

2daintancia_Ceron_reg_log_onevsall_01_PoKer.ipynb

Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda Se guardaron todos los cambios

Comentar Compartir

Archivos + Código + Texto RAM Disco Editando

Universitario: Ceron Beimar Miguel

Carrera: Ingeniería de Sistemas

Datasets: Conjunto de datos del juego Poker

<https://www.kaggle.com/datasets/hosseinah1/poker-game-dataset>

Datasets usado es poker-hand-testing.csv

Repositorio GitHub:

<https://github.com/Beimar98/SIS420>

Haz doble clic (o ingresa) para editar

Información de atributos: Variables "X": 1) S1 "Palo de la carta n.º 1" ordinal (1-4) que representa {Corazones, Picas, Diamantes, Tréboles} 2) C1 "Rank of card #1" Numérico (1-13) que representa (As, 2, 3, ..., Queen, King) 3) S2 "Palo de la carta #2" Ordinal (1-4) que representa {Corazones, Picas, Diamantes, Tréboles} 4) C2 "Rank of card #2" Numérico (1-13) que representa (As, 2, 3, ..., Queen, King) 5) S3 "Palo de la carta #3" Ordinal (1-4) que representa {Corazones, Picas, Diamantes, Tréboles} 6) C3 "Rank of card #3" Numérico (1-13) que representa (As, 2, 3, ..., Queen, King) 7) S4 "Palo de la carta #4" Ordinal (1-4) que representa {Corazones, Picas, Diamantes, Tréboles} 8) C4 "Rank of card #4" Numérico (1-13) que representa (As, 2, 3, ..., Queen, King) 9) S5 "Palo de la carta #5" Ordinal (1-4) que representa {Corazones, Picas, Diamantes, Tréboles} 10) C5 "Rank of card 5" Numérico (1-13) que representa (As, 2, 3, ..., Queen, King)

Variables "Y": 11) CLASE "Mano de Poker" Ordinal Tipos(0-9) Describiendo cada tipo del cero al nueve es: 0: Nada en la mano; no es una mano de póquer reconocida 1: Un par; un par de rangos iguales dentro de cinco cartas 2: dos pares; dos pares de rangos iguales dentro de cinco cartas 3: Trío; tres rangos iguales dentro de cinco cartas 4: Escalera; cinco cartas, clasificadas secuencialmente sin espacios 5: Color; cinco cartas del mismo palo 6: Full; par + rango diferente tres de una clase 7: Cuatro de una clase; cuatro rangos iguales dentro de cinco cartas 8: Escalera de color; escalera + color o también valores de atributos faltantes: ninguno 9: escalera real; {As, Rey, Reina, Jota, Diez} + color

Clasificación multiclas

Introduction

En este ejercicio se implementa la regresión one-vs-all y una red neuronal para reconocimiento de dígitos.

Antes de empezar la ejecución de las partes de código correspondientes a los ejercicios, se requiere importar todas las librerías necesarias.

```
[2] # utilizado para la manipulación de directorios y rutas
import os

# Cálculo científico y vectorial para python
import numpy as np

# Librería para gráficos
from matplotlib import pyplot

# Módulo de optimización en scipy
from scipy import optimize

# módulo para cargar archivos en formato MATLAB
from scipy.io import loadmat

# le dice a matplotlib que incruste gráficos en el cuaderno
%matplotlib inline
```

1 Clasificación multiclas

Para este ejercicio, se usará regresión logística y redes neuronales para reconocer dígitos escritos a mano (de 0 a 9). El reconocimiento automático de dígitos escritos a mano se usa ampliamente en la actualidad, desde el reconocimiento de códigos postales (códigos

postales) en sobres de correo hasta el reconocimiento de montos escritos en cheques bancarios. Este ejercicio mostrara como los métodos que ha aprendido se pueden utilizar para esta tarea de clasificación.

La primera parte del ejercicio, extenderá la implementación anterior de la regresión logística y la aplicará a la clasificación de uno contra todos (one vs all).

1.1 Dataset

Se proporciona un conjunto de datos en `ex3data1.mat` que contiene 5000 ejemplos de entrenamiento de dígitos escritos a mano (este es un subconjunto del conjunto de datos de dígitos escritos a mano [MNIST] <http://yann.lecun.com/exdb/mnist>). El formato `.mat` significa que los datos se han guardado en un formato de matriz nativo Octave/MATLAB, en lugar de un formato de texto (ASCII) como un archivo `csv`. Usamos el formato `.mat` aquí para mostrar los diferentes formatos en los que se pueden presentar los datasets. Python proporciona mecanismos para cargar el formato nativo de MATLAB usando la función `loadmat` dentro del módulo `scipy.io`. Esta función devuelve un diccionario de Python con claves que contienen los nombres de las variables dentro del archivo `.mat`.

Hay 5000 ejemplos de entrenamiento en `ex3data1.mat`, donde cada ejemplo de entrenamiento es una imagen en escala de grises de 20 píxeles por 20 píxeles del dígito. Cada píxel está representado por un número de punto flotante que indica la intensidad de la escala de grises en esa ubicación. La cuadrícula de 20 por 20 píxeles se "desenrolla" en un vector de 400 dimensiones. Cada uno de estos ejemplos de entrenamiento se convierte en una sola fila en nuestra matriz de datos "X". Esto da una matriz `x` de 5000 por 400 donde cada fila es un ejemplo de entrenamiento para una imagen de dígitos escrita a mano.

$$X = \begin{bmatrix} - (x^{(1)})^T - \\ - (x^{(2)})^T - \\ \vdots \\ - (x^{(m)})^T - \end{bmatrix}$$

La segunda parte del conjunto de entrenamiento es un vector "y" de 5000 dimensiones que contiene etiquetas para el conjunto de entrenamiento.

