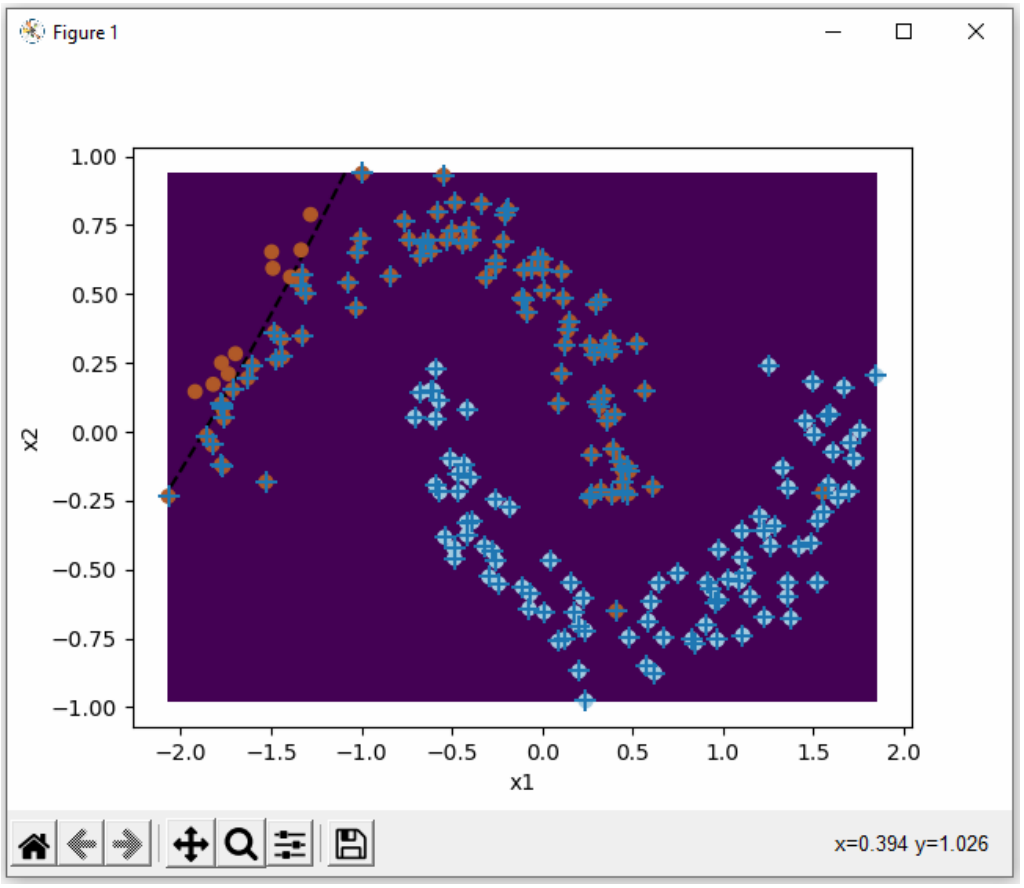


Figure 1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
x2
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x1
x=0.394 y=1.026

