

Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Estado de México

Inteligencia Artificial Avanzada I. Módulo 2

Implementación de una técnica de aprendizaje máquina sin el uso de un framework.

Enrique Maldonado Chavarría

Profesor:

Dr. Jorge Adolfo Ramirez Uresti

Fecha de entrega:

5 de septiembre del 2022

Bitácora:

Decidí implementar la técnica de <u>regresión logística</u> mejorando los parámetros con descenso del gradiente total.

Elegí el <u>dataset de semillas de calabaza extraído de Kaggle</u> el cuál cuenta con 12 columnas numéricas que representan las características y medidas de las semillas de diversas calabazas y una columna de clase qué indica el tipo de calabaza al que pertenece la semilla analizada.

Esta columna de clase será nuestra variable a predecir contando únicamente con dos salidas posibles:

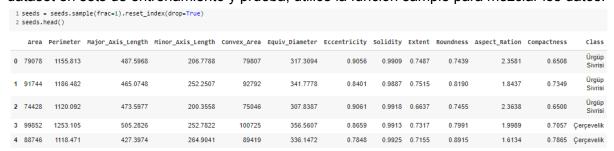
- Calabaza tipo Çerçevelik
- Calabaza tipo Ürgüp Sivrisi

Al tener solamente dos clases posibles como resultado, este dataset es adecuado para trabajar con regresión logística una vez transformadas las variables categóricas a numéricas.



Tras cargar el dataset, con uso de la librería de Pandas, el primer reto con el que me encontré fue que los datos estaban ordenados de forma que la primera mitad de estos pertenecían a la clase "Ürgüp Sivrisi" y la segunda mitad a la clase "Çerçevelik".

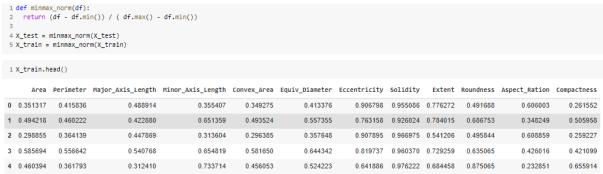
Para evitar que esto representara un sesgo en la información al momento de dividir nuestro dataset en sets de entrenamiento y prueba, utilicé la función sample para mezclar los datos.



Tras separar el dataset entre las variable de entrada y las de salida el segundo reto que resolví fue la división en sets de entrenamiento y prueba sin el uso de librerías especializadas.

```
1 test_size=0.2
2 #X_train = X.sample(frac=0.8)
3
4 X_train = X[:int(X.shape[0]*(1-test_size))]
5 X_test = X[int(X.shape[0]*(1-test_size)):]
6 y_train = y[:int(X.shape[0]*(1-test_size))]
7 y_test = y[int(X.shape[0]*(1-test_size)):]
8
9 print(X_train.shape,y_train.shape, X_test.shape, y_test.shape)
(2000, 12) (2000, 1) (500, 12) (500, 1)
```

Decidí normalizar los datos después de este procedimiento para evitar el *data leakage*. Normalice los datos siguiendo la estrategia de "min-max" manualmente.



Terminado el preprocesamiento de los datos decidí desarrollar mi versión de la regresión logística creando diversas funciones para cada paso de la misma con apoyo de la librería numpy para la parte matemática.

```
1 import numpy as np
   0
(210] 1 def perceptron(X_train, params):
        2 YS=[]
        3 for i in X_train.index:
            vector_x = [1]
            for col in X_train.columns:
             vector_x.append(X_train.loc[i,col])
           vector_x = np.array(vector_x)
        8 YS.append((vector_x*params).sum())
        9 return np.array(YS)

√ [211] 1 def sigmoide(YS):
        2 SIGS=[]
        3 for ys in YS:
            SIGS.append(1/(1+np.exp(ys*-1)))
        5 y_pred=[round(e) for e in SIGS]
        6 return np.array(SIGS), y_pred
```

Procedí de igual forma para calcular la función de pérdida con *cross-entropy* e ir mejorando los parámetros de forma iterativa con descenso del gradiente completo.

```
[212] 1 def crossentropy(y_train, SIG):
     2 cross=[]
     3 y_inds=y_train.index
     4 for n in range(len(SIG)):
          y = y_train.loc[y_inds[n],'Class']
          yp = SIG[n]
          ce= -(y*np.log(yp) + (1-y)*np.log(1-yp))
     8 cross.append(ce)
     10 loss = np.mean(np.array(cross))
     11 return loss
[213] 1 def gradient_descent(params, alfa, y_pred, loss):
     2 #print('Old parameters', params)
     3 for p in range(len(params)):
     4 params[p] = params[p]-(alfa/len(y_pred))*loss
     5 #print('New parameters', params)
     6 return params
```

