

Instituto Tecnológico de Estudios Superiores de Monterrey



**Tecnológico
de Monterrey**

Inteligencia artificial avanzada para la ciencia de datos I (Gpo 101)

Profesores:

Jorge Adolfo Uresti

**Actividad: Momento de Retroalimentación:
Módulo 2 Implementación de una técnica de
aprendizaje máquina sin el uso de un
framework.**

Eric Manuel Navarro Martínez A01746219

1 de Septiembre de 2024

Comenzamos con las siguientes funciones de apoyo para que el código pueda funcionar de forma adecuada:

- normalization
 - Se encarga de normalizar los datos basándonos en la Z score, tome en cuenta esta normalización debido a que decrementa el impacto de los outliers en los datos esto ya que mi dataset contiene outliers considerables debido a las calificaciones que puede que aunque bajas hayan tenido un gran desempeño en taquilla y viceversa. (*Effects Of Normalization Techniques On Logistic Regression In Data Science, 2022*)

```
5 def normalization(data):
6     #This normalization utilizes Z-score
7     mean = np.mean(data, axis=0)
8     std = np.std(data, axis=0)
9     return (data-mean)/std
```

- sigmoid
 - Calcula la función sigmoideal, al igual que tiene unos stops que permite que no se haga un overflow al usar datos extremadamente grandes evita igualmente dividir entre 0
- r_squared
 - Calcula la r cuadrada usando la suma de cuadrados al igual
- confusion_matrix
 - Realiza los cálculos de accuracy, specificity, precision, recall, f1 score a través del cálculo de los true positives, true negatives, false positives y false negatives que puedan ser obtenidos de una predicción errónea o certera de una buena o mala película, al igual que tiene un rango de error.

```
25 def confusion_matrix(y_true, y_pred, threshold=0.5):
26     TP = 0 #true positive
27     TN = 0 #true negative
28     FP = 0 #false positive
29     FN = 0 #false negative
30
31     y_true = np.array(y_true)
32     y_pred = np.array(y_pred)
33
34     predicted = (y_pred >= threshold).astype(int)
35     #Check if the prediction is correct
36     TP = np.sum((predicted == 1) & (y_true == 1))
37     TN = np.sum((predicted == 0) & (y_true == 0))
38     FP = np.sum((predicted == 1) & (y_true == 0))
39     FN = np.sum((predicted == 0) & (y_true == 1))
40
41     accuracy = (TP+TN)/(TP+TN+FP+FN) if (TP+TN+FP+FN) > 0 else 0 #avoid dividing by zero and crashing the program
42     specificity = (TN)/(TN+FP) if (TN+FP) > 0 else 0 #avoid dividing by zero and crashing the program
43     precision = (TP)/(TP+FP) if (TP+FP) > 0 else 0 #avoid dividing by zero and crashing the program
44     recall = (TP)/(TP+FN) if (TP+FN) > 0 else 0 #avoid dividing by zero and crashing the program
45     F1 = 2*precision*recall/(precision+recall) if (precision+recall) > 0 else 0 #avoid dividing by zero and crashing the program
46
47     return {'True Positives: ': TP, 'True Negatives: ': TN,
48           'False Positives: ': FP, 'False Negatives: ': FN,
49           'Accuracy': accuracy, 'Specificity': specificity,
50           'Precision': precision, 'F1 Score': F1,
51           'Recall: ': recall}
52
```

- linear

- Realizamos a la muestra y parámetros un producto punto, evaluando una función lineal por x elementos de las muestras y de los parámetros

```
54 def linear(params, sample):
55     return np.dot(params, sample) # Apply dot function to the parameters and the sample
56
```

- binary_cross_entropy

- Permite que se calcule el error medio al momento de usar el logistical regression al igual que tienen un stop a números demasiado pequeños para que no se realicen operaciones como lo es el $\log(0)$

```
57 def binary_cross_entropy(y_true, y_pred):
58     y_pred = np.clip(y_pred, 1e-15, 1 - 1e-15) # Prevent log(0) issues
59     return -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))
60
```

- gradient_descent

- En la función de gradient descent hacemos diversos cálculos los cuales nos permiten en la longitud de los parámetros que enviamos y con un alfa el cual nos da pequeños saltos para estar mejorando de forma más precisa el mismo, estas predicciones toman la sigmoide nuevamente para encontrar el error predicho, y actualizarse continuamente.

```
61 def gradient_descent(params, samples, y, alfa):
62     adjusted = np.copy(params) # Doesn't modify the original array
63     for i in range(len(params)):
64         c = 0
65         for j in range(len(samples)):
66             prediction = sigmoid(linear(params, samples[j]))
67             error = prediction - y[j] #Compute the prediction error
68             c += error*samples[j][i] #C gets updated based on the error and the sample
69             adjusted[i] = params[i]-alfa*(1/len(samples))*c #Update the parameter
70     return adjusted
```