Se inicia el ejercicio cargando primero el conjunto de datos.

```
[3] # Ingresar imágenes de dígitos de 20x20
    input_layer_size = 11

    # 10 etiquetas, de 1 a 10 (tomar en cuenta que se asigna "0" a la etiqueta 10)
    num_labels = 1000

    # datos de entrenamiento almacenados en los arreglos X, y
    data = np.loadtxt('/content/sample_data/poker-hand-testing_one_vs_all.txt', delimiter=',')
    X, y = data[:,10], data[:,10]
    # establecer el dígito cero en 0, en lugar del 10 asignado a este conjunto de datos
    # Esto se hace debido a que el conjunto de datos se utilizó en MATLAB donde no hay índice 0
    y[y == 10] = 0

    m = y.size

[4] print(X.shape)
    print(y.shape)
    print("-----")
    print(X)
    print(y)

    (1000000, 10)
    (1000000,)

    [[ 1.  1.  1. ... 3.  1. 12.]
     [ 3. 12.  3. ... 5.  2.  5.]
     [ 1.  9.  4. ... 2.  3.  9.]
     ...
     [ 1. 11.  4. ... 13.  2.  7.]
     [ 3. 11.  1. ... 13.  2.  8.]
     [ 2.  5.  2. ... 3.  3.  3.]]
    [0.  1.  1. ... 1.  1.  2.]]

[5] ### normalización de los datos
def normalizacion(X):
    X_norm = X.copy()
    mu = np.zeros(X.shape[1])
    sigma = np.zeros(X.shape[1])

    mu = np.mean(X, axis = 0)
    sigma = np.std(X, axis = 0)
    X_norm = (X - mu) / sigma

    return X_norm, mu, sigma

[6] X_norm, mu, sigma = normalizacion(X)
    print(X_norm.shape)

    (1000000, 10)
```

▼ 1.2 Visualización de los datos

Se comenzará visualizando un subconjunto del conjunto de entrenamiento. Se selecciona al azar, 100 filas de `x` y pasa esas filas a la función `displayData`. Esta función asigna cada fila a una imagen en escala de grises de 20 píxeles por 20 píxeles y muestra las imágenes juntas.

```

✓ [7] def displayData(X, example_width=None, figsize=(10, 10)):
    """
    Muestra datos 2D almacenados en X en una cuadrícula apropiada.
    """
    # Calcula filas, columnas
    if X.ndim == 2:
        m, n = X.shape
    elif X.ndim == 1:
        n = X.size
        m = 1
        X = X[None] # Promocionar a una matriz bidimensional
    else:
        raise IndexError('La entrada X debe ser 1 o 2 dimensiones.')

    example_width = example_width or int(np.round(np.sqrt(n)))
    example_height = n / example_width

    # Calcula el número de elementos a mostrar
    display_rows = int(np.floor(np.sqrt(m)))
    display_cols = int(np.ceil(m / display_rows))

    fig, ax_array = pyplot.subplots(display_rows, display_cols, figsize=figsize)
    fig.subplots_adjust(wspace=0.025, hspace=0.025)

    ax_array = [ax_array] if m == 1 else ax_array.ravel()

    for i, ax in enumerate(ax_array):
        ax.imshow(X[i].reshape(example_width, example_width, order='F'),
                  cmap='Greys', extent=[0, 1, 0, 1])
        ax.axis('off')

✓ [8] # Selecciona aleatoriamente 100 puntos de datos para mostrar
#rand_indices = np.random.choice(m, 80, replace=False)
#sel = X[rand_indices, :]

#displayData(sel)

```

▼ 1.3 Vectorización de regresión logística

Se utilizará múltiples modelos de regresión logística uno contra todos para construir un clasificador de clases múltiples. Dado que hay 10 clases, deberá entrenar 10 clasificadores de regresión logística separados. Para que esta capacitación sea eficiente, es importante asegurarse de que el código esté bien vectorizado.

En esta sección, se implementará una versión vectorizada de regresión logística que no emplea ningún bucle "for".

Para probar la regresión logística vectorizada, se usara datos personalizados como se definen a continuación.

```

✓ [9] # valores de prueba para los parámetros theta
theta_t = np.array([-2, -1, 1, 2], dtype=float)

# valores de prueba para las entradas
X_t = np.concatenate([np.ones((5, 1)), np.arange(1, 16).reshape(5, 3, order='F')/10.0], axis=1)

# valores de testeo para las etiquetas
y_t = np.array([1, 0, 1, 0, 1])

# valores de testeo para el parametro de regularizacion
lambda_t = 3

```

▼ 1.3.1 Vectorización de la función de costo

Se inicia escribiendo una versión vectorizada de la función de costo. En la regresión logística (no regularizada), la función de costo es

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

Para calcular cada elemento en la suma, tenemos que calcular $h_\theta(x^{(i)})$ para cada ejemplo i , donde $h_\theta(x^{(i)}) = g(\theta^T x^{(i)})$ y $g(z) = \frac{1}{1+e^{-z}}$ es la función sigmoidea. Resulta que podemos calcular esto rápidamente para todos los ejemplos usando la multiplicación de matrices. Definamos X y θ como

$$X = \begin{bmatrix} -(x^{(1)})^T \\ -(x^{(2)})^T \\ \vdots \\ -(x^{(m)})^T \end{bmatrix} \quad \text{and} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

Luego, de calcular el producto matricial $X\theta$, se tiene:

$$X\theta = \begin{bmatrix} -(x^{(1)})^T \theta \\ -(x^{(2)})^T \theta \\ \vdots \\ -(x^{(m)})^T \theta \end{bmatrix} = \begin{bmatrix} -\theta^T x^{(1)} \\ -\theta^T x^{(2)} \\ \vdots \\ -\theta^T x^{(m)} \end{bmatrix}$$

En la última igualdad, usamos el hecho de que $a^T b = b^T a$ si a y b son vectores. Esto permite calcular los productos $\theta^T x^{(i)}$ para todos los

ejemplos i en una línea de código.

1.3.2 Vectorización del gradiente

Recordemos que el gradiente del costo de regresión logística (no regularizado) es un vector donde el elemento j^{th} se define como

$$\frac{\partial J}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^m \left((h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \right)$$

Para vectorizar esta operación sobre el conjunto de datos, se inicia escribiendo todas las derivadas parciales explícitamente para todos θ_j ,

$$\begin{aligned} \begin{bmatrix} \frac{\partial J}{\partial \theta_0} \\ \frac{\partial J}{\partial \theta_1} \\ \frac{\partial J}{\partial \theta_2} \\ \vdots \\ \frac{\partial J}{\partial \theta_n} \end{bmatrix} &= \frac{1}{m} \begin{bmatrix} \sum_{i=1}^m ((h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}) \\ \sum_{i=1}^m ((h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)}) \\ \sum_{i=1}^m ((h_\theta(x^{(i)}) - y^{(i)}) x_2^{(i)}) \\ \vdots \\ \sum_{i=1}^m ((h_\theta(x^{(i)}) - y^{(i)}) x_n^{(i)}) \end{bmatrix} \\ &= \frac{1}{m} \sum_{i=1}^m ((h_\theta(x^{(i)}) - y^{(i)}) x^{(i)}) \\ &= \frac{1}{m} X^T (h_\theta(x) - y) \end{aligned}$$

donde

$$h_\theta(x) - y = \begin{bmatrix} h_\theta(x^{(1)}) - y^{(1)} \\ h_\theta(x^{(2)}) - y^{(2)} \\ \vdots \\ h_\theta(x^{(m)}) - y^{(m)} \end{bmatrix}$$

Nota $x^{(i)}$ es un vector, mientras $h_\theta(x^{(i)}) - y^{(i)}$ es un escalar (simple número). Para comprender el último paso de la derivación, dejemos $\beta_i = (h_\theta(x^{(m)}) - y^{(m)})$ y observar que:

$$\sum_i \beta_i x^{(i)} = \begin{bmatrix} | & | & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & & | \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_m \end{bmatrix} = x^T \beta$$

donde los valores $\beta_i = (h_\theta(x^{(i)}) - y^{(i)})$.

