Pablo Arranz ropero y Juan Alberto camino Sáez



Si me lo como, ¿me muero?

**Índice**

* [Descripción del proyecto](#Descripción del proyecto)…………………….………………………………………………………….……3
* Apéndices……………………………..

# **Descripción del proyecto**

El propósito de este proyecto es aplicar diferentes algoritmos de aprendizaje automático a un conjunto de datos encontrado en la siguiente URL:

<https://www.kaggle.com/uciml/mushroom-classification>

Estos datos corresponden a un conjunto de atributos de diferentes setas y, en base a estos, si son venenosas o no.

El conjunto de datos presenta un total de 8124 instancias con 22 atributos cada una, y una columna indicando si la seta es venenosa o no.

Cada uno de los datos representa lo explicado en el Excel adjunto.

# **Consideraciones previas al desarrollo de los algoritmos**

Como podemos observar en el conjunto de datos original tenemos un conjunto de datos que maneja caracteres para definir el valor de cada atributo. Para una mayor facilidad de cómputo transformaremos dichos atributos a dígitos numéricos según el Excel adjunto. Nos ayudaremos de ciertas expresiones regulares similares a:

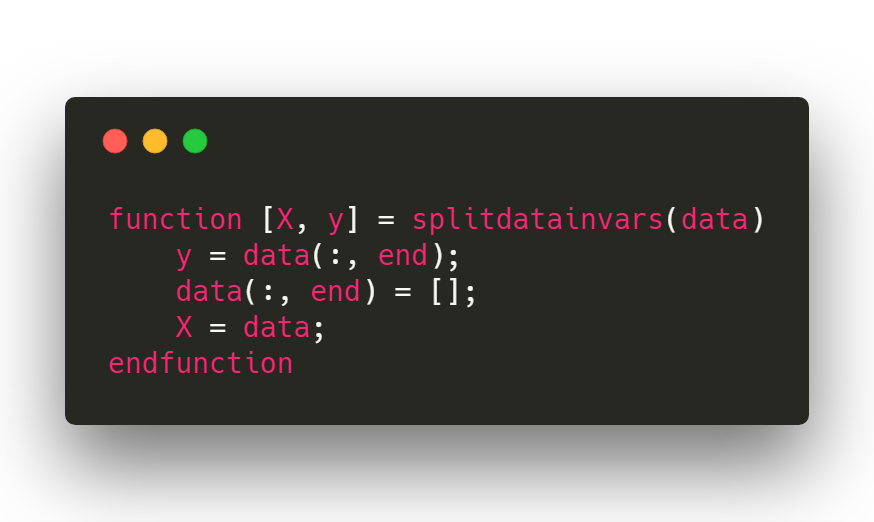
sed -i -E 's/^([0-9]+\,[0-9]+\,[0-9]+\,[0-9]+\,)t/\11/g' mushrooms.csv

Para adaptar la expresión regular, si queremos sustituir el atributo n repetiremos “[0-9]+\,” n-1 veces y después del cierre de paréntesis escribiremos el carácter que queremos sustituir seguido de “/\1” y el dígito por el que lo sustituiremos.

En el caso de la expresión regular del ejemplo se sustituirá, en la columna del 5º atributo, todos los caracteres t por el dígito 1.