- logistic_regression_train:

- Entrenamos con el dataset de training para poder obtener los parámetros adecuados que sean capaces de crear un modelo que pueda predecir de la manera más precisa las películas y su calidad, para hacer esto tomamos los valores de nuestro dataset de training, el cual vamos a normalizar para poder crear mejores predicciones en el y_train el cual es la columna que queremos predecir.
- Esta función entonces usa el gradient descent en conjunto con la evaluación lineal a la cual le realizaremos su sigmoide para que sea logística, por último tomamos en cuenta el error el cual permite que la función esté activa siempre y cuando el error sea mayor a 0.000001.
- Aquí es donde se ven reflejadas las funciones ya que estará mejorando continuamente, por lo que guardamos el error actual y en qué época nos encontramos hasta que lleguemos al error menor; en este momento se considera el entrenamiento terminado y los parámetros obtenidos

```

73 def logistic_regression_train(x_train, y_train, alfa):
74     # Normalizing the data
75     normalized_x_train = normalization(x_train)
76     # normalized_x_test = normalization(x_test)
77
78     # Initial parameters
79     params = np.zeros(x_train.shape[1]) #Set the parameters to an array of zeros
80     previous_error = float('inf') #Set the previous error to infinity
81     epochs = 0
82
83     while True:
84         # Update the parameters using gradient descent
85         params = gradient_descent(params, normalized_x_train, y_train, alfa)
86
87         # Predict the values for the test set
88         y_pred_prob = [sigmoid(linear(params, sample)) for sample in normalized_x_train]
89
90         #Calculate the error
91         current_error = binary_cross_entropy(y_train, y_pred_prob)
92
93         # Print results for the current epoch
94         epochs += 1
95         print(f'Epoch: {epochs}')
96         print('Parameters:', params)
97         print('Current error:', current_error)
98         print('Previous error:', previous_error)
99         print('-----')
100
101         if abs(previous_error - current_error) < 0.000001:
102             print("Finished training")
103             break
104         previous_error = current_error
105
106     return params

```

-
- logistic_regression_valid
 - A diferencia del logistic regression de training esta vez si queremos calcular la matriz de errores ya que podemos hacer la comparativa una vez que recibimos los params de el logistic regression de training, este despliega estos datos para una vez que se termine la comparación se pueda hacer un análisis del mismo
- probabilities
 - En esta función imprimiremos todas las probabilidades de nuestro dataset de que sean una buena o mala película, esto usando los parametros obtenidos de nuestro modelo entrenado

```

150 #Compare the model with the test set AKA validation dataset
151 def probabilities(x_test, params):
152     # Normaliza los datos de prueba
153     normalized_x_test = normalization(x_test)
154
155     # Calcula las probabilidades para el conjunto de prueba
156     y_pred_prob = [sigmoid(linear(params, sample)) for sample in normalized_x_test]
157
158     # Imprime todas las probabilidades junto con su interpretación
159     for i, prob in enumerate(y_pred_prob):
160         print(f"Title {testing_data['Title'].iloc[i+1]}: Probability = {prob:.4f}")
161         if prob >= 0.5:
162             print("Expected a good movie with this probability: ", prob)
163         else:
164             print("Expect a bad movie with this probability: ", prob)
165         print('-----')
166
167     return y_pred_prob

```

Tomaremos el siguiente dataset de IMDB, el cual lo limpie previamente para limitar la cantidad de columnas y datos que recibe para predecir, manteniendo solamente, el título el cual no lo usaremos en ningún momento de la predicción, rating, votes y revenue en millions, el dato a predecir es la metascor de la misma.