La expresión anterior nos permite calcular todas las derivadas parciales sin bucles. Si se siente cómodo con el álgebra lineal, le recomendamos que trabaje con las multiplicaciones de matrices anteriores para convencerse de que la versión vectorizada hace los mismos cálculos.

** Consejo de depuración: ** El código de vectorización a veces puede ser complicado. Una estrategia común para la depuración es imprimir los tamaños de las matrices con las que está trabajando usando la propiedad 'shape' de las matrices 'numpy'.

Por ejemplo, dada una matriz de datos X de tamaño 100×20 (100 ejemplos, 20 características) y θ , un vector con tamaño 20, puede observar que `np.dot(X, theta)` es una operación de multiplicación válida, mientras que `np.dot(theta, X)` no lo es.

Además, si tiene una versión no vectorizada de su código, puede comparar la salida de su código vectorizado y el código no vectorizado para asegurarse de que produzcan las mismas salidas.

```
[10] def sigmoid(z):
    """
    Calcula la sigmoid de z.
    """
    return 1.0 / (1.0 + np.exp(-z))

[11] def lrCostFunction(theta, X, y, lambda_):
    m = y.size

    # convierte las etiquetas a valores enteros si son booleanos
    if y.dtype == bool:
        y = y.astype(int)

    J = 0
    grad = np.zeros(theta.shape)

    h = sigmoid(X.dot(theta.T))

    temp = theta
    temp[0] = 0

    J = (1 / m) * np.sum(-y.dot(np.log(h)) - (1 - y).dot(np.log(1 - h))) + (lambda_ / (2 * m)) * np.sum(np.square(temp))

    grad = (1 / m) * (h - y).dot(X)
    grad = grad + (lambda_ / m) * temp

    return J, grad
```

1.3.3 Vectorización regularizada de la regresión logística

Una vez implementada la vectorización para la regresión logística, corresponde agregar regularización a la función de costo. Para la regresión logística regularizada, la función de costo se define como

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m \left[-y^{(i)} \log\left(h_\theta\left(x^{(i)}\right)\right) - \left(1 - y^{(i)}\right) \log\left(1 - h_\theta\left(x^{(i)}\right)\right) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

Tomar en cuenta que no debería regularizarse θ_0 que se usa para el término de sesgo. En consecuencia, la derivada parcial del costo de regresión logística regularizado para θ_j se define como

$$\begin{aligned}\frac{\partial J(\theta)}{\partial \theta_0} &= \frac{1}{m} \sum_{i=1}^m \left(h_\theta\left(x^{(i)}\right) - y^{(i)} \right) x_j^{(i)} && \text{for } j = 0 \\ \frac{\partial J(\theta)}{\partial \theta_0} &= \left(\frac{1}{m} \sum_{i=1}^m \left(h_\theta\left(x^{(i)}\right) - y^{(i)} \right) x_j^{(i)} \right) + \frac{\lambda}{m} \theta_j && \text{for } j \geq 1\end{aligned}$$

** Python/numpy Consejo: ** Al implementar la vectorización para la regresión logística regularizada, a menudo es posible que solo desee sumar y actualizar ciertos elementos de θ . En `numpy`, puede indexar en las matrices para acceder y actualizar solo ciertos elementos.

Por ejemplo, $A[:, 3:5] = B[:, 1:3]$ reemplazará las columnas con índice 3 a 5 de A con las columnas con índice 1 a 3 de B.

Para seleccionar columnas (o filas) hasta el final de la matriz, puede dejar el lado derecho de los dos puntos en blanco. Por ejemplo, $A[:, 2:]$ solo devolverá elementos desde 3^{rd} a las últimas columnas de A . Si deja el tamaño de la mano izquierda de los dos puntos en blanco, seleccionará los elementos del principio de la matriz. Por ejemplo, $A[:, :2]$ selecciona las dos primeras columnas y es equivalente a $A[:, 0:2]$. Además, puede utilizar índices negativos para indexar matrices desde el final. Por lo tanto, $A[:, :-1]$ selecciona todas las columnas de A excepto la última columna, y $A[:, :-5]$ selecciona la columna 5^{th} desde el final hasta la última columna.

Por lo tanto, podría usar esto junto con las operaciones de suma y potencia (**) para calcular la suma de solo los elementos que le interesan (por ejemplo, `np.sum(z[1:]**2)`).

```
[12]: J, grad = lrCostFunction(theta_t, X_t, y_t, lambda_t)

      Costo       : {:.6f}'.format(J))
      Costo esperadot: 2.534819'
      -----
      Gradienes:
      print' [{:.6f}, {:.6f}, {:.6f}]'.format(*grad))
      Gradienes esperados:
      print' [0.146561, -0.548558, 0.724722, 1.398003]');
```

```
Costo       : 2.534819
Costo esperadot: 2.534819
-----
Gradienes:
[0.146561, -0.548558, 0.724722, 1.398003]
Gradienes esperados:
[0.146561, -0.548558, 0.724722, 1.398003]
```

1.4 Clasificación One-vs-all

En esta parte del ejercicio, se implementará la clasificación de uno contra todos mediante el entrenamiento de múltiples clasificadores de regresión logística regularizados, uno para cada una de las clases K en nuestro conjunto de datos. En el conjunto de datos de dígitos escritos a mano, $K = 10$, pero su código debería funcionar para cualquier valor de K .

El argumento `y` de esta función es un vector de etiquetas de 0 a 9. Al entrenar el clasificador para la clase $k \in \{0, \dots, K-1\}$, querrá un vector K -dimensional de etiquetas y , donde y_j in 0, 1 indica si la instancia de entrenamiento j^{th} pertenece a la clase k ($y_j = 1$), o si pertenece a una clase diferente ($y_j = 0$).