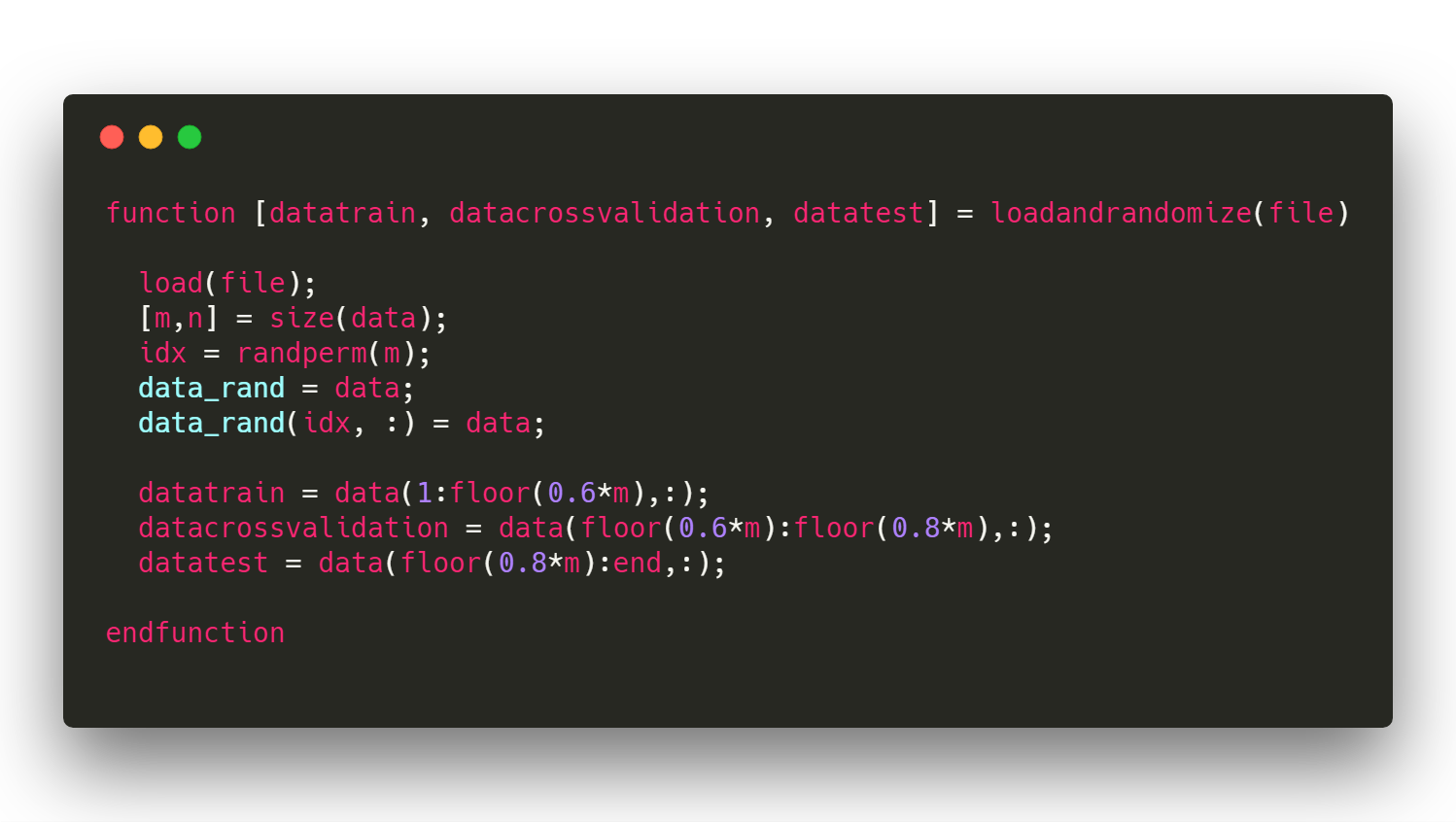
A continuación, crearemos un archivo llamado *mushroomdata* en el que se guardarán los datos de manera que ejecutando la instrucción load se cargarán automáticamente en la variable *data*.

Además, hemos creado un método llamado *splitdatainvars.m*, que al pasarle la variable *data*, devuelve los datos desglosados en X (todos los datos con sus atributos) e Y (clasificación de cada dato). Este es el código:



También hemos creado una función llamada *loadanrandomize.m*, que recibe un nombre de fichero y carga los datos y los separa en 3 subconjuntos aleatoriamente: subconjunto de entrenamiento (60% de los datos), subconjunto de cross-validation y subconjunto de test (20% de los datos respectivamente).