Finalmente opté por usar *accuracy* cómo mi parámetro final de evaluación del modelo para poder comparar de forma clara las diferentes pruebas que se harán a continuación.

```
1 def accuracy(y_pred, y_train):
2    sum=0
3    for n in y_train.index:
4     if y_pred[n-y_train.index[0]] == y_train.loc[n,'Class']:
5        sum += 1
6    print('Accuracy:', sum/len(y_pred))
7    #return (sum/len(y_pred))
```

Una vez creadas las funciones para iniciar la regresión logística, elegí un alfa de 0.3 y utilicé la librería *random* para definir los pesos iniciales.

```
1 #Coeficiente de aprendizaje
2 alfa = 0.3

1 #Random initial parameters
2 from random import random
3
4 params =[]
5 for i in range(X_train.shape[1]+1):
6    params.append(random())
7
8 params = np.array(params)
9 params
```

Para el entrenamiento del modelo ocupé las funciones ya mostradas y la función de pérdida fue el criterio utilizado para definir cuándo finalizar las iteraciones.

En cuanto la función de pérdida creciera en comparación a la iteración anterior el entrenamiento se terminaría y finalmente se muestra en pantalla la cantidad de épocas transcurridas y el *accuracy* del modelo para la parte de entrenamiento.

```
1 # Entrenamiento del modelo
2 epoch=0
3 losses = [100]
4 while losses[-1] == min(losses):
5    YS = perceptron(X_train, params)
6    SIG, y_pred = sigmoide(YS)
7    loss = crossentropy(y_train, SIG)
8    losses.append(loss)
9    epoch +=1
10    #print('Epoch:', epoch)
11    #print('Loss:', loss)
12    params = gradient_descent(params, alfa, y_pred, loss)
13    print('Epochs:', epoch)
14    accuracy(y_pred, y_train)
```

Comparación de pruebas

- 1. Primera prueba
 - a. Alfa = 0.3 sin Acelerador GPU
 - b. 3257 épocas de mejora de parámetros
 - c. 59.55% de accuracy en la etapa de entrenamiento
 - d. 59.4% de accuracy en la etapa de prueba

Testing del modelo

```
[ ] 1 YS = perceptron(X_test, params)
2 SIG, y_pred = sigmoide(YS)
3 accuracy(y_pred, y_test)
Accuracy: 0.594
```

2. Segunda prueba

- a. Alfa = 0.3 con Acelerador GPU
- b. 3275 épocas de mejora de parámetros
- c. 51.3% de accuracy en la etapa de entrenamiento
- d. 48.6% de accuracy en la etapa de prueba

Epochs: 3275 Accuracy: 0.513

Testing del modelo

```
18] 1 YS = perceptron(X_test, params)
2 SIG, y_pred = sigmoide(YS)
3 accuracy(y_pred, y_test)
```

Accuracy: 0.486

3. Tercera prueba

- a. Alfa = 0.4 con Acelerador GPU
- b. 2509 épocas de mejora de parámetros
- c. 74.1% de accuracy en la etapa de entrenamiento
- d. 73.6% de accuracy en la etapa de prueba

Epochs: 2509 Accuracy: 0.741

Testing del modelo

```
[36] 1 YS = perceptron(X_test, params)
2 SIG, y_pred = sigmoide(YS)
3 accuracy(y_pred, y_test)
```

Accuracy: 0.736

4. Cuarta prueba

- a. Alfa = 0.5 con Acelerador GPU
- b. 1755 épocas de mejora de parámetros
- c. 29.4% de accuracy en la etapa de entrenamiento
- d. 21% de accuracy en la etapa de prueba

Epochs: 1755 Accuracy: 0.294

Testing del modelo

```
[54] 1 YS = perceptron(X_test, params)
2 SIG, y_pred = sigmoide(YS)
3 accuracy(y_pred, y_test)
```

Accuracy: 0.21

5. Quinta prueba

- a. Alfa = 0.4 sin Acelerador GPU
- b. 2702 épocas de mejora de parámetros
- c. 82.8% de accuracy en la etapa de entrenamiento
- d. 80.4% de accuracy en la etapa de prueba

Epochs: 2702 Accuracy: 0.828

Testing del modelo

```
[38] 1 YS = perceptron(X_test, params)
2 SIG, y_pred = sigmoide(YS)
3 accuracy(y_pred, y_test)

Accuracy: 0.804
```

Conclusiones:

Uno de los aprendizajes clave que me llevo es el entendimiento del mejoramiento de parámetros con descenso del gradiente completo y lo tardado aunque seguro que es esta estrategia. Cada corrida de las 5 pruebas tomó un promedio de 15 minutos en entrenar el modelo.