Además, se utiliza `optimize.minimize` de `scipy` para este ejercicio.

```
[13]: def oneVsAll(X, y, num_labels, lambda_):
    """
    Trains num_labels logistic regression classifiers and returns
    each of these classifiers in a matrix all_theta, where the i-th
    row of all_theta corresponds to the classifier for label i.

    Parameters
    -----
    X : array_like
        The input dataset of shape (m x n). m is the number of
        data points, and n is the number of features. Note that we
        do not assume that the intercept term (or bias) is in X, however
        we provide the code below to add the bias term to X.

    y : array_like
        The data labels. A vector of shape (m,).

    num_labels : int
        Number of possible labels.

    lambda_ : float
        The logistic regularization parameter.

    Returns
    -----
    all_theta : array_like
        The trained parameters for logistic regression for each class.
        This is a matrix of shape (K x n+1) where K is number of classes
        (ie. `numlabels`) and n is number of features without the bias.
    """

    m, n = X.shape
    K = num_labels
```

```

# algunas variables utiles
m, n = X.shape

all_theta = np.zeros((num_labels, n + 1))

# Agrega unos a la matriz X
X = np.concatenate([np.ones((m, 1)), X], axis=1)

for c in np.arange(num_labels):
    initial_theta = np.zeros(n + 1)
    options = { 'maxiter': 50}
    res = optimize.minimize(lrCostFunction,
                            initial_theta,
                            (X, (y == c), lambda_),
                            jac=True,
                            method='CG',
                            options=options)

    all_theta[c] = res.x

return all_theta

```

[14] `lambda_ = 0.01`
`num_labels = 10`
`all_theta = oneVsAll(X_norm, y, num_labels, lambda_)`

[15] `print(all_theta)`

```

[[ 4.84221251e-03  1.1988172e-03 -5.46366261e-03  1.77418344e-03
-1.3132612e-03 -8.38304847e-04 -3.20777668e-03  2.24926534e-03
-3.42323117e-03  2.10941969e-03  3.25274495e-04]
[-3.12533596e-01 -1.60481401e-03  3.59709888e-03 -2.40560539e-03
4.05196829e-04  5.40505258e-04  1.53630388e-03 -1.16843820e-03
1.00046705e-03 -8.55696854e-04 -2.65720453e-03]
[-2.99572986e+00 -7.19525648e-04 -5.75722175e-03  6.53433954e-04
-5.82586621e-03 -2.93526318e-03 -1.93224905e-03 -3.70151741e-03
2.69794716e-03 -2.28119882e-03  1.41286261e-04]
[-3.83593510e+00  4.42011435e-03  1.57358126e-02  7.84697461e-03
6.05503454e-03  2.56129432e-03  8.47609391e-03 -3.12677623e-03
5.00286948e-03 -5.68063348e-03  9.45758848e-03]
[-5.56451019e+00  1.15607992e-02  8.81778823e-02 -2.44218272e-02
7.55268883e-02  1.05898786e-02  8.38708618e-02 -2.54395579e-02
8.29217051e-02 -2.64477209e-02  9.78300799e-02]
[-6.22069098e+00  2.27341176e-02  1.93177739e-02  2.33062612e-02
2.48856026e-02  2.23573798e-02  3.96860222e-03  2.27515765e-02
-7.75124857e-03  2.36112001e-02 -2.60472286e-02]
[-6.55966998e+00 -1.41804908e-02  4.01147075e-02  2.53487535e-03
2.47562816e-02  4.28324840e-02  2.27277092e-02 -2.22094820e-03
5.77935280e-02 -3.00610774e-02  3.49647439e-02]
[-8.41675878e+00 -5.90252331e-02 -6.50647925e-02 -1.21080098e-02
-4.17730643e-02  4.68829576e-02 -1.18936064e-01  4.33187655e-02
-1.13216064e-02  3.77201190e-03 -6.41512438e-02]
[-1.08142564e+01 -1.80407481e-02  1.00209572e-01 -1.79405238e-02
3.75617467e-02 -1.81409569e-02  7.42775365e-02 -1.80459678e-02
6.59055477e-03 -1.78496607e-02  2.30282459e-02]
[-1.22172068e+01  1.78301958e-01  4.66987918e-02  1.78385143e-01
4.57808758e-02  1.78183353e-01  3.14543056e-01  1.78587231e-01
6.68139943e-02  1.78395383e-01  2.94682346e-01]]

```

1.4.1 Prediccion One-vs-all

Después de entrenar el clasificador de one-vs-all, se puede usarlo para predecir el dígito contenido en una imagen determinada. Para cada entrada, debe calcular la "probabilidad" de que pertenezca a cada clase utilizando los clasificadores de regresión logística entrenados. La función de predicción one-vs-all seleccionará la clase para la cual el clasificador de regresión logística correspondiente genera la probabilidad más alta y devolverá la etiqueta de clase (0, 1, ..., K-1) como la predicción para el ejemplo de entrada.

```

[16] def predictOneVsAll(all_theta, X):

    m = X.shape[0];
    num_labels = all_theta.shape[0]

    p = np.zeros(m)

    # Add ones to the X data matrix
    X = np.concatenate([np.ones((m, 1)), X], axis=1)
    p = np.argmax(sigmoid(X.dot(all_theta.T)), axis = 1)

    return p

```

Una vez que haya terminado, se llama a la función `predictOneVsAll` usando el valor aprendido de θ . Debería apreciarse que la precisión del conjunto de entrenamiento es de aproximadamente 95,1% (es decir, clasifica correctamente el 95,1% de los ejemplos del conjunto de entrenamiento).

```

[17] ####      Prueba del emtrenamiento

[18] data = np.loadtxt("../content/sample_data/poker-hand-testing_one_vs_all.txt", delimiter=',')
X_test = data[:, :10]
print(X_test.shape)

```

✓ 0 s se ejecutó 17:06

2daIntancia_Ceron_IA.ipynb

Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda Se guardaron todos los cambios

Archivos

+ Código + Texto RAM Disco Editando

Universitario: Ceron Beimar Miguel

Carrera: Ingeniería de Sistemas

Datasets: Conjunto de datos del juego Poker

<https://www.kaggle.com/datasets/hosseinh1/poker-game-dataset>

Repository GitHub:

<https://github.com/Beimar98/SIS420>

Información de atributos: Variables "X": 1) S1 "Palo de la carta n." 1º ordinal (1-4) que representa (Corazones, Picas, Diamantes, Tréboles)
 2) C1 "Rank of card #1" Numérico (1-13) que representa (As, 2, 3, ..., Queen, King)
 3) S2 "Palo de la carta #2" Ordinal (1-4) que representa (Corazones, Picas, Diamantes, Tréboles)
 4) C2 "Rank of card #2" Numérico (1-13) que representa (As, 2, 3, ..., Queen, King)
 5) S3 "Palo de la carta #3" Ordinal (1-4) que representa (Corazones, Picas, Diamantes, Tréboles)
 6) C3 "Rank of card #3" Numérico (1-13) que representa (As, 2, 3, ..., Queen, King)
 7) S4 "Palo de la carta #4" Ordinal (1-4) que representa (Corazones, Picas, Diamantes, Tréboles)
 8) C4 "Rank of card #4" Numérico (1-13) que representa (As, 2, 3, ..., Queen, King)
 9) S5 "Palo de la carta #5" Ordinal (1-4) que representa (Corazones, Picas, Diamantes, Tréboles)
 10) C5 "Rank of card #5" Numérico (1-13) que representa (As, 2, 3, ..., Queen, King)

