Inteligencia Artificial
Grado en Ingeniería Informática en Sistemas de Información
ENSEÑANZAS PRÁCTICAS Y DE DESARROLLO

EPD3: Machine Learning – Regresión Logística

Objetivos

- Implementación en Python de un algoritmo de Regresión Logística para la construcción de un modelo de clasificación.

Bibliografía Básica

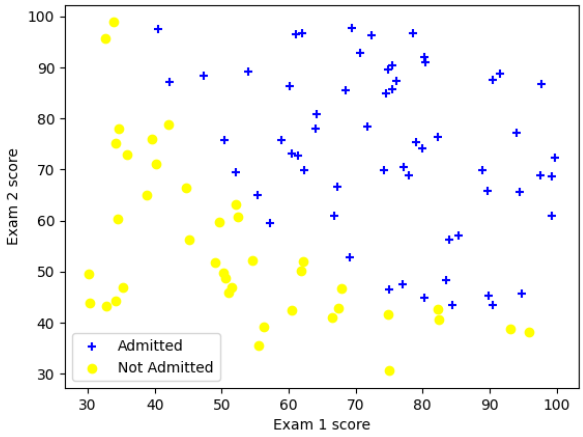
Machine Learning. Tom Mitchell. MacGraw-Hill, 1997

Ejercicios

Implementar un modelo de regresión logística para predecir las personas que formarán parte de nuestra empresa. Ponte en el papel de Director del departamento de recursos humanos, y quieres determinar que solicitudes de admisión puedes admitir en base al resultado de dos exámenes.

Dispones de datos históricos de anteriores convocatorias (`ex2data1.txt`) que se pueden utilizar como conjunto de entrenamiento para la regresión logística. Cada ejemplo revela la puntuación de cada examen y la decisión de ingreso.

Utilizar el script `main.py` para ir incorporando el código y las llamadas a las funciones que se piden en los siguientes ejercicios.



EJ1. Al igual que en la pasada EPD, antes de empezar la tarea visualizar el conjunto de datos. Añadir al fichero `main.py` el código necesario para cargar los datos, e implementar la función `plotData.py` para representar los datos gráficamente. A la derecha se muestra un ejemplo del resultado de la visualización.

EJ2. Antes de comenzar con la función de coste, recuerda que la hipótesis de la regresión logística está definida por $h_\theta(x) = g(\theta^T x)$, siendo `g` la función sigmoide definida como $g(z) = \frac{1}{1+e^{-z}}$. Implementa esta función en `sigmoid.py` de manera que se pueda llamar desde el programa principal `main.py`. Al evaluar esta función con el valor 0 el resultado debe ser exactamente 0.5, y que para un valor alto positivo será 1 y para un valor alto negativo será 0.

EJ3. Implementa una función que devuelva el coste. Con los parámetros `theta` inicializados a 0, la función devolverá un coste de 0.693. Además, implementa la función descenso del gradiente. Nota: Recuerde que debe usar la forma vectorizada para que la implementación sea eficiente.

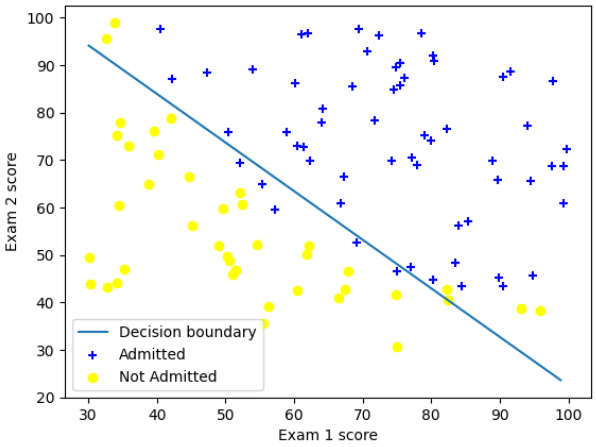
$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log\left(1 - h_\theta(x^{(i)})\right)\right]$$

EJ4. Python dispone de una función de optimización en la librería scipy que se puede utilizar para encontrar el mínimo de la función de coste con los parámetros `theta`. Usaremos la función `fmin_cg` (que usa como optimizador el gradiente conjugado). En el programa principal puedes ver escrito el código de la llamada a la función con los siguientes parámetros: la función de coste que devuelve el coste, los valores iniciales de `theta`, la función del descenso del gradiente y los argumentos de las funciones.

Modifique la función descenso del gradiente para usarla correctamente con la función de optimización.