Están divididos en validation, testing y training para que pueda analizarlo de forma eficaz usando pandas, con training midiendo 251 datos, validation 30 y 30 para test. Al momento de analizar con respecto a solamente training toma 3785 epochs en contrar un parametro adecuado al tener un error menor al 0.000001

```

-----
Epoch: 3784
Parameters: [ 1.37951938 -0.36575352  0.1141889 ]
Current error: 0.5600980114318054
Previous error: 0.5600990112209014
-----
Finished training

```

Posteriormente guardamos estos parametros para realizar una corrida con el dataset de validation el cual nos despliega los siguientes valores de matriz de confusión.

```

Epoch: 2220
Parameters: [1.81120621 0.07288508 0.07650213]
Current error: 0.4364558326402712
Previous error: 0.4364568320058274
-----
R^2 test and prediction: 0.5772727272727272
Confusion Matrix: {'True Positives': 17, 'True Negatives': 11, 'False Positives': 0, 'False Negatives': 3, 'Accuracy': 0.9032258064516129, 'Specifity': 1.0, 'Precision': 1.0, 'F1 Score': 0.9189189189189189, 'Recall': 0.85}
-----
Finished comparing

```

True Positives: 17,
True Negatives: 11,
False Positives: 0,

False Negatives: 3,
Accuracy: 0.9032258064516129,
Specifity: 1.0,
Precision: 1.0,
F1 Score: 0.9189189189189189
Recall: : 0.85

Por último para comparar las probabilidades que lanza el modelo usando el test, podemos ver las siguientes probabilidades para cada caso, con varias películas las cuales el modelo considerará malas o buenas.

```
Title Stardust:
Expected a good movie with this probability: 0.8037309536424588
-----
Title American Hustle:
Expected a good movie with this probability: 0.6068540956275106
-----
Title Jennifer's Body:
Expect a bad movie with this probability: 0.04769989847721063
-----
Title Midnight in Paris:
Expected a good movie with this probability: 0.7656455969949044
-----
```

En conclusión podemos decir que el modelo realiza su cometido ya que tiene probabilidades vistas para películas que no conozca haciendo un análisis, que si es una buena o mala película.

```
Sample 1: Probability = 0.2867
Expect a bad movie with this probability: 0.28668046935816976
-----
Sample 2: Probability = 0.6694
Expected a good movie with this probability: 0.6694057962504452
-----
Sample 3: Probability = 0.3996
Expect a bad movie with this probability: 0.3996397562533787
-----
Sample 4: Probability = 0.3938
Expect a bad movie with this probability: 0.39380684995698306
-----
Sample 5: Probability = 0.6663
Expected a good movie with this probability: 0.6662897755501807
-----
Sample 6: Probability = 0.3453
Expect a bad movie with this probability: 0.3453457568546494
-----
Sample 7: Probability = 0.6813
Expected a good movie with this probability: 0.6813459947091449
-----
Sample 8: Probability = 0.2152
Expect a bad movie with this probability: 0.21516767839700324
-----
Sample 9: Probability = 0.2247
Expect a bad movie with this probability: 0.22471293804442397
-----
Sample 10: Probability = 0.7926
Expected a good movie with this probability: 0.7926396873731616
-----
Sample 11: Probability = 0.6221
Expected a good movie with this probability: 0.6220710804509563
-----
Sample 12: Probability = 0.0886
Expect a bad movie with this probability: 0.08859725967713639
-----
```

```
Sample 13: Probability = 0.6172
Expected a good movie with this probability: 0.6171973879352592
-----
Sample 14: Probability = 0.6796
Expected a good movie with this probability: 0.679576435317383
-----
Sample 15: Probability = 0.8328
Expected a good movie with this probability: 0.8328496036933907
-----
Sample 16: Probability = 0.7473
Expected a good movie with this probability: 0.7472800729560392
-----
Sample 17: Probability = 0.8604
Expected a good movie with this probability: 0.8603736446466942
-----
Sample 18: Probability = 0.8282
Expected a good movie with this probability: 0.828195829069278
-----
Sample 19: Probability = 0.3952
Expect a bad movie with this probability: 0.3951972534731864
-----
Sample 20: Probability = 0.3972
Expect a bad movie with this probability: 0.3972297160582582
-----
Sample 21: Probability = 0.5806
Expected a good movie with this probability: 0.5805729453495858
-----
Sample 22: Probability = 0.3163
Expect a bad movie with this probability: 0.3163095651914222
-----
Sample 23: Probability = 0.5736
Expected a good movie with this probability: 0.5736440103839169
-----
Sample 24: Probability = 0.2010
Expect a bad movie with this probability: 0.20096736628277476
```



```
Sample 25: Probability = 0.1523
Expect a bad movie with this probability: 0.1523350839045311
-----
Sample 26: Probability = 0.8037
Expected a good movie with this probability: 0.8037309536424588
-----
Sample 27: Probability = 0.6069
Expected a good movie with this probability: 0.6068540956275106
-----
Sample 28: Probability = 0.0477
Expect a bad movie with this probability: 0.04769989847721063
-----
Sample 29: Probability = 0.7656
Expected a good movie with this probability: 0.7656455969949044
-----
```