Variables "Y": 11) CLASE "Mano de Poker" Ordinal Tipos(0-9) Describiendo cada tipo del cero al nueve es: 0: Nada en la mano; no es una mano de póker reconocida 1: Un par; un par de rangos iguales dentro de cinco cartas 2: dos pares; dos pares de rangos iguales dentro de cinco cartas 3: Tiro; tres rangos iguales dentro de cinco cartas 4: Escalera; cinco cartas, clasificadas secuencialmente sin espacios 5: Color; cinco cartas del mismo palo 6: Full; par + rango diferente tres de una clase 7: Cuatro de una clase; cuatro rangos iguales dentro de cinco cartas 8: Escalera de color; escalera + color o también valores de atributos faltantes: ninguno 9: escalera real; {As, Rey, Reina, Jota, Diez} + color

```
[??] # se utiliza para el manejo de rutas y directorios.
import os

# Calculo científico y vectorial para python
import numpy as np
import pandas as pd
# Librerías para graficar
from matplotlib import pyplot

# Módulo de optimización de scipy
from scipy import optimize

# le dice a matplotlib que incluya gráficos en el cuaderno
%matplotlib inline
```

```
[78] data = pd.read_csv('/content/sample_data/poker-hand-testing3.csv', header=None)
```

	0	1	2	3	4	5	6	7	8	9	10
0	1	1	1	13	2	4	2	3	1	12	0
1	3	12	3	2	3	11	4	5	2	5	1
2	1	9	4	6	1	4	3	2	3	9	1
3	1	4	3	13	2	13	2	1	3	6	1
4	3	10	2	7	1	2	2	11	4	9	0
..
999995	3	1	1	12	2	9	4	9	2	6	1
999996	3	3	4	5	2	7	1	4	4	3	1
999997	1	11	4	7	3	9	1	13	2	7	1
999998	3	11	1	8	1	1	3	13	2	8	1
999999	2	5	2	9	4	9	2	3	3	3	2

1000000 rows x 11 columns

```
[79] data.describe()
```

	0	1	2	3	4	5	6
count	1000000.000000	1000000.000000	1000000.000000	1000000.000000	1000000.000000	1000000.000000	1000000.000000
mean	2.500493	6.997927	2.499894	7.006997	2.500871	6.998873	2.500393
std	1.117768	3.743374	1.118568	3.743481	1.118225	3.741890	1.117245
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
25%	2.000000	4.000000	1.000000	4.000000	1.000000	4.000000	2.000000
50%	3.000000	7.000000	3.000000	7.000000	3.000000	7.000000	3.000000
75%	3.000000	10.000000	4.000000	10.000000	4.000000	10.000000	3.000000
max	4.000000	13.000000	4.000000	13.000000	4.000000	13.000000	4.000000

```
[80] #data = data.drop([0], axis=1)
#data
```

```
[81] data = np.array(data)
X = data[:, 0:10]
y = data[:, 10]
print(X)
print(y)
```

```
[[1 1 1 13 2 4 2 3 1 12]
 [3 12 3 2 3 11 4 5 2 5]
 [1 9 4 6 1 4 3 2 3 9]
 ...
 [11 4 7 3 9 1 13 2 7]
 [11 1 8 1 1 3 13 2 8]
 [5 2 9 4 9 2 3 3 3]
 [1 1 1 1 2]
 [3 12 3 ... 5 2 5]
 [1 9 4 ... 2 3 9]
 ...
 [11 4 ... 13 2 7]
 [11 1 8 1 1 3 2 8]
 [5 2 9 3 3 3]]
 [0 1 1 ... 1 1 2]
```

```
[81]
```

```
[81]
```

```
[81]
```

```
[82] def featureNormalize(X):
    X_norm = X - copy()
    mu = np.zeros(X.shape[1])
    sigma = np.zeros(X.shape[1])

    mu = np.mean(X, axis = 0)
    sigma = np.std(X, axis = 0)
    X_norm = (X - mu) / sigma

    return X_norm, mu, sigma
```

```

[83] # llama featureNormalize con los datos cargados
X_norm, mu, sigma = featureNormalize(X)

print(X)
print('Media calculada:', mu)
print('Desviación estandar calculada:', sigma)
print(X_norm)

[[ 1  1  1  ... 3  1 12]
 [ 1  3 12 3  ... 5  2  5]
 [ 1  1  9  4  ... 2  3  9]
 ...
 [[ 1 11  4  ... 13  2  7]
 [ 1 11  3  ... 13  2  8]
 [ 2  2  3  ... 11 11]
Media calculada: [2.060493 6.997927 2.499894 7.006097 2.500871 6.998873 2.500393 7.002298
2.499451 6.989481]
Desviación estandar calculada: [1.11726731 3.74337237 1.11856781 3.73437911 1.11822415 3.74188826
1.117248194 1.11426886 1.11884759 3.73989229]
[[ 1.-34240194 -1.60227902 -1.3409572 ... -1.06977044 -1.34065473
1.044687924 1.33624777 0.44709493 ... -0.53519234 -0.44635782
-0.53196211]
[ 1.-34240194 0.53483138 1.34109525 ... -1.3370595 0.44733989
0.53758741]
[ 1.-34240194 1.06910898 1.34109525 ... 1.06312009 -0.44635782
0.00281265]
[ 1.044687924 1.06910898 -1.34090572 ... 1.60312009 -0.44635782
0.27020083]
[ 1.-44776135 -0.53372382 -0.4469854 ... -1.06977044 0.44733989
-1.06673687]]

```

```

[84] # Añade el termino de intersección a X
# (columna de unos para X0)
m = y.size
X = np.concatenate([np.ones((m, 1)), X_norm], axis=1)

```

```

[85] print(X)

[[ 1.  -1.34240194 -1.60227902 ... -1.06977044 -1.34005473
1.33974955 0.44687924 1.33624777 ... -0.53519234 -0.44635782
-0.53196211]
[ 1. -1.34240194 0.53483138 1.3370595 0.44733989
0.53758741]
...
[[ 1.  -1.34240194 1.06910898 1.34109525 ... 1.06312009 -0.44635782
0.00281265]
[ 1. 0.44687924 1.06910898 ... 1.60312009 -0.44635782
0.27020083]
[ 1. -0.44776135 -0.53372382 -0.4469854 ... -1.06977044 0.44733989
-1.06673687]]

