

# Práctica 2

## Regresión logística

Mario Emilio Jiménez Vizcaíno  
A01173359@itesm.mx  
Tecnológico de Monterrey  
Ingeniería en Tecnologías Computacionales  
Monterrey, N.L., México

Jesus Abraham Haros Madrid  
A01252642@itesm.mx  
Tecnológico de Monterrey  
Ingeniería en Tecnologías Computacionales  
Monterrey, N.L., México

### ABSTRACT

## 1 INTRODUCCIÓN

Actualmente, los modelos de regresión se han convertido en uno de los componentes más útiles del análisis de datos y el aprendizaje automático, en donde sea necesario describir la relación entre una variable resultante y una o más variables independientes. En algunos casos, la variable de resultado es discreta y adopta dos o más valores, es por eso que el modelo de regresión logística se ha convertido en muchos campos como el método estándar de análisis en esta situación.[1]

En esta práctica demostraremos dos implementaciones de este modelo, la implementación de la librería de Python *sci-kit learn* y nuestra implementación usando el método de gradiente descendente.

## 2 CONCEPTOS PREVIOS

- Programación básica en los lenguajes R y Python
- Conocimiento de las librerías scikit-learn, pandas y numpy
- Conocimientos de estadística y de regresión logística

## 3 METODOLOGÍA

### 3.1 Datasets

Para comparar ambas implementaciones utilizamos dos datasets, expuestos a continuación:

#### 3.1.1 Dataset DEFAULT.

Este dataset está compuesto por 10,000 filas, cada una representa un cliente de un banco que puede o no cumplir con los pagos de su tarjeta de crédito (columna "default"). De cada cliente tenemos la siguiente información:

- Columna "default": Tiene los valores "Yes"/"No", representa si la persona realizó el pago mínimo a su tarjeta de crédito.
- Columna "student": Valores "Yes"/"No", representa si el cliente es un estudiante en ese momento.
- Columna "balance": Número decimal positivo que representa el balance de la tarjeta de crédito del cliente. Promedio de 835.4, números en el rango [0, 2654.3].
- Columna "income": Número decimal positivo que representa los ingresos que tiene el cliente. Promedio de 33517, números en el rango [772, 73554]

De este dataset, nuestro objetivo es predecir la columna "default" a partir de los otros tres parámetros, y como preparación cambiamos los valores de "student" ("Yes"/"No") a valores 1 y 0 respectivamente.

#### 3.1.2 Dataset GENERO.

Este dataset representa las mediciones de peso y altura de 10,000 personas, en conjunto con el género de la persona a la que se realizaron las medidas. Las columnas son:

- "Gender": Valores "Male"/"Female", el género de la persona a la que le corresponde esta fila de mediciones.
- "Height": La altura de la persona en pulgadas, promedio de 66.37, en el rango [54.26 y 79.00].
- "Weight": El peso de la persona en libras, promedio de 161.4, en el rango [64.7, 270.0].

La columna objetivo seleccionada de este dataset fue el género ya que tiene dos clasificaciones.

## 3.2 Regresión logística con *sklearn*

Para la primera parte de la práctica, en la que utilizamos la implementación de *sci-kit learn*, seleccionamos la clase *sklearn.linear\_model.LogisticRegression*[2] para nuestros scripts.

#### 3.2.1 Dataset DEFAULT.

Para este dataset primero leemos el archivo CSV a un *Dataframe* de *pandas*, transformamos las columnas "default" y "student" para que contengan valores booleanos y enteros respectivamente, seleccionamos las columnas que nos servirán como variables independientes (columnas "student", "balance" e "income") y variable dependiente (columna "default"). Después partimos las filas del dataset en una porción del 80% que usaremos para entrenar el modelo, y otra porción del 20% para probarlo.

Instanciamos el modelo de regresión de *sklearn*, lo entrenamos con los datos y después predecimos la variable dependiente con el modelo para así compararlo con los datos reales de prueba, usando una medida de tasa de precisión y la matriz de confusión.