El coste resultante debe ser 0,203. Los theta óptimos encontrados deben ser: [-25.17594986 0.20634863 0.20158987].

Representa la frontera de decisión como aparece en la figura de la derecha.



EJ5. Una vez obtenidos los parámetros, puedes utilizar el modelo para predecir si un candidato en particular será admitido o no. Prueba en el programa principal que un candidato con una puntuación de 45 y 85 en cada examen tiene una probabilidad del 0,776 de ser admitido.

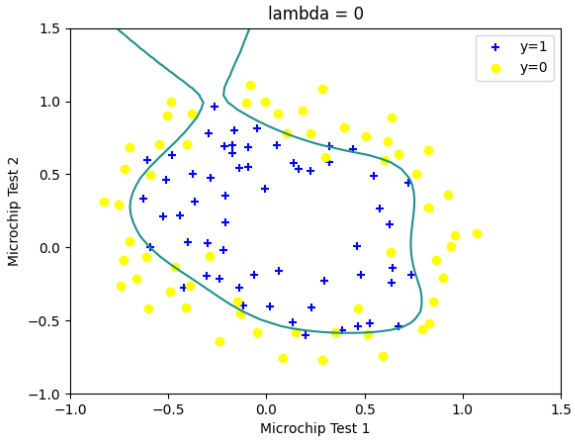
EJ6. Implementa la función `p=predict(theta, X)` de manera que con el modelo calculado `theta`, devuelva la predicción de un conjunto de test que se le pase como parámetro. En este caso se utilizará el mismo conjunto de entrenamiento `X`. Mostrar la exactitud obtenida.

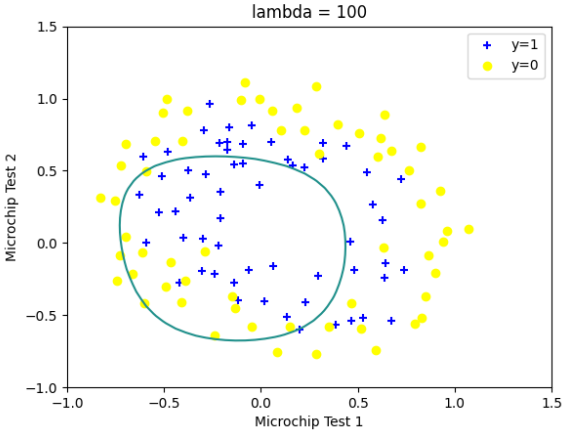
Problemas

P1. Repita los ejercicios EJ4, EJ5 y EJ6 usando la regresión logística de la librería sklearn.

P2. Implementa una regresión logística **regularizada** para predecir si los microchips de una fábrica pasan el control de calidad. Dispones de un conjunto de datos (`ex2data2.txt`) que contiene un histórico con los resultados de dos test realizados anteriormente a otras piezas, indicando si fueron aceptadas o no. Estos datos se utilizarán para obtener un modelo de regresión logística usando regularización para evitar el problema de *overfitting*.

1. Cargar y visualizar los datos.
2. Como se puede comprobar en la gráfica, los puntos no son separables linealmente. Una solución es crear más atributos mediante la función proporcionada en el fichero `P2_mapFeature.py`, que permitirá mapear los atributos en términos polinomiales de x_1 y x_2 hasta la sexta potencia (x_1, x_2, x_1^2, x_2^2, $x_1 * x_2$, $x_1 * x_2^2$,...). De esta manera obtendremos una matriz con 28 atributos, permitiendo obtener una frontera de decisión mucho más compleja que se pueda adaptar mejor a los puntos. Revisar la función facilitada. Otra posible opción es utilizar PolynomialFeatures de sklearn, realiza una prueba.
3. Implementa de nuevo la función que calcula el coste y el gradiente pero esta vez con regularización.
4. Igual que en el EJ4, utiliza la optimización de la librería scipy para obtener los parámetros `theta` óptimos.
5. Revisar cómo se muestra la frontera de decisión entre los ejemplos positivos y negativos mediante la función que se encuentra en el fichero `P2_plotDecisionBoundary.py`
6. Finalmente, predecir el mismo conjunto de entrenamiento y mostrar la exactitud obtenida. Después, probar con diferentes valores de `lambda` de manera que, como ocurre en las gráficas que se muestran a continuación, se observe el *overfitting* y la *underfitting*.


lambda = 0
y=1
y=0
Microchip Test 2
Microchip Test 1
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5


lambda = 100
y=1
y=0
Microchip Test 2
Microchip Test 1
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5