```

```

[86] #FUNCION SIGMOIDE
def sigmoid():
    # Calcula la sigmoid de una entrada z
    # convierte la entrada a un arreglo numpy
    z = np.array(z)

    g = np.zeros(z.shape)
    g = 1 / (1 + np.exp(-z))

    return g

```

```

[87] # Prueba la implementacion de la funcion sigmoid
z = 0
g = sigmoid(z)

print('g( z ) = ', g)
g( 0 ) = 0.5

```

```

[88] def calcularCosto(theta, X, y):
    # Inicializar algunos valores utiles
    m = y.size # numero de ejemplos de entrenamiento

    J = 0
    h = sigmoid(X.dot(theta.T))
    J = (1 / m) * np.sum(-(y.dot(np.log(h)) - (1 - y).dot(np.log(1 - h)))) 

    return J

```

```

[89] def descensoGradiente(theta, X, y, alpha, num_iters):
    # Inicializa algunos valores
    m = y.shape[0] # numero de ejemplos de entrenamiento

    # realiza una copia de theta, el cual será actualizada por el descenso por el gradiente
    theta = theta.copy()
    J_history = []

    for i in range(num_iters):
        h = sigmoid(X.dot(theta.T))
        theta = theta - (alpha / m) * (h - y).dot(X)

        J_history.append(calcularCosto(theta, X, y))

    return theta, J_history

```

```

[90] # Elegir algun valor para alpha (probar varias alternativas)
alpha = 0.0001
num_iters = 1000
alpha = 0.1
num_iters = 30
# inicializa theta y ejecuta el descenso por el gradiente
theta = np.zeros(11)
theta, J_history = descensoGradiente(theta, X, y, alpha, num_iters)

# Grafica la convergencia del costo
pyplot.plot(np.arange(len(J_history)), J_history, lw=2)
pyplot.xlabel('Número de iteraciones')
pyplot.ylabel('Costo J')

# Muestra los resultados del descenso por el gradiente
print('theta calculado por el descenso por el gradiente: {:s}'.format(str(theta)))

#0.0000
X_array = [[1,1,1,1,1,1,2,4,2,3,1,12]
            tipoi = sigmoid(np.dot(X_array, theta)) # Se debe cambiar esto
            print("Características: ", X_array[1]) y (X_array[2]) y (X_array[3]) y (X_array[4]) y (X_array[5]) y (X_array[6]) y (X_array[7])
theta calculado por el descenso por el gradiente: [ 4.76473855e-01 -1.17496835e-05 1.33212786e-02 -7.52951640e-04
6.3974397e-03 3.7030589e-03 8.84378807e-03 3.82588146e-03
1.05287972e-02 -4.98858631e-03 6.03617269e-03]

```

```

[91] def costFunction(theta, X, y):
    # Inicializar algunos valores utiles
    m = y.size # numero de ejemplos de entrenamiento

    J = 0
    grad = np.zeros(theta.shape)

    h = sigmoid(X.dot(theta.T))

    J = (1 / m) * np.sum(-(y.dot(np.log(h)) - (1 - y).dot(np.log(1 - h))))
    grad = (1 / m) * (h - y).dot(X)

    return J, grad

```

```

[92] # Inicialización de parámetros de ajuste
initial_theta = np.zeros(11)
print(initial_theta)
cost, grad = costfunction(initial_theta, X, y)

print('Costo en theta inicial (zeros): {:.3f}'.format(cost))
print('Costo esperado (aproximado): 0.093\n')
print(grad)
print('Gradiente en theta inicial (zeros):')
print('  |t|{:.4f}, {:.4f}, {:.4f}]\n'.format(grad))
print('Gradiente esperado (aproximado):\n|t|{-0.1080, -12.0092, -11.2628]\n')

[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
Costo en theta inicial (zeros): 0.093
Costo esperado (aproximado): 0.093

[-1.1693086e-01 -2.4604595e-05 2.99632256e-03 1.50735953e-04
 -1.32236079e-03 -9.2349852e-04 1.22021997e-03 8.94560815e-04
 -2.32845906e-03 1.17102656e-03 1.23932656e-03]

Gradiente en theta inicial (zeros):
[-0.1169, -0.0000, -0.0030]
Gradiente esperado (aproximado):
[-0.1080, -12.0092, -11.2628]

[93] # Calcula y muestra el costo y el gradiente con valores de theta diferentes a cero
test_theta = np.array([1,1,1,1,1,13,2,4,2,3,1,12])
#test_theta = np.array([-11.74749157, 0.09927308, 0.09316497])
print(test_theta)
cost, grad = costfunction(test_theta, X, y)

print('Costo en theta prueba: {:.12f}\n'.format(cost))
print('Costo esperado (aproximado): 0.218\n')

print('Gradiente en theta prueba:')
#print('|\t{:.12f}, {:.12f}, {:.12f}, {:.12f}, {:.12f}, {:.12f}, {:.12f}, {:.12f}, {:.12f}, {:.12f}, {:.12f}]\n'.format(grad))
print('Gradiente esperado (aproximado):\n|t|{0.043, 2.566, 2.647}\n')

[1 1 1 1 13 2 4 2 3 1 12]
Costo en theta prueba: nan
Costo esperado (aproximado): 0.218

Gradiente en theta prueba:
Gradiente esperado (aproximado):
[0.043, 2.566, 2.647]
<ipython-input-91:1a3804f56ce>:10: RuntimeWarning: divide by zero encountered in log
J = (1 / m) * np.sum(-y.dot(np.log(h)) - (1 - y).dot(np.log(1 - h)))

[94] # Establecer las opciones para optimize.minimize
options= {'maxiter': 1000}

# revisar la documentacion de scipy's optimize.minimize para mayor descripcion de los parametros
# La funcion devuelve un objeto 'OptimizeResult'
# Se utiliza el algoritmo de Newton truncado para la optimizacion.

res = optimize.MinimizeResult(
    initial_theta,
    (X, y),
    jac=True,
    method='TNC',
    options=options)

# la propiedad fun del objeto devuelto por `OptimizeResult`
# contiene el valor del costfunction de un theta optimizado
cost = res.fun

# Theta optimizada esta en la propiedad x
theta = res.x

# Imprimir theta en la pantalla
print('Costo con un valor de theta encontrado por optimize.minimize: {:.12f}\n'.format(cost))
print('Costo esperado (aproximado): 0.203\n');

print('theta:')
#print('|\t{:.12f}, {:.12f}, {:.12f}, {:.12f}, {:.12f}, {:.12f}, {:.12f}, {:.12f}, {:.12f}, {:.12f}, {:.12f}]\n'.format(theta))
print('Theta esperado (aproximado):\n|t|{-25.161, 0.206, 0.201, 0.206, 0.201, 0.206, 0.201, 0.206, 0.201, 0.206}\n')

Costo con un valor de theta encontrado por optimize.minimize: 0.66550530957
Costo esperado (aproximado): 0.203

theta:
Theta esperado (aproximado):
[-25.161, 0.206, 0.201, 0.206, 0.201, 0.206, 0.201, 0.206, 0.201, 0.206]

[94]

[95] ## predict(theta, X):

    m = X.shape[0] # Número de ejemplo de entrenamiento
    p = np.zeros(m)
    p = np.round(sigmoid(X.dot(theta.T)))
    return p

[98] #Predice la probabilidad de mano de poker de 5 cartas de 9 tipos de baraja en la mano
#Entre mayor porcentaje mayor sera la probabilidad de ganar el mano de poker entre cero al 100%
prob = sigmoid(np.dot([1,1,1,1,13,2,4,2,3,1,12], theta))
#print('La probabilidad de ganar la mano de poker es: {:.2f}%\n'.format(prob))
print('Valor esperado: 0.775 +/- 0.002\n')

# Computación de entrenamiento
p = predict(theta, X)
print('Precisión de entrenamiento: {:.2f} %.'.format(np.mean(p == y) * 100))
print('Entre mayor porcentaje mayor sera la probabilidad de ganar mi mano de poker entre cero al 100%')

Valor esperado: 0.775 +/- 0.002
Precisión de entrenamiento: 42.25 %
Entre mayor porcentaje mayor sera la probabilidad de ganar mi mano de poker entre cero al 100%