El código de este ejemplo puede encontrarse en el apéndice A.

#### 3.2.2 Dataset GENERO.

Para este dataset realizamos un procedimiento similar: leer el dataset para crear un *Dataframe*, seleccionar las columnas de variables independientes ("Height" y "Weight") y la dependiente ("Gender"), dividir el dataset en 80%/20%, entrenar el modelo, y predecir la variable dependiente para los datos de prueba, para así comparar estos con los datos reales.

El código para esta sección se encuentra en el apéndice B.

## 3.3 Regresión logística con Gradiente Descendente

En esta sección de la práctica implementamos nuestra propia clase de gradiente descendente, e intentamos que los métodos de nuestro

modelo fueran muy parecidos a los de la clase de *sklearn* para que así fuera fácil intercambiar las implementaciones. A pesar de esto, implementamos algunas diferencias para que el código funcionara correctamente.

Nuestra implementación del modelo de regresión logística se encuentra en el apéndice C.

### 3.3.1 Dataset DEFAULT.

**TODO: Hablar de diferencias con el código anterior**

El código fuente de este ejemplo se encuentra en el apéndice D.

### 3.3.2 Dataset GENERO.

**TODO: Hablar de diferencias con el código anterior**

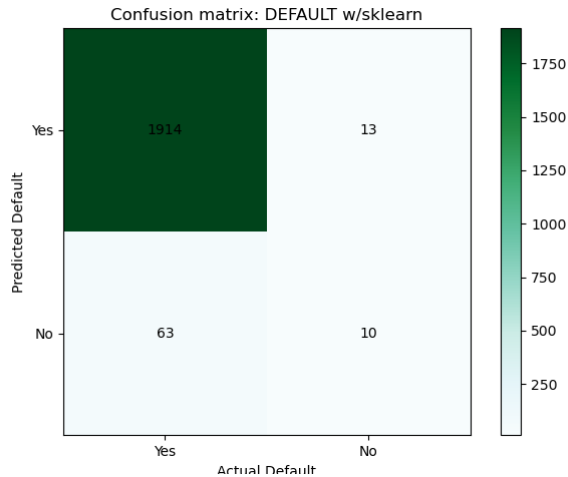
El código fuente puede ser encontrado en el apéndice E.

## 4 RESULTADOS

### 4.1 Regresión logística con *sklearn*

#### 4.1.1 Dataset DEFAULT.

El modelo de regresión logística de *sklearn* tuvo una tasa de precisión alrededor de 0.965, con una gráfica de la matriz de confusión así:

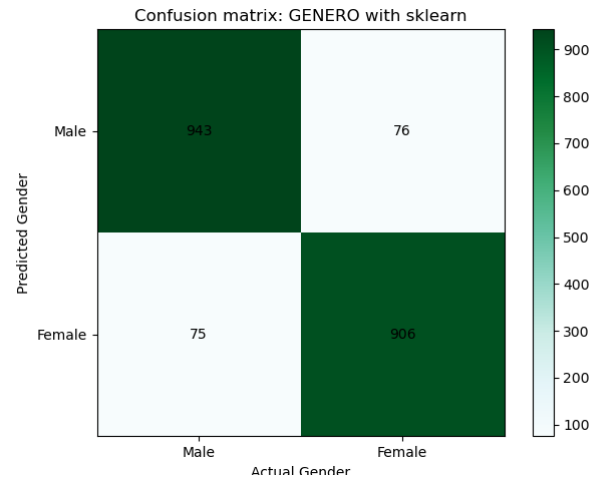


**Figure 1: Matriz de confusión del dataset DEFAULT con la implementación de sklearn**

Se puede observar que la gran mayoría de los resultados son positivos, por lo que el modelo usualmente acierta al clasificar la casi la totalidad de las filas como positivas.