El código de dicho método es el siguiente:



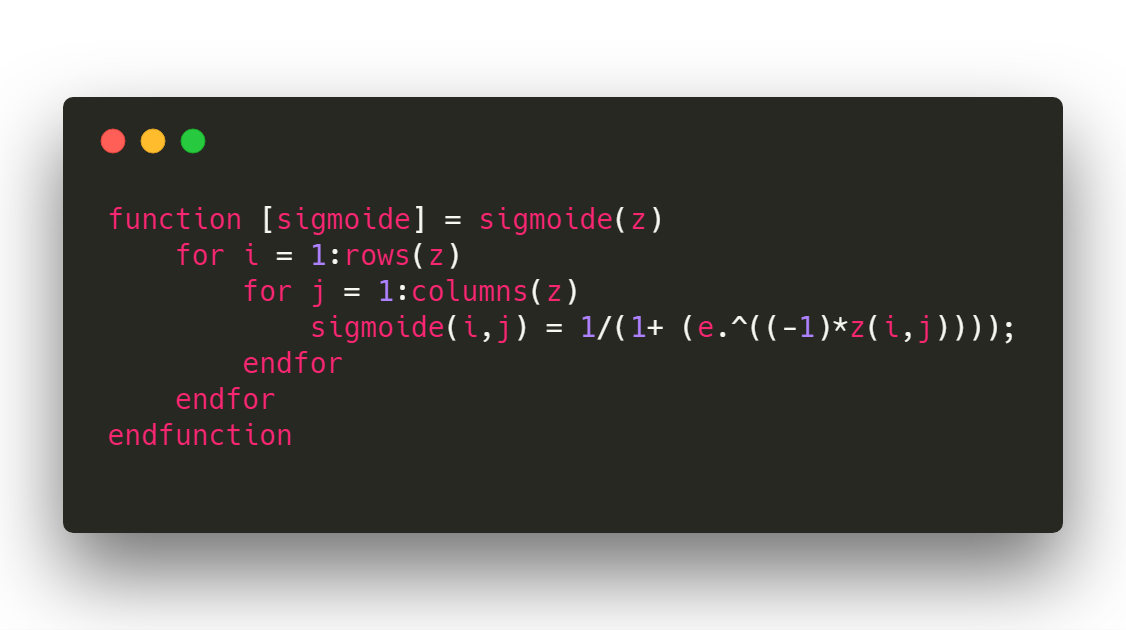
Una vez adaptados los datos de entrada y creados los métodos de carga, hemos aplicado distintas técnicas de aprendizaje automático con distintos detalles y ejecuciones propios para comprobar diferencias tanto en tiempo de ejecución, eficiencia, y porcentaje de datos clasificados correctamente en cada una. Hemos aplicado regresión logística, redes neuronales y Support Vector Machines (SVM)

