Sheet

Proyecto 1.

Universidad: UTEC
Curso: Inteligencia Artificial-CS2601
Profesor: Cristian López Del Alamo
Tema: Regresión y *machine learning* para predicción de áreas de incendio forestal.
Integrantes:

- Sebastián Lizárraga
- Carlos Guerrero

1. En esta práctica se pide realizar pruebas utilizando diferentes funciones de pérdida.
2. Su equipo debe implementar el algoritmo de machine learning para regresión lineal múltiple y realizar las correspondientes pruebas usando el siguiente Dataset.
3. MSE Loss Function \

$MSE = \frac{1}{2m} \sum_{i=0}^{m} (y_i - h(x_i))^2$

4. Utilize todo los datos del dataset para entrenar y grafique el plano que mejor separa a los datos. Help

Importante: No se olvide de normalizar los datos entre cero y uno, por cada columna.

Crear el DataSet

Data set info: https://www.kaggle.com/datasets/elikplim/forest-fires-data-set

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Normalización

```
def normalice(df_input):
    z = pd.DataFrame(df_input)
    # Normalizacion con media y desviacion estandar
    return z.apply(lambda x: (x-x.mean())/ x.std(), axis=0)
    # Normalizacion con minimos y maximos
    # return (z - z.min()) / ( z.max() - z.min())
```

```
dataset = pd.read_csv('forestfires.csv')
X = dataset[["RH","temp"]]
Y = dataset[["area"]]
# Normalizamos la data
X = normalice(X)
Y = normalice(Y)
# Luego de normalizar agregamos la columna del bias
X = np.concatenate([np.ones((X.shape[0], 1)), X], axis=1)

#print(X)
#print(Y)
```

Modelo

Nota: Antes añadir Añadir una columna de nx1 a X con valor 1.

$h(X) = X * W^t$

```
def h(X, W):
    return np.dot(X,np.transpose(W))
```

Loss function

Nota: La función de pérdida no cambia, solo la llamada a la función h $\mathcal{L} = ||Y - XW^t||_2^2$

Norma L2

```
def Error(X, W, Y, rho, lam):
    ridge_component = (1- rho)*lam*np.sum(np.power(W, 2))
    lasso_component = rho*(lam*np.sum(np.abs(W)))
    n = X.shape[0]
    normal_L2 = np.sum(np.power(Y - h(X, W), 2)) / (2 * n)
    loss = normal_L2 + ridge_component + lasso_component
    return normal_L2
```

Cálculo de derivadas

Nota: Intente resolver este algoritmo desde un punto de vista matricial.

$dw_j = \frac{1}{m}\sum_{i=0}^{m}(y_i - h(x_i))(-x_{i,j})$

```
def derivada(X, W, Y, lam, rho):
    n = X.shape[0]
    normal_L2_derivate = np.dot((Y - h(X,W)).transpose(), (-1 * X)) / n
    ridge_derivate = 2*lam*(1-rho)*W
    lasso_derivate = np.sign(W)*rho*lam
    dW = normal_L2_derivate  + ridge_derivate + lasso_derivate
    return normal_L2_derivate
```

Actualiación de parámetros

Recuerde: $\frac{\partial L}{\partial w}$ representa un vector con todas las derivadas de la función de pérdida con rescto a W.

$W = W - \alpha * \frac{\partial L}{\partial W}$

```
def update(W,  dW, alpha):
    return W - alpha * dW
```

Training

```
def train(X, Y, umbral, alfa):
    # Creamos un vector fila W aleatoria con el tamaño de columnas de X
    X_columns = X.shape[1]
    W = np.random.rand(1,X_columns)
    rho = 0
    lam = 10
    L = Error(X,W,Y,rho, lam)
    #print(L[0])
    loss = []
    while (L[0] > umbral):
        dW = derivada(X, W, Y, lam, rho)
        W = update(W, dW, alfa)
        L = Error(X, W,Y, rho, lam)
        #print(W)
        #print("--------------")
        loss.append(L)
    return W, loss


def Plot_Loss(epochs,loss):
   plt.plot(epochs, loss)
```

Plot

```
# import numpy as np
# import matplotlib.pyplot as plt

# plt.rcParams["figure.figsize"] = [7.00, 3.50]
# plt.rcParams["figure.autolayout"] = True

# x = np.linspace(-10, 10, 100)
# y = np.linspace(-10, 10, 100)

# x, y = np.meshgrid(x, y)

# eq = 0.1 * x + 40 * y + 100.09  # ecuacion del plano: x_1w_1 + x_2w_2 + b

# fig = plt.figure()

# ax = fig.gca(projection='3d')

# ax.plot_surface(x, y, eq)

# plt.show()
```

Testing

```python
def plot3D(w_0, w_1, w_2, X, Y):
    plt.rcParams["figure.figsize"] = [7.00, 3.50]
    plt.rcParams["figure.autolayout"] = True

    x = np.linspace(-10, 10, 100)
    y = np.linspace(-10, 10, 100)

    x, y = np.meshgrid(x, y)

    eq = 0.1 * w_0 + 40 * w_1 + w_2 # ecuacion del plano: x_1w_1 + x_2w_2 + b

    fig = plt.figure()

    ax = fig.gca(projection='3d')

    ax.plot_surface(x, y, eq)

    plt.show()
```

```python
umbral = 0.25
alpha = 2

W = train(X, Y, umbral, alpha)
#plot3D(W[0],W[1],W[2],X,Y)
```

```
<ipython-input-16-d4a31dd5c6b9>:2: RuntimeWarning: overflow encountered in do
  ridge_component = (1- rho)*lam*np.sum(np.power(W, 2))
<ipython-input-16-d4a31dd5c6b9>:2: RuntimeWarning: overflow encountered in po
  ridge_component = (1- rho)*lam*np.sum(np.power(W, 2))
<ipython-input-16-d4a31dd5c6b9>:3: RuntimeWarning: invalid value encountered
  lasso_component = rho*(lam*np.sum(np.abs(W)))
```