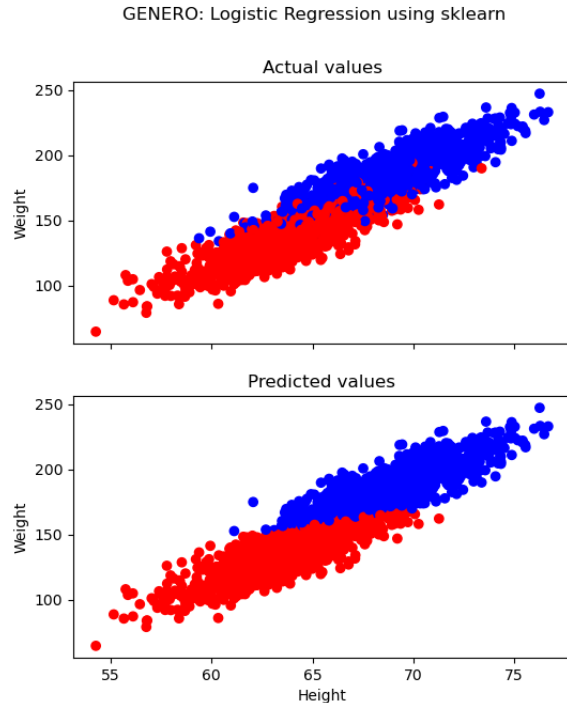
#### 4.1.2 Dataset GENERO.

En este ejemplo, el modelo de *sklearn* tuvo una tasa de precisión cercana a 0.925, y una matriz de confusión de esta forma:



**Figure 2: Matriz de confusión del dataset GENERO con la implementación de sklearn**

También se creó un diagrama de dispersión utilizando como eje X la altura de la persona, y en el eje Y el peso. Los puntos rojos representan personas del género femenino y los azules personas del género masculino.



**Figure 3: Diagrama de dispersión del dataset GENERO con la implementación de sklearn**

La gráfica superior representa el conjunto de valores de prueba, mientras que la gráfica inferior muestra el género predecido por el modelo de regresión a partir de los valores de prueba.

Para la generación de estas gráficas se utilizó el código incluido en el apéndice F.

## 4.2 Regresión logística con Gradiente Descendente

### 4.2.1 Dataset *DEFAULT*.

**TODO: Incluir gráficas**

### 4.2.2 Dataset *GENERO*.

**TODO: Incluir gráficas**

## 5 CONCLUSIONES Y REFLEXIONES

**TODO: Echarle choro de que la regresión logística sirve para clasificar, y que en el dataset *DEFAULT* no jaló muy bien porque había demasiados "Yes" en la columna de "default", y entonces el modelo frecuentemente los clasificaba "Yes"**

### 5.1 Reflexión de Abraham

**TODO**

### 5.2 Reflexión de Mario

Considero que esta práctica resolvió muchas de las dudas que tenía sobre la regresión logística y me ayudó a comprender en qué situaciones son útiles este tipo de algoritmos de clasificación, además de que, con el primer dataset, pude observar un ejemplo de un caso en el que probablemente la regresión logística no es la mejor solución a este problema ya que el dataset está demasiado sesgado hacia los clientes que no pagaron su tarjeta contra los que si.

Además, mejoré mis habilidades en Python utilizando librerías como *sklearn*, *numpy* y *pandas*, herramientas usadas frecuentemente en este tipo de análisis de datos por su facilidad de manipular grandes cantidades de información y abstraer modelos complejos, para ser expresados en pocas líneas de código.

## REFERENCES

- [1] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. 2013. *Applied logistic regression*. Vol. 398. John Wiley & Sons.
- [2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