```

Archivos



+ Código + Texto

RAM Disco Editando

{x} sample_data

 README.md
 anscombe.json
 california_housing_test.csv
 california_housing_train.csv
 mnist_test.csv
 mnist_train_small.csv
 poker-hand-testing7.csv

<>

Disco 84.14 GB disponibles

Universitario: Ceron Beimar Miguel

Carrera: Ingenieria de Sistemas

Datasheet: Conjunto de datos del juego Poker

<https://www.kaggle.com/datasets/hosseinah1/poker-game-dataset>

Datasheet usado es poker-hand-testing.csv

Repositorio GitHub:

<https://github.com/Beimar98/SIS420>

Información de atributos: Variables "X": 1) S1 "Palo de la carta n.º 1" ordinal (1-4) que representa {Corazones, Picas, Diamantes, Tréboles}

2) C1 "Rank of card #1" Numérico (1-13) que representa (As, 2, 3, ..., Queen, King)

3) S2 "Palo de la carta #2" Ordinal (1-4) que representa {Corazones, Picas, Diamantes, Tréboles}

4) C2 "Rank of card #2" Numérico (1-13) que representa (As, 2, 3, ..., Queen, King)

5) S3 "Palo de la carta #3" Ordinal (1-4) que representa {Corazones, Picas, Diamantes, Tréboles}

6) C3 "Rank of card #3" Numérico (1-13) que representa (As, 2, 3, ..., Queen, King)

7) S4 "Palo de la carta #4" Ordinal (1-4) que representa {Corazones, Picas, Diamantes, Tréboles}

8) C4 "Rank of card #4" Numérico (1-13) que representa (As, 2, 3, ..., Queen, King)

9) S5 "Palo de la carta #5" Ordinal (1-4) que representa {Corazones, Picas, Diamantes, Tréboles}

10) C5 "Rank of card 5" Numérico (1-13) que representa (As, 2, 3, ..., Queen, King)

Variables "Y": 11) CLASE "Mano de Poker" Ordinal Tipos(0-9)

0: Nada en la mano; no es una mano de póquer reconocida 1: Un par; un par de rangos iguales dentro de cinco cartas 2: dos pares; dos pares de rangos iguales dentro de cinco cartas 3: Trío; tres rangos iguales dentro de cinco cartas 4: Escalera; cinco cartas, clasificadas secuencialmente sin espacios 5: Color; cinco cartas del mismo palo 6: Full; par + rango diferente tres de una clase 7: Cuatro de una clase; cuatro rangos iguales dentro de cinco cartas 8: Escalera de color; escalera + color o tambien valores de atributos faltantes: ninguno 9: escalera real; {As, Rey, Reina, Jota, Diez} + color

```
[5] import numpy as np
     import pandas as pd

# Configuración para las librerías
pd.options.mode.chained_assignment = None
np.set_printoptions(precision = 3)

# Se carga el archivo
data = pd.read_csv("/content/sample_data/poker-hand-testing7.csv")

print(data)
```

	Suit of Card 1	Rank of Card 1	Suit of Card 2	Rank of Card 2	\
0	1	1	1	13	
1	3	12	3	2	
2	1	9	4	6	
3	1	4	3	13	
4	3	10	2	7	
...
999995	3	1	1	12	
999996	3	3	4	5	
999997	1	11	4	7	
999998	3	11	1	8	
999999	2	5	2	9	

	Suit of Card 3	Rank of Card 3	Suit of Card 4	Rank of Card 4	\
0	2	4	2	3	
1	3	11	4	5	
2	1	4	3	2	
3	2	13	2	1	
4	1	2	2	11	
...
999995	2	9	4	9	
999996	2	7	1	4	
999997	3	9	1	13	
999998	1	1	3	13	
999999	4	9	2	3	

	Suit of Card 5	Rank of Card 5	Poker Hand
0	1	12	0
1	2	5	1
2	3	9	1
3	3	6	1
4	4	9	0
...
999995	2	6	1

```

999996      4      3      1
999997      2      7      1
999998      2      8      1
999999      3      3      2

```

[1000000 rows x 11 columns]

```

[6] # la normalizacion del data set entre la Media y la Estandar
def normalize(xs):
    xs_norm = xs.copy()
    mu = np.zeros(xs.size)
    sigma = np.zeros(xs.size)
    mu = np.mean(xs, axis = 0) # esta funcion saca la media
    sigma = np.std(xs, axis = 0) # esta funcion saca la estandar
    xs_norm = (xs - mu) / sigma
    print("Dataset normalizado")
    return xs_norm, mu, sigma

