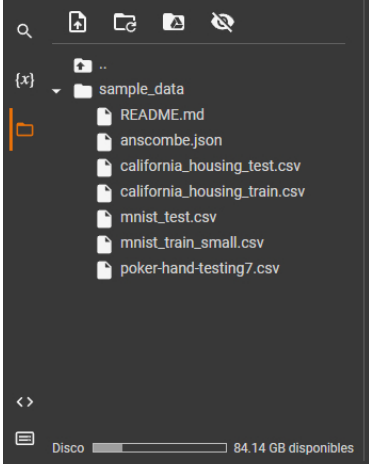

Universitario: Ceron Beimar Miguel

Carrera: Ingenieria de Sistemas

Datashet: Conjunto de datos del juego Poker
https://www.kaggle.com/datasets/hosseinah1/poker-game-dataset

Datashet 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.° 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

    # Configuracion para las librerias
    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
```

```
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)
```

```
(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 funciopn 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 = []
         # función de pérdida y derivada
         self.loss = loss
         self.grad_loss = grad_loss
         # función de activación
         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(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 multiclase
     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 hacioedo predecir enciadole las Pesos ws y el dato escogido aleatoriamente
     pred1 = np.argmax(softmax(pred1)) # Esta prediciondo con el softmax
     real1 = y[p1]  # onteniendo el valor real que tiene la y que escogimos
     print(f'El valor real es {real1}')
     print('Predicción 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('Predicción 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('Predicción de la red neuronal: {}'.format(pred3))
```

```
El valor real es 0
Predicción de la red neuronal: 0
El valor real es 0
Predicción de la red neuronal: 1
El valor real es 0
Predicción de la red neuronal: 1
```


Productos pagados de Colab - Cancela los contratos aquí
0 s se ejecutó 16:26