# **Regresión Logística**

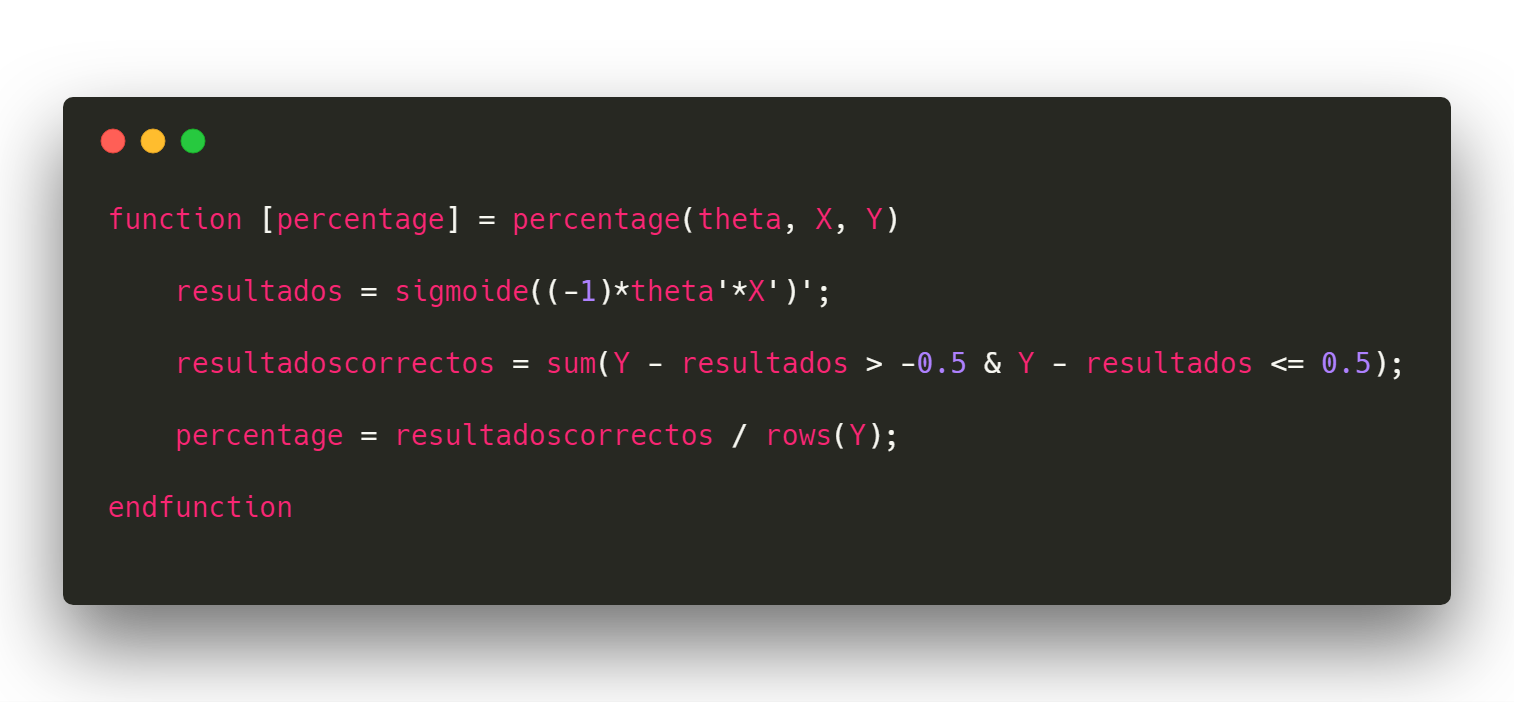
El primer algoritmo de aprendizaje empleado ha sido la regresión logística. No ha sido necesario emplear la multiclase debido a que nuestros datos únicamente tienen dos clases posibles.

Todo lo realizado con regresión logística se encuentra en la función *regLogistica.m*, que, al ser bastante larga debido a varios cálculos empleados, se irá desglosando por partes, incluyendo una explicación con lo realizado, los resultados obtenidos y las conclusiones que se han sacado en base a estos resultados.

En primer lugar, hemos cargado los datos con la función *load* y la función *splitdatainvars.m* mencionada anteriormente. Además, hemos creado y usado la función *sigmoide. m,* que se encarga de calcular el sigmoide de los datos pasados por parámetro. Este es el código:

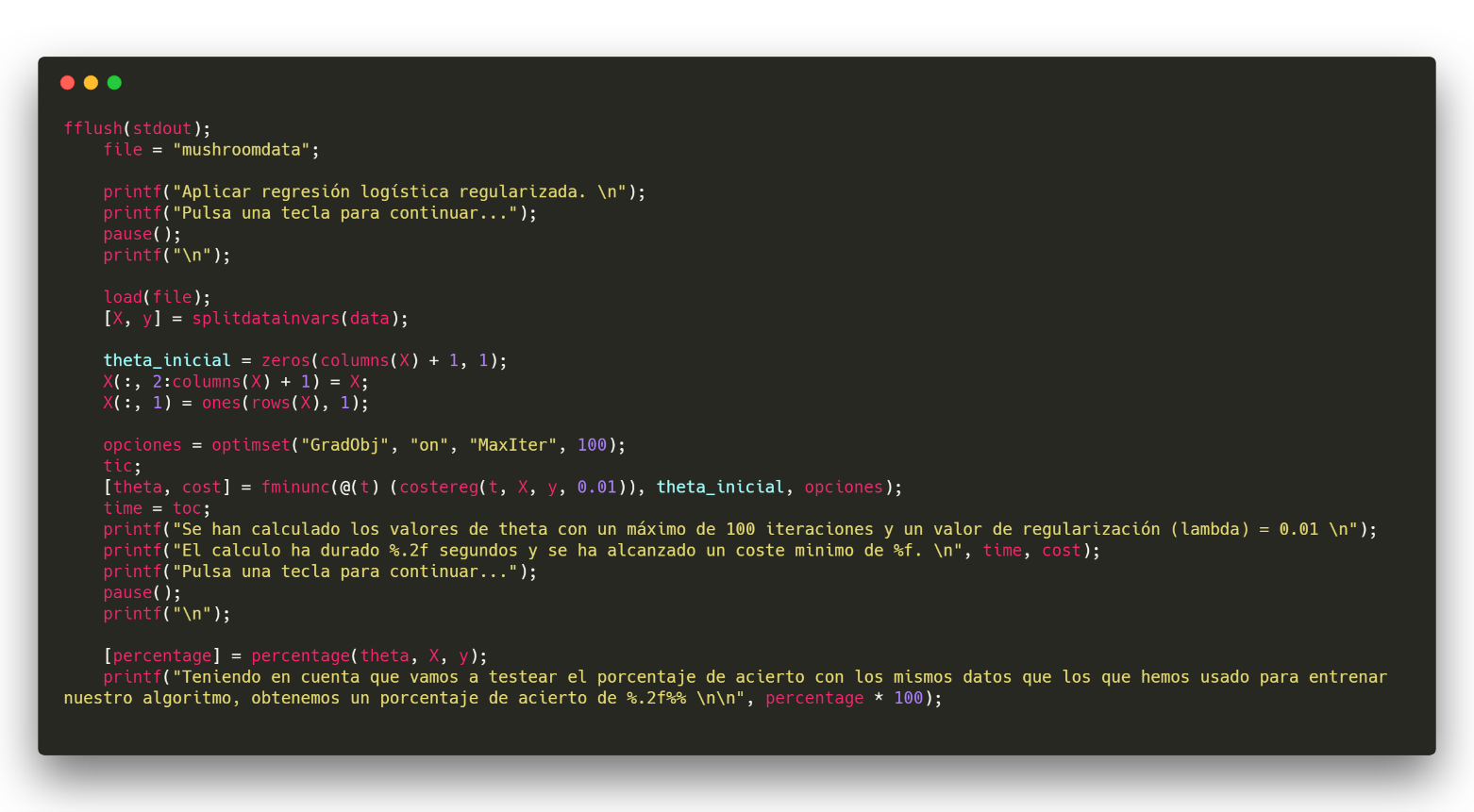


A continuación, hemos aplicado regresión logística sobre el 100% de los datos, sin hacer una división entre datos de entrenamiento, de cross-validation y de test en esta ocasión, y con un valor de lambda para la regularización de 0.01, obteniendo las thetas óptimas con un valor de la función de coste J = 0.120214 y un tiempo de ejecución de tiempo segundos. Posteriormente, hemos usado estas thetas para calcular el porcentaje de los datos clasificados correctamente, usando para ello la función *percentaje.m,* que se encarga de comprobar si la clasificación obtenida por el algoritmo de aprendizaje es la misma que la que tiene en los datos, y devolver el porcentaje de aciertos que la regresión logística ha obtenido. Este es el código de esta función

**

El resultado obtenido es que, tras realizar la regresión, un 95.34 % de los datos de entrenamiento son clasificados correctamente por el algoritmo de aprendizaje.