## A CÓDIGO DE REGRESIÓN LOGÍSTICA CON SKLEARN DEL DATASET DEFAULT

```

1 import pandas as pd
2 from sklearn.linear_model import LogisticRegression
3 from sklearn.model_selection import train_test_split
4 from sklearn.metrics import accuracy_score, confusion_matrix
5
6 from graphs import graphConfusionMatrix
7
8 print(" ~ Reading default.txt and generating train and test sets")
9 data = pd.read_csv('default.txt', sep=" ")
10 # Transform 'default' column from Yes/No to a boolean
11 data["default"] = (data["default"] == "Yes").astype(bool)
12 # Transform 'student' column from Yes/No to an integer
13 data["student"] = (data["student"] == "Yes").astype(int)
14 x = data.iloc[:, 1:4].values.reshape(-1, 3)
15 y = data.iloc[:, 0].values.reshape(-1, 1)
16 xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size=0.2)
17
18 print(" ~ Creating sklearn's logistic regression model")
19 regressor = LogisticRegression()
20 regressor.fit(xTrain, yTrain.ravel())
21
22 print(" ~ Testing sklearn's logistic regression model")
23 yPredicted = regressor.predict(xTest)
24 accuracy = accuracy_score(yTest, yPredicted)
25 print(" → Accuracy:", accuracy)
26 cm = confusion_matrix(yTest, yPredicted)
27 print(" → Confusion matrix:", cm.tolist())
28
29 graphConfusionMatrix(cm, ["Yes", "No"], "Default",
30                        "DEFAULT w/sklearn", "default_cm_sklearn.png")

```

## B CÓDIGO DE REGRESIÓN LOGÍSTICA CON SKLEARN DEL DATASET GENERO

```
1 import pandas as pd
2 from sklearn.linear_model import LogisticRegression
3 from sklearn.model_selection import train_test_split
4 from sklearn.metrics import accuracy_score, confusion_matrix
5
6 from graphs import graphConfusionMatrix, graphGENERO
7
8 print(" ~ Reading genero.txt and generating train and test sets")
9 data = pd.read_csv('genero.txt')
10 x = data.iloc[:, 1:3].values.reshape(-1, 2)
11 y = data.iloc[:, 0].values.reshape(-1, 1)
12 xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size=0.2)
13
14 print(" ~ Creating sklearn's logistic regression model")
15 regressor = LogisticRegression()
16 regressor.fit(xTrain, yTrain.ravel())
17
18 print(" ~ Testing sklearn's logistic regression model")
19 yPredicted = regressor.predict(xTest)
20 accuracy = accuracy_score(yTest, yPredicted)
21 print(" → Accuracy:", accuracy)
22 cm = confusion_matrix(yTest, yPredicted)
23 print(" → Confusion matrix:", cm.tolist())
24
25 graphGENERO(xTest, yTest, yPredicted, "Logistic Regression using sklearn",
26             "genero_sklearn.png")
27 graphConfusionMatrix(cm, ["Male", "Female"], "Gender",
28                        "GENERO with sklearn", "genero_cm_sklearn.png")
```

## C NUESTRA IMPLEMENTACIÓN DE GRADIENTE DESCENDENTE

```

1 from typing import Tuple
2 import numpy as np
3 import pandas as pd
4 from sklearn.metrics import mean_squared_error
5
6
7 def normalize(x: pd.DataFrame) -> Tuple[np.ndarray, float, float]:
8     mu = np.mean(x, axis=0)
9     sigma = np.std(x, axis=0, ddof=1)
10    x_norm = (x - mu) / sigma
11    return x_norm, mu, sigma
12
13
14 class GradientDescent:
15     def __init__(self, learning_rate: float = 0.1, max_iterations: int = 200,
16                  precision: float = 0.00001):
17         self.learning_rate = learning_rate
18         self.max_iterations = max_iterations
19         self.precision = precision
20         self.theta = None
21         self.mu = None
22         self.sigma = None
23
24     def fit(self, x: pd.DataFrame, y: pd.DataFrame):
25         x, self.mu, self.sigma = normalize(x)
26         x = np.hstack((x, np.ones((x.shape[0], 1))))
27         self.theta = np.zeros(x.shape[1])
28         prev_cost = -1
29         for _ in range(self.max_iterations):
30             predictions = x.dot(self.theta)
31
32             # TODO
33
34             cost = mean_squared_error(y, predictions)
35             if abs(cost - prev_cost) < self.precision:
36                 break
37             prev_cost = cost
38
39     def predict(self, x: pd.DataFrame):
40         if self.theta is None or self.mu is None or self.sigma is None:
41             raise Exception(
42                 "GradientDescent::predict() called before model was trained")
43
44         x = (x - self.mu) / self.sigma
45         x = np.hstack((x, np.ones((x.shape[0], 1))))
46         x.dot(self.theta)
47
48         # TODO