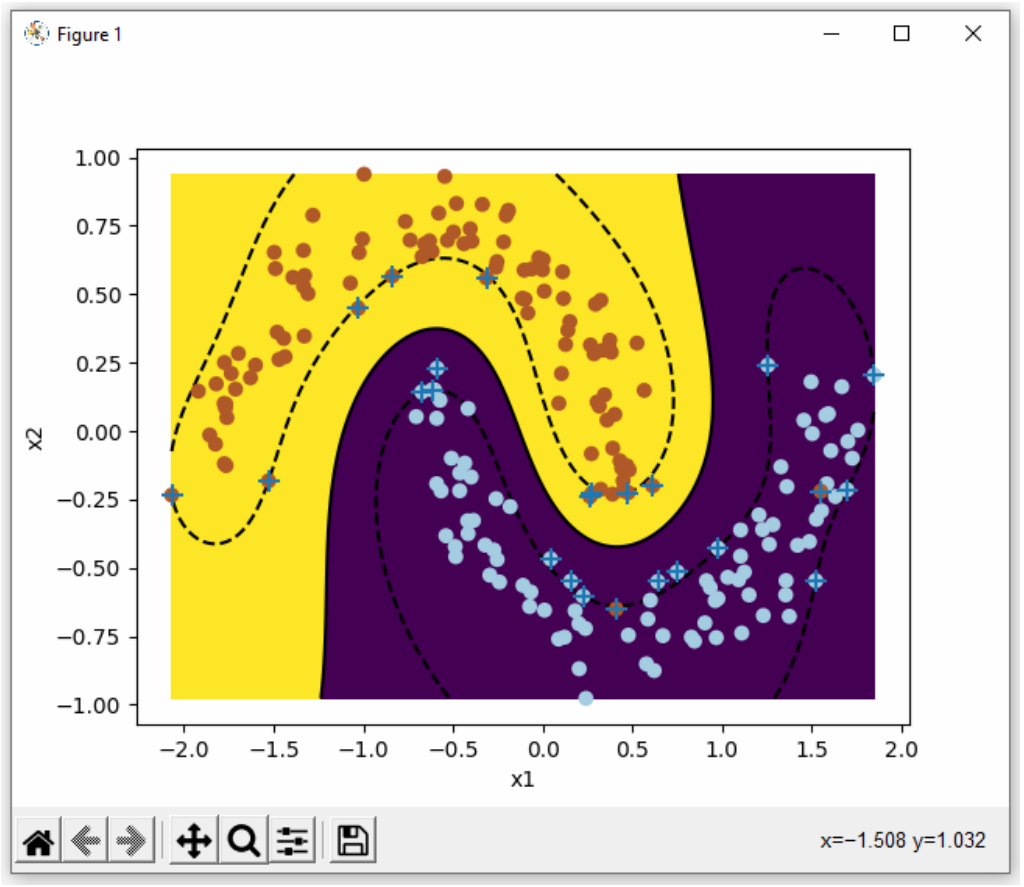


Figure 1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
x2
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x1
x=−1.508 y=1.032

5. Interfaz de consola

5.1 Pregunta [8]

Para realizar este ejercicio, comenzamos estandarizando los datos de entrada, para ello utilizamos la clase *StandardScaler*, según el ejemplo del guión. La partición de los datos se lleva a cabo con la clase *StratifiedShuffleSplit*. Estos métodos han sido implementados en el script python, junto con el cálculo de la puntuación CCR para evaluar los resultados.

Si ejecutamos el script utilizando para este caso C = 100 y gamma con valor 0.2, podemos jugar con la semilla para comprobar si hay variación en el CCR. En este caso, para una semilla con valor 1 obtenemos una puntuación de *0.9411*, y sí modificamos la semilla a 300 por ejemplo, obtenemos un valor de CCR del *0.9803*. Si volvemos a darle repetidamente valores distintos, observaremos que de nuevo tanto el CCR como las representaciones gráficas cambian, por tanto, podemos concluir que la semilla sí influye en los resultados.

5.1 Pregunta [9]

Ahora tenemos como objetivo encontrar los parámetros C y gamma de forma óptima sin tener que cambiar manualmente los valores. Este objetivo se logra alcanzar utilizando una partición de los datos llamada K-fold, obteniendo subconjuntos de datos los cuales probaremos con distintos valores del parámetros K, que determina el número de entrenamientos, obteniendo así diferentes entrenamientos con los que sacaremos los valores los hiperparámetros de nuestro modelo.

En nuestro script, podemos utilizar la clase *GridSearchCV* para ello. En este caso, según el ejemplo del guión, aplicamos un 5-fold.

Una vez completado el código, vamos probando con distintas semillas y observamos que se obtienen mejores resultados.

Tras la ejecución del script con *GridSearchCV* se obtuvo un 0.97 para RandomState con valor 1, un resultado de 1 para RandomState 5 y también una salida de 1 en el CCR para RandomState igual a 10.

5.1 Pregunta [10]

El método de K-fold permite obtener los mejores valores posibles para nuestro hiperparámetros, por lo que este método permite obtener los mejores resultados posibles para nuestro modelo, además de obtener dichos valores de forma mucho más rápida que sí se fueran obteniendo a mano.

5.1 Pregunta [11]

Para la realización de este ejercicio se ha utilizado un script diferente del original (*libsvm_11.py)* donde se encuentra escrito el algoritmo para obtener los hiperparámetros *C* y *gamma* de nuestro modelo. Para ello, se utilizó el método *StratifiedKFold* para poder repartir los datos de forma equitativa, para poder así probar una máquina vector soporte con diferentes hiper parámetros. Se ha utilizado el CCR para reconocer cual de todos los valores probados es el mejor para ser usado.