Esta es la parte del código de la función principal dedicada a esto:



Después, hemos vuelto a cargar los datos, pero esta vez sí hemos dividido los datos en tres subconjuntos (entrenamiento, cross-validation y test), usando la función *loadanrandomize.m* mencionada anteriormente.

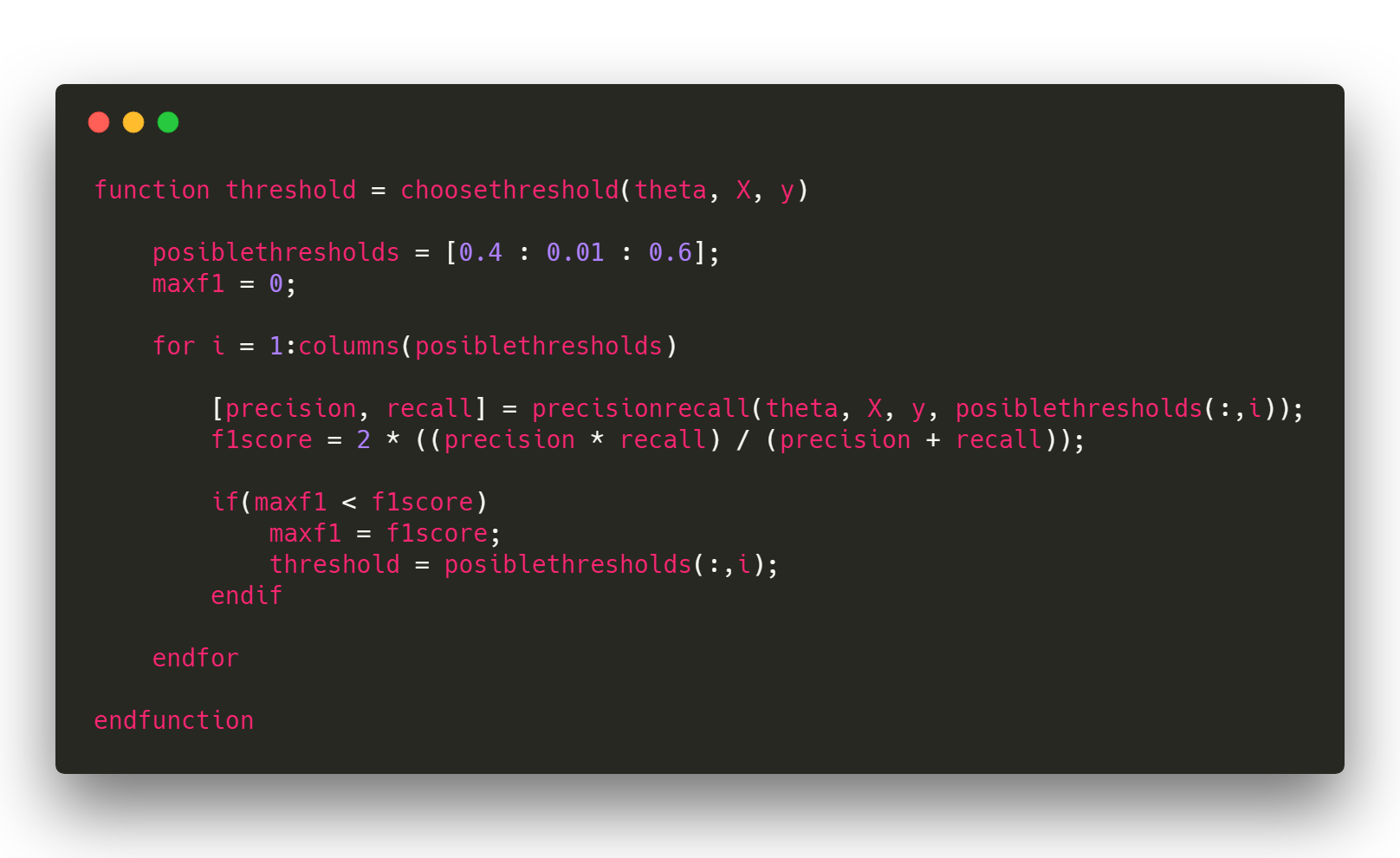
De esta manera, hemos vuelto a aplicar regresión logística, pero esta vez únicamente sobre el subconjunto de entrenamiento, usando el subconjunto de cross-validation para comprobar cuál era la lambda óptima para ser empleado. Para ello, hemos realizado en múltiples ocasiones la regresión logística variando lambda, y calculando el porcentaje de datos de cross-validation que clasificaba correctamente con la función *percentage.m,* y obteniendo el máximo. Este cálculo ha llevado un total de tiempo segundos, obteniendo que el lamba óptimo es 0.01, clasificando correctamente un 97.88% de los datos de cross-validation. Por último, hemos usado el subconjunto de test para comprobar cuántos datos clasificaba correctamente, obteniendo un porcentaje de 67.43% de datos bien clasificados.