```

## D CÓDIGO DE REGRESIÓN LOGÍSTICA CON GRADIENTE DESCENDENTE DEL DATASET DEFAULT

```
1 # TODO
```

## E CÓDIGO DE REGRESIÓN LOGÍSTICA CON GRADIENTE DESCENDENTE DEL DATASET GENERO

```
1 import pandas as pd
2 from sklearn.model_selection import train_test_split
3 from sklearn.metrics import accuracy_score, confusion_matrix
4
5 from GradientDescent import GradientDescent
6 from graphs import graphConfusionMatrix, graphGENERO
7
8 print(" ~ Reading genero.txt and generating train and test sets")
9 data = pd.read_csv('genero.txt')
10 x = data.iloc[:, 1:3].values.reshape(-1, 2)
11 y = data.iloc[:, 0].values.reshape(-1, 1)
12 xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size=0.2)
13
14 print(" ~ Creating our logistic regression model with gradient descent")
15 regressor = GradientDescent()
16 regressor.fit(xTrain, yTrain.ravel())
17
18 print(" ~ Testing our logistic regression model with gradient descent")
19 yPredicted = regressor.predict(xTest)
20 accuracy = accuracy_score(yTest, yPredicted)
21 print(" → Accuracy:", accuracy)
22 cm = confusion_matrix(yTest, yPredicted)
23 print(" → Confusion matrix:", cm.tolist())
24
25 graphGENERO(xTest, yTest, yPredicted, "Logistic Regression using gradient descent",
26             "genero_sklearn.png")
27 graphConfusionMatrix(cm, ["Yes", "No"], "Gender",
28                        "GENERO with gradient descent", "genero_cm_gradient.png")
```

## F CÓDIGO DE GENERACIÓN DE GRÁFICAS

```

1 import sys
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 shouldDisplay = "--display-graphs" in sys.argv
6 shouldSave = "--save-graphs" in sys.argv
7
8 mapColor = np.vectorize(lambda x: "b" if x == "Male" else "r")
9
10
11 def graphGENERO(xys, actualVals, predictedVals, title, filename):
12     fig, (ax1, ax2) = plt.subplots(2, sharex=True)
13     fig.set_size_inches(6, 7)
14     fig.suptitle("GENERO: " + title)
15
16     ax1.set_title("Actual values")
17     ax1.set_ylabel("Weight")
18     actualColors = mapColor(actualVals).flatten()
19     ax1.scatter(xys[:, 0], xys[:, 1], c=actualColors)
20
21     ax2.set_title("Predicted values")
22     ax2.set_xlabel("Height")
23     ax2.set_ylabel("Weight")
24     predictedColors = mapColor(predictedVals).flatten()
25     ax2.scatter(xys[:, 0], xys[:, 1], c=predictedColors)
26
27     if shouldSave:
28         fig.savefig(filename)
29     if shouldDisplay:
30         plt.show()
31
32
33 def graphConfusionMatrix(cm, labels, variableName, title, filename):
34     plt.figure()
35     plt.imshow(cm, interpolation="nearest", cmap="BuGn")
36     plt.title("Confusion matrix: " + title)
37     plt.colorbar()
38     plt.xticks(np.arange(2), labels, size=10)
39     plt.yticks(np.arange(2), labels, size=10)
40     plt.tight_layout()
41     plt.xlabel("Actual " + variableName)
42     plt.ylabel("Predicted " + variableName)
43
44     for x in range(2):
45         for y in range(2):
46             plt.annotate(cm[x][y], xy=(y, x),
47                         horizontalalignment="center",
48                         verticalalignment="center")
49
50     if shouldSave:
51         plt.savefig(filename)
52     if shouldDisplay:
53         plt.show()

```