UNIVERSIDAD MAYOR REAL Y PONTIFICIA DE SAN FRANCISCO XAVIER DE CHUQUISACA

Informe Laboratorio 10

Regresión Logística Multiclase One vs All

UNIVERSITARIA: Ortube Rengel Erika Mariana

CARRERA: Ingeniería de Sistemas

MATERIA: Inteligencia Artificial I (SIS420)

DOCENTE: Ing. Carlos W. Pacheco Lora

SUCRE – BOLIVIA

Regresión Logística Multiclase *One vs All*

Se recopilaron datos de un *dataset* sobre *Fashion MNIST* que contiene conjunto de datos de imágenes de artículos de Zalando; consta de un conjunto de entrenamiento ‘train’ de 60000 ejemplos y un conjunto de prueba ‘test’ de 10 000 ejemplos.

Lectura de datos

```
#Ingresar imágenes de ropa/accesroioa de 28x28
input_layer_size  = 784

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

#datos de entrenamiento almacenados en los arreglos X, y
data = np.loadtxt('/content/drive/MyDrive/fashion-mnist_train.csv', delimiter=',', skiprows=1)

X, y = data[:,1:], data[:,0].ravel()
print(y)

#establecer el dígito cero en 0, en lugar del 10 asignado a este conjunto de datos
y[y == 10] = 0
print(y)

m = y.size
```

```
[2. 9. 6. ... 8. 8. 7.]
[2. 9. 6. ... 8. 8. 7.]
```

```
print(X)
print(y)
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
[2. 9. 6. ... 8. 8. 7.]
```

Definición de función para mostrar los datos en 2D

```
#definicion de funcion que muestra datos 2D almacenados en 'X', en una cuadrícula apropiada
def displayData(X, example_width=None, figsize=(10, 10)):
    #calcula las filas y las columnas
    if X.ndim == 2:
        m, n = X.shape
    elif X.ndim == 1:
        n = X.size
        m = 1
        #proporciona a una matriz bidimensional
        X = X[None]
    else:
        raise IndexError('La entrada X debe ser 1 o 2 dimenosinal.')

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

    #calcula el numero de elementos que se van 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')
```

```
#muestra la dimension de 'X'
X.ndim
```

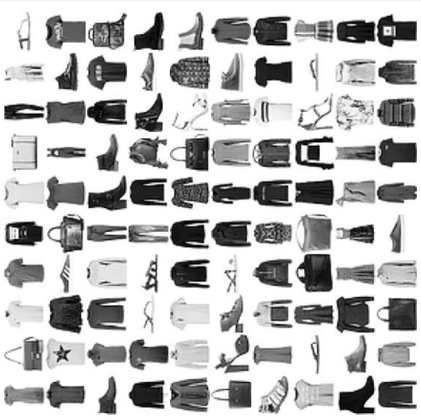
```
2
```

```
#selecciona aleatoriamente 100 puntos de datos para mostrar

#genera un array de 100 puntos diferentes (sin que se repitan)
rand_indices = np.random.choice(m, 100, replace=False)

#obtiene el valor de 'X' en cada posicion que ha sido generada en 'rand_indices'
sel = X[rand_indices, :]

#muestra una imagen en 2D de los puntos aleatoriamente seleccionados
displayData(sel)
```



Para probar la regresión logística vectorizada

```
#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
```

Definición de función para calcular la sigmoide

```
#definicion de funcion que calcula la sigmoide de 'z'
def sigmoid(z):
    return 1.0 / (1.0 + np.exp(-z))
```

Definición de función que calcula el costo de usar theta como parámetro

```
#defincion de funcion que calcula el costo de usar theta como parámetro
#para la regresión logística regularizada y el gradiente del costo w.r.t. a los parámetros.
def lrCostFunction(theta, X, y, lambda_):
    """
     Parametros
        theta: array_like
            Parametro theta de la regresion logistica. Vector de la forma(shape) (n, ).
            n es el numero de caracteristicas incluida la intercepcion

        X: array_like
            Dataset con la forma(shape) (m x n); m es el numero de ejemplos
            n es el numero de caracteristicas incluida la intercepcion

        y: array_like
            Conjunto de etiquetas. Un vector con la forma (shape) (m, ).
            m es el numero de ejemplos

        lambda_: float
            Parametro de regularización.

     Devuelve
        J: float
            El valor calculado para la funcion de costo regularizada.

        grad: array_like
            Un vector de la forma (shape) (n, ) que es el gradiente de la
            función de costo con respecto a theta, en los valores actuales de theta.
    """
    #inicializamos algunos valores utiles
    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
```

```
J, grad = lrCostFunction(theta_t, X_t, y_t, lambda_t)

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

```
Costo: 3.085728
Costo esperado: 2.534819
-----------------------
Gradientes:
 [0.355376, -0.491709, 0.885979, 1.663668]
Gradientes esperados:
 [0.146561, -0.548558, 0.724722, 1.398003]
```

Definición de función para la regresión logística multiclase "One vs All"

```python
#defincion de funcion para la regresion logistica multiclase "One vs All"
#que entrena num_labels clasificadores de regresión logística y devoluciones
#cada uno de estos clasificadores en una matriz all_theta, donde la i-ésima fila de all_theta corresponde al clasificador para la etiqueta i.
def oneVsAll(X, y, num_labels, lambda_):
    """
    Parametros
        X: array_like
            El conjunto de datos de entrada de forma (m x n); m es el número de ejemplos
            n es el número de características.

        y: array_like
            Las etiquetas de datos. Un vector de forma (m, ).

        num_labels: int
            Número de etiquetas posibles.

        lambda_: float
            El parámetro de regularización logística.