A continuación, se muestra la parte del código de la función principal dedicada a lo explicado anteriormente:

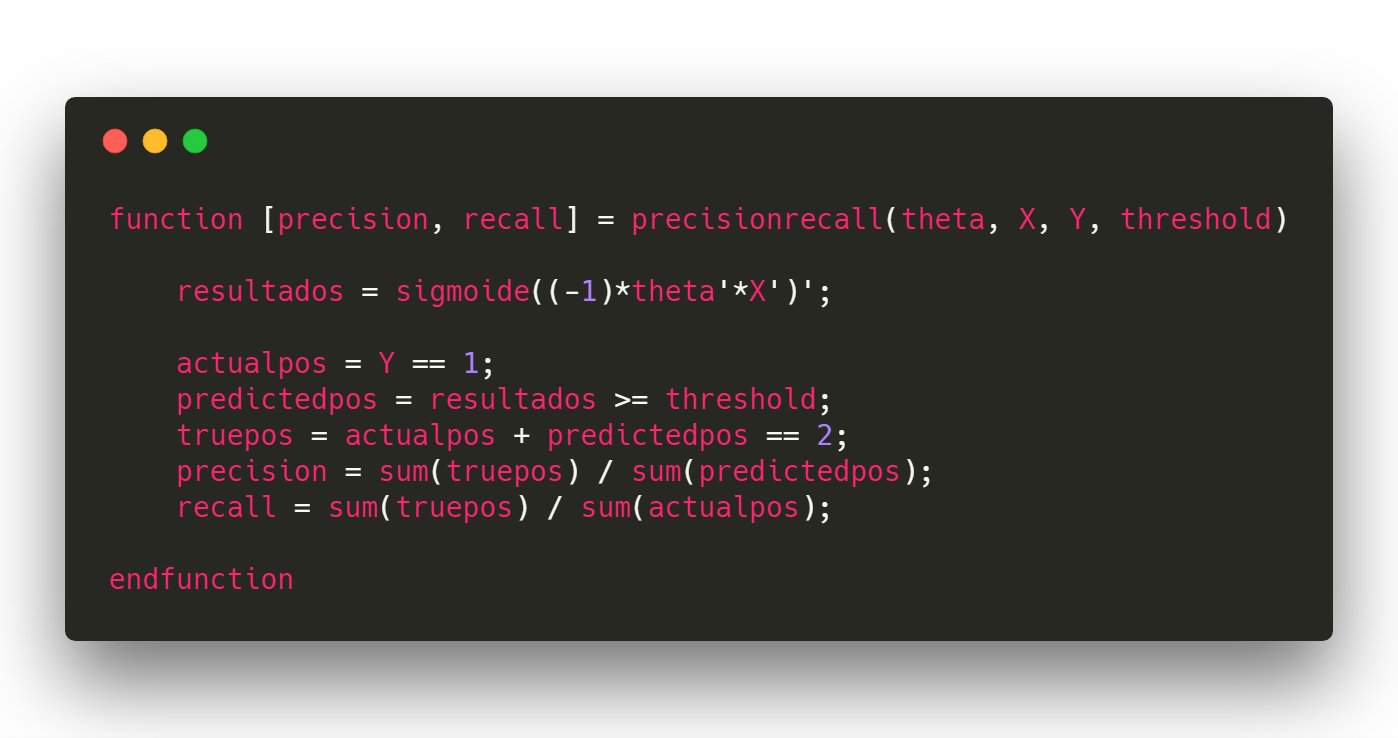


Para finalizar con la regresión logística, hemos calculado la precisión y el recall del algoritmo anterior, usando para ello la función *choosethreshold.m,* que obtiene el mejor threshold de este algoritmo, y posteriormente llamando a la función *precisionrecall.m,* que calcula los valores de precisión y recall obtenidos en base a las thetas resultantes del algoritmo de aprendizaje.

Este es el código de la función *choosethreshold.m*:

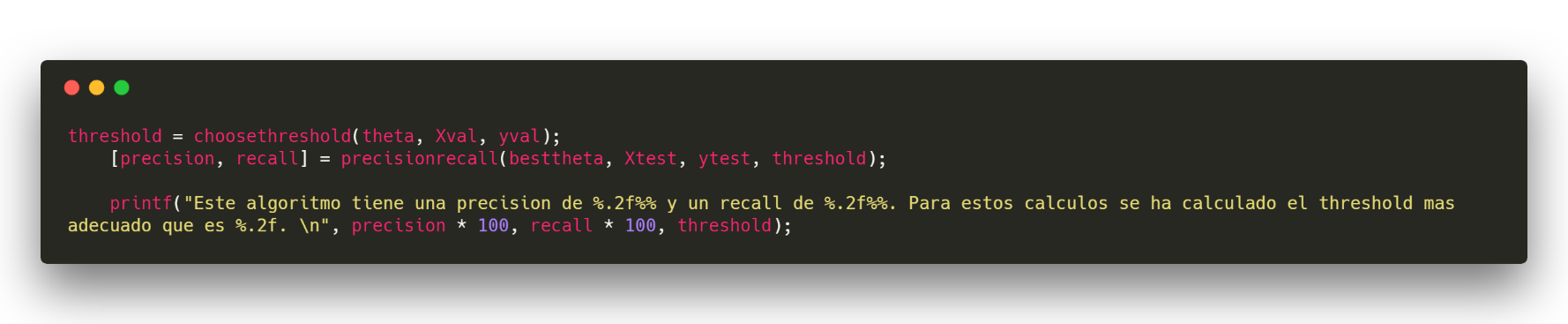


Y este, el código de la función *precisionrecall.m*:



Tras aplicar estas funciones, obtenemos que el threshold óptimo es 0.40, y de esta manera, obteniendo un precision y un recall de 88.93% y 73.31%., respectivamente.

Este es el código de la función principal que se dedica a esta parte:



Como conclusiones tras aplicar la regresión logística de esta manera, podemos destacar la diferencia de porcentaje de acierto en la clasificación al dividir los datos en subconjuntos frente a no hacerlo (siendo mayor en el segundo caso). Sin embargo, esto hace que funcione perfecto para los datos de entrenamiento, pero no tan bien para la predicción de nuevos datos. Por ello, es mejor separar los datos, comprobar cuál es la mejor regularización probando distintas lambdas y ver su porcentaje real de aciertos usando datos independientes de los datos usados para entrenar el algoritmo. También podemos apreciar que, a pesar de ser mejor, tarda mucho mas tiempo el primer caso que el segundo.

**Redes Neuronales**