```

```

[7] print(data.shape)

y = data[data.columns[10]] # asignacion a la varia y la predicion
x = data.iloc[:, 0:10] # asignacion de las variables de x1 x2, x3.....x11

y = np.array(y)

x, mu, sigma = normalize(x) # llamdoa a la funcion de normalizacion
x = np.concatenate([np.ones((x.shape[0], 1)), x], axis=1) # estas concatenado los 1 al inicio a x0

print(x.shape)
print(y.shape)

print(x)
print(y)

[8] (1000000, 11)
Dataset normalizado
(1000000, 11)
(1000000,)
[[ 1.   -1.342 -1.602 ... -1.07  -1.34   1.34 ]
 [ 1.    0.447  1.336 ... -0.535 -0.446 -0.532]
 [ 1.   -1.342  0.535 ... -1.337  0.447  0.538]
 ...
 [ 1.   -1.342  1.069 ...  1.603 -0.446  0.003]
 [ 1.    0.447  1.069 ...  1.603 -0.446  0.27 ]
 [ 1.   -0.448 -0.534 ... -1.07   0.447 -1.067]]
[0 1 1 ... 1 1 2]

```

```

[8] def relu(x):
    return np.maximum(0, x) # funcion de activacion de relu conbierte los valores negativos en 0 y los positivos los deja tal

def reluPrime(x):
    return x > 0 # la derivada de la funcion de Relu

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def linear(x):
    return x

def softmax(x):
    return np.exp(x) / np.exp(x).sum(axis=-1,keepdims=True)

def crossentropy(y, y_hat): # esta funcion el para la Clasificacion multiclass que es 0 1 2 3 4 5 6 7 8 9 10
    logits = y_hat[np.arange(len(y_hat)),y]
    entropy = - logits + np.log(np.sum(np.exp(y_hat),axis=-1))
    return entropy.mean()

def grad_crossentropy(y, y_hat): # esta funciojn es la deriba de funcion de Crossentropy
    answers = np.zeros_like(y_hat)
    answers[np.arange(len(y_hat)),y] = 1
    return (- answers + softmax(y_hat)) / y_hat.shape[0]

```

```

[9] class MLP():
    def __init__(self, D_in, H, D_out, loss, grad_loss, activation):
        # pesos de la capa 1
        self.w1, self.b1 = np.random.normal(loc=0.0,
                                             scale=np.sqrt(2/(D_in+H)),
                                             size=(D_in, H)), np.zeros(H)
        # pesos de la capa 2
        self.w2, self.b2 = np.random.normal(loc=0.0,
                                             scale=np.sqrt(2/(H+D_out)),
                                             size=(H, D_out)), np.zeros(D_out)
        self.ws = []
        # funcion de perdida y derivada
        self.loss = loss
        self.grad_loss = grad_loss
        # funcion de activacion
        self.activation = activation

    def __call__(self, x):
        # salida de la capa 1
        self.h_pre = np.dot(x, self.w1) + self.b1
        self.h = relu(self.h_pre)
        # salida del MLP
        y_hat = np.dot(self.h, self.w2) + self.b2
        return self.activation(y_hat)

```

```

def fit(self, X, Y, epochs = 100, lr = 0.001, batch_size=None, verbose=True, log_each=1):
    batch_size = len(X) if batch_size == None else batch_size
    batches = len(X) // batch_size
    l = []
    for e in range(1,epochs+1):
        # Mini-Batch Gradient Descent
        _l = []
        for b in range(batches):
            # batch de datos
            x = X[b*batch_size:(b+1)*batch_size]
            y = Y[b*batch_size:(b+1)*batch_size]
            # salida del perceptrón
            y_pred = self.f(x)
            # función de pérdida
            loss = self.loss(y, y_pred)
            _l.append(loss)
        # Backprop
        dldy = self.grad_loss(y, y_pred)
        grad_w2 = np.dot(self.h.T, dldy)
        grad_b2 = dldy.mean(axis=0)
        dldh = np.dot(dldy, self.w2.T)*reluPrime(self.h_pre)
        grad_w1 = np.dot(x.T, dldh)
        grad_b1 = dldh.mean(axis=0)
        # Update (GD)
        self.w1 = self.w1 - lr * grad_w1
        self.b1 = self.b1 - lr * grad_b1
        self.w2 = self.w2 - lr * grad_w2
        self.b2 = self.b2 - lr * grad_b2
        l.append(np.mean(_l))
    # guardamos pesos intermedios para visualización
    self.ws.append(
        self.w1.copy(),
        self.b1.copy(),
        self.w2.copy(),
        self.b2.copy()
    )
    if verbose and not e % log_each:
        print(f'Epoch: {e}/{epochs}, Loss: {np.mean(l):.5f}')

def predict(self, ws, x):
    w1, b1, w2, b2 = ws
    h = relu(np.dot(x, w1) + b1)
    y_hat = np.dot(h, w2) + b2
    return self.activation(y_hat)

```

```

[10] # MLP para clasificación multiclas
class MLPClassification(MLP):
    def __init__(self, D_in, H, D_out):
        super().__init__(D_in, H, D_out, crossentropy, grad_crossentropy, linear)

```

```
[10]
```

```

[11] model = MLPClassification(D_in = 11, H = 50, D_out = 10) # datos entrada D_in 11   Neuronas H =50   datos de Salida D_10
epochs, lr = 100, 0.01
model.fit(x, y, epochs, lr, batch_size = 500, log_each = 50)

```

```

Epoch: 50/100, Loss: 0.93189
Epoch: 100/100, Loss: 0.88948

```

```
[12] import random
```

```

ws = model.ws[-1]

p1 = random.randint(0, 100000) # estas escogiendo un dato aleatorio de los 100000 datos
x1 = x[p1,:]
pred1 = model.predict(ws, x1) # estas haciendo predecir enciendole las Pesos ws y el dato escogido aleatoriamente
pred1 = np.argmax(softmax(pred1)) # Esta prediciono con el softmax
real1 = y[p1] # obteniendo el valor real que tiene la y que escogimos
print(f'El valor real es {real1}')
print('Prediccion de la red neuronal: {}'.format(pred1))

p2 = random.randint(0, 100000) # estas escogiendo un dato aleatorio de los 100000 datos
x2 = x[p2,:]
pred2 = model.predict(ws, x2)
pred2 = np.argmax(softmax(pred2))
real2 = y[p2]
print(f'El valor real es {real2}')
print('Prediccion de la red neuronal: {}'.format(pred2))

p3 = random.randint(0, 100000) # estas escogiendo un dato aleatorio de los 100000 datos
x3 = x[p3,:]
pred3 = model.predict(ws, x3)
pred3 = np.argmax(softmax(pred3))
real3 = y[p3]
print(f'El valor real es {real3}')
print('Prediccion de la red neuronal: {}'.format(pred3))

```

```

El valor real es 0
Prediccion de la red neuronal: 0
El valor real es 0
Prediccion de la red neuronal: 1
El valor real es 0
Prediccion de la red neuronal: 1

```

Productos pagados de Colab - Cancela los contratos aquí

✓ 0 s se ejecutó 16:26