El script utilizado da un resultado de valor 1 para el CCR, C con valor 0.5 y Gamma con valor 1.681792830507429. Obtengo el mismo valor de CCR para el script que utiliza el método GridSearchCV.

6. Bases de datos reales

6.1 Pregunta [12]

Como se indica en el enunciado, se debe de ejecutar el script del ejercicio 9 y se debe de observar los resultados. El resultado ha sido un CCR del 0.89333, resultado un poco inferior al resultado obtenido en la pregunta 9 para valor 1 en el RandomState.

Para este script, se ha modificado un poco la lectura de los datos, dado que los datos de entrenamiento y test se encuentran en ficheros distintos, por ello, se ha creado un script con nombre diferente al usado actualmente, el nombre es *libsvm_12.py*.

6.2 Pregunta [13]

El valor de K permite reducir el tiempo computacional usado, ya que se le puede decir al programa, en este caso al GridSearch, el número de particiones que se desea realizar. Para ello, tenemos que indicarlo en el parámetro *cv*, ahí será donde indiquemos los valores 3, 5 y 10, correspondientes al número de particiones, como ya se ha mencionado anteriormente. Usaremos la clase *time* para obtener el tiempo que se tarda en ejecutar todo el programa para cada una de las particiones. Como ya se ha dicho, variando el parámetro K podemos reducir el tiempo de cómputo, pero la idea de tener un buen K es encontrar un punto de encuentro entre un CCR y un tiempo de cómputo decentes. Los resultados son los siguientes:

Valor K	CCR	Tiempo
3	0.9033	20.84563446044922
5	0.8933	46.44850540161133
10	0.9133	134.67156147956848

6.3 Pregunta [14]

Nuevamente, se ha creado un script llamado *libsvm.py* el cual permitirá realizar el cálculo del CCR para el dataset *Spam*, usando para ellos distintos C. Se ha formado una tabla con todos los valores.

Valor C	CCR
0.01	0.98
0.1	0.989
1	0.978
10	0.975

Según los resultados, todos los valores de C presentan resultados similares, pero el que mejor resultados da es el valor 0.1 para C, dado que presenta el CCR más alto.

6.4 Pregunta [15]

Se usará el script libsvm_15.py para la realización de este ejercicio. Para solventar nuestro problema. Para ello, obtenemos los datos predecidos respecto al conjunto de test y calculamos la matriz de confusión. Tras esto, recorremos la misma y vemos donde no coinciden los valores del conjunto precedido y del conjunto de test. La salida de esto junto con el CCR la siguiente:

```
CCR:  0.989
Different values in position [ 9 ]. Predicted value is [ 0 ], but the real value is [ 1 ]
Different values in position [ 21 ]. Predicted value is [ 1 ], but the real value is [ 0 ]
Different values in position [ 58 ]. Predicted value is [ 1 ], but the real value is [ 0 ]
Different values in position [ 73 ]. Predicted value is [ 1 ], but the real value is [ 0 ]
Different values in position [ 147 ]. Predicted value is [ 1 ], but the real value is [ 0 ]
Different values in position [ 328 ]. Predicted value is [ 1 ], but the real value is [ 0 ]
Different values in position [ 407 ]. Predicted value is [ 0 ], but the real value is [ 1 ]
Different values in position [ 526 ]. Predicted value is [ 1 ], but the real value is [ 0 ]
Different values in position [ 560 ]. Predicted value is [ 1 ], but the real value is [ 0 ]
Different values in position [ 842 ]. Predicted value is [ 1 ], but the real value is [ 0 ]
Different values in position [ 881 ]. Predicted value is [ 0 ], but the real value is [ 1 ]
```

6.5 Pregunta [16] y [17]

A continuación se mostrará una tabla que resumen las salidas de ambos scripts, el de la práctica 3 con una red RBF y el de la práctica actual, con una máquina vector soporte, usando el fichero *libsvm_17.py* con valor de C de 0.10, dado que ha sido el mejor valor anterior obtenido, y en gamma se dejará que la clase tome el valor.

Modelo	CCR
Maquina vector soporte	0.94
Red RBF	0.99

Según los resultados del script de la práctica anterior, especificando que el problema es de clasificación, se ha obtenido un resultado mejor que el de la máquina vector soporte.+

Se han utilizado un total de 400 neuronas RBF, un error cuadrático medio de 0.003996 en entrenamiento.

7. Documentación

[1]
https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[2]
https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

[3]
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

[4]
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

[5]
https://numpy.org/doc/stable/reference/generated/numpy.logspace.html