    Devuelve
        all_theta: array_like
            Los parámetros entrenados para la regresión logística para cada clase.
            Esta es una matriz de forma (K x n+1) donde K es el número de clases
            (es decir, num_labels) y n es el número de características sin el sesgo.
    """
    #algunas variables utiles
    m, n = X.shape

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

    #agrega 1's 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
```

```python
lambda_ = 0.1
all_theta = oneVsAll(X, y, num_labels, lambda_)
print(all_theta.shape)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: RuntimeWarning: overflow encountered in exp
  This is separate from the ipykernel package so we can avoid doing imports until
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:44: RuntimeWarning: divide by zero encountered in log
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: RuntimeWarning: overflow encountered in exp
  This is separate from the ipykernel package so we can avoid doing imports until
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:44: RuntimeWarning: divide by zero encountered in log
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: RuntimeWarning: overflow encountered in exp
  This is separate from the ipykernel package so we can avoid doing imports until
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:44: RuntimeWarning: divide by zero encountered in log
(10, 785)
```

Definición de función que devuelve un vector de predicciones para "One vs All"

```python
#definicion de funcion que devuelve un vector de predicciones para cada ejemplo en la matriz X
#tener en cuenta que 'X' contiene los ejemplos en filas.
#all_theta es una matriz donde la i-ésima fila es un vector theta de regresión logística entrenada para la i-ésima clase
#se debe establecer p en un vector de valores de 0..K-1 (por ejemplo, p = [0, 2, 0, 1]; predice clases 0, 2, 0, 1 para 4 ejemplos).
#
def predictOneVsAll(all_theta, X):
    """
      Parametros
          all_theta: array_like
              Los parámetros entrenados para la regresión logística para cada clase.
              Esta es una matriz de forma (K x n+1) donde K es el número de clases
              y n es el número de características sin el sesgo.

          X: array_like
              Puntos de datos para predecir sus etiquetas. Esta es una matriz de forma (m x n);
              m es el número de puntos de datos para predecir, y n es el número de características sin el término de sesgo.
              Tener en cuenta que se agrega el término de sesgo para 'X' en esta función.

      Devuelve
          p: array_like
              Las predicciones para cada punto de datos en 'X'. Este es un vector de forma (m, ).
    """
    m = X.shape[0];
    num_labels = all_theta.shape[0]

    p = np.zeros(m)

    #agrega 1's a la matriz de datos 'X'
    X = np.concatenate([np.ones((m, 1)), X], axis=1)
    p = np.argmax(sigmoid(X.dot(all_theta.T)), axis = 1)

    return p
```

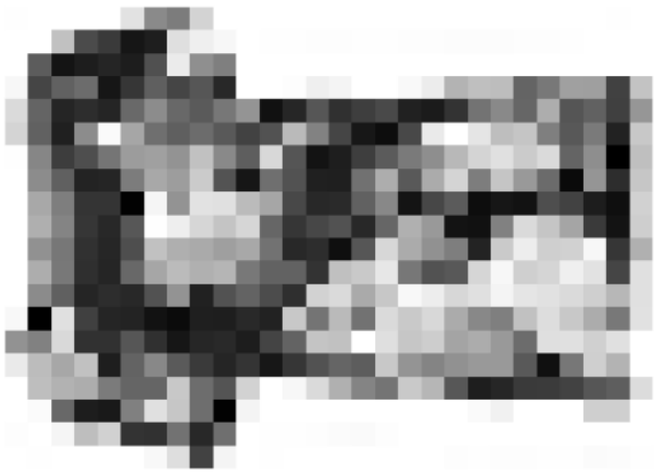
Con los datos se entrenamiento 'train' se logró obtener una precisión de 85.33%

```
print(X.shape)
pred = predictOneVsAll(all_theta, X)
print('Precision del conjuto de entrenamiento: {:.2f}%'.format(np.mean(pred == y) * 100))
XPrueba = X[1002:1003, :].copy()
print(XPrueba.shape)

XPrueba = np.concatenate([np.ones((1, 1)), XPrueba], axis=1)
print(XPrueba.shape)
p = np.argmax(sigmoid(XPrueba.dot(all_theta.T)), axis = 1)
print(p)

displayData(X[1002:1003, :])
```

```
(60000, 784)
Precision del conjuto de entrenamiento: 85.53%
(1, 784)
(1, 785)
[0]
```



Con los datos se prueba 'test se logró obtener una precisión de 88.06%

```
print(X.shape)
pred = predictOneVsAll(all_theta, X)
print('Precision del conjuto de entrenamiento: {:.2f}%'.format(np.mean(pred == y) * 100))
XPrueba = X[1002:1003, :].copy()
print(XPrueba.shape)

XPrueba = np.concatenate([np.ones((1, 1)), XPrueba], axis=1)
print(XPrueba.shape)
p = np.argmax(sigmoid(XPrueba.dot(all_theta.T)), axis = 1)
print(p)

displayData(X[1002:1003, :])
```

```
(10000, 784)
Precision del conjuto de entrenamiento: 88.06%
(1, 784)
(1, 785)
[4]
```