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 regresión logística mejorando los parámetros con descenso del gradiente total.

Elegí el dataset de semillas de calabaza extraído de Kaggle 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.

```
seeds['Class'].replace(['Çerçevelik', 'Ürgüp Sivrisi'],[0, 1,], inplace= True)
seeds.head()
```

	Area	Perimeter	Major_Axis_Length	Minor_Axis_Length	Convex_Area	Equiv_Diameter	Eccentricity	Solidity	Extent	Roundness	Aspect_Ration	Compactness	Class
0	79078	1155.813	487.5968	206.7788	79807	317.3094	0.9056	0.9909	0.7487	0.7439	2.3581	0.6508	1
1	91744	1186.482	465.0748	252.2507	92792	341.7778	0.8401	0.9887	0.7515	0.8190	1.8437	0.7349	1
2	74428	1120.092	473.5977	200.3558	75046	307.8387	0.9061	0.9918	0.6637	0.7455	2.3638	0.6500	1
3	99852	1253.105	505.2826	252.7822	100725	356.5607	0.8659	0.9913	0.7317	0.7991	1.9989	0.7057	0
4	88746	1118.471	427.3974	264.9041	89419	336.1472	0.7848	0.9925	0.7155	0.8915	1.6134	0.7865	0

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.

```
seeds = seeds.sample(frac=1).reset_index(drop=True)
seeds.head()
```

	Area	Perimeter	Major_Axis_Length	Minor_Axis_Length	Convex_Area	Equiv_Diameter	Eccentricity	Solidity	Extent	Roundness	Aspect_Ration	Compactness	Class
0	79078	1155.813	487.5968	206.7788	79807	317.3094	0.9056	0.9909	0.7487	0.7439	2.3581	0.6508	Ürgüp Sivrisi
1	91744	1186.482	465.0748	252.2507	92792	341.7778	0.8401	0.9887	0.7515	0.8190	1.8437	0.7349	Ürgüp Sivrisi
2	74428	1120.092	473.5977	200.3558	75046	307.8387	0.9061	0.9918	0.6637	0.7455	2.3638	0.6500	Ürgüp Sivrisi
3	99852	1253.105	505.2826	252.7822	100725	356.5607	0.8659	0.9913	0.7317	0.7991	1.9989	0.7057	Çerçevelik
4	88746	1118.471	427.3974	264.9041	89419	336.1472	0.7848	0.9925	0.7155	0.8915	1.6134	0.7865	Çerçevelik

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.

```
test_size=0.2
#X_train = X.sample(frac=0.8)

X_train = X[:int(X.shape[0]*(1-test_size))]
X_test = X[int(X.shape[0]*(1-test_size)):]
y_train = y[:int(X.shape[0]*(1-test_size))]
y_test = y[int(X.shape[0]*(1-test_size)):]

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.

```
def minmax_norm(df):
  return (df - df.min()) / ( df.max() - df.min())

X_test = minmax_norm(X_test)
X_train = minmax_norm(X_train)
```

```
X_train.head()
```

	Area	Perimeter	Major_Axis_Length	Minor_Axis_Length	Convex_Area	Equiv_Diameter	Eccentricity	Solidity	Extent	Roundness	Aspect_Ration	Compactness
0	0.351317	0.415836	0.488914	0.355407	0.349275	0.413376	0.906798	0.955086	0.776272	0.491688	0.606003	0.261552
1	0.494218	0.460222	0.422880	0.651359	0.493524	0.557355	0.763158	0.926024	0.784015	0.686753	0.348249	0.505958
2	0.298855	0.364139	0.447869	0.313604	0.296385	0.357648	0.907895	0.966975	0.541206	0.495844	0.608859	0.259227
3	0.585694	0.556642	0.540768	0.654819	0.581650	0.644342	0.819737	0.960370	0.729259	0.635065	0.426016	0.421099
4	0.460394	0.361793	0.312410	0.733714	0.456053	0.524223	0.641886	0.976222	0.684458	0.875065	0.232851	0.655914

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.

```
import numpy as np
```

```
[210]
def perceptron(X_train, params):
  YS=[]
  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)
    YS.append((vector_x*params).sum())
  return np.array(YS)
```

```
[211]
def sigmoide(YS):
  SIGS=[]
  for ys in YS:
    SIGS.append(1/(1+np.exp(ys*-1)))
  y_pred=[round(e) for e in SIGS]
  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)):
       5     y = y_train.loc[y_inds[n],'Class']
       6     yp = SIG[n]
       7     ce= -(y*np.log(yp) + (1-y)*np.log(1-yp))
       8     cross.append(ce)
       9
      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.

```
# Entrenamiento del modelo
epoch=0
losses = [100]
while losses[-1] == min(losses):
  YS = perceptron(X_train, params)
  SIG, y_pred = sigmoide(YS)
  loss = crossentropy(y_train, SIG)
  losses.append(loss)
  epoch +=1
  #print('Epoch:', epoch)
  #print('Loss:', loss)
  params = gradient_descent(params, alfa, y_pred, loss)
print('Epochs:', epoch)
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

```
3257
Loss: 0.6767172748849805

[0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0,
Accuracy: 0.5955
0.5955
```

Testing del modelo

```
[ ]  YS = perceptron(X_test, params)
     SIG, y_pred = sigmoide(YS)
     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.