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

2 CONCEPTOS PREVIOS

- Concepto

3 METODOLOGÍA

3.1 Regresión logística con *sklearn*

3.1.1 Dataset DEFAULT.

3.1.2 Dataset GENERO.

3.2 Regresión logística con Gradiente Descendente

3.2.1 Dataset DEFAULT.

3.2.2 Dataset GENERO.

4 RESULTADOS

4.1 Regresión logística con *sklearn*

4.1.1 Dataset DEFAULT.

4.1.2 Dataset GENERO.

4.2 Regresión logística con Gradiente Descendente

4.2.1 Dataset DEFAULT.

4.2.2 Dataset GENERO.

5 CONCLUSIONES Y REFLEXIONES

5.1 Refrexión de Abraham

5.2 Reflexión de Mario

REFERENCES

A CÓDIGO DE REGRESIÓN LOGÍSTICA CON SKLEARN DEL DATASET DEFAULT

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

from graphs import graphDEFAULT

print(” ~ Reading default.txt and generating train and test sets”)
data = pd.read_csv('default.txt', sep=” ”)
# Transform 'default' column from Yes/No to a boolean
data[”default”] = (data[”default”] == ”Yes”).astype(bool)
# Transform 'student' column from Yes/No to an integer
data[”student”] = (data[”student”] == ”Yes”).astype(int)
x = data.iloc[:, 1:4].values.reshape(-1, 3)
y = data.iloc[:, 0].values.reshape(-1, 1)
xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size=0.2)

print(” ~ Creating sklearn's logistic regression model”)
regressor = LogisticRegression()
regressor.fit(xTrain, yTrain.ravel())

print(” ~ Testing sklearn's logistic regression model”)
yPredicted = regressor.predict(xTest)
accuracy = accuracy_score(yTest, yPredicted)
print(” →   Accuracy:”, accuracy)

# TODO
graphDEFAULT(xTest, yTest, yPredicted, ”Logistic Regression using sklearn”,
             ”default_sklearn.png”)
```

B CÓDIGO DE REGRESIÓN LOGÍSTICA CON SKLEARN DEL DATASET GENERO

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

from graphs import graphGENERO

print(” ~ Reading genero.txt and generating train and test sets”)
data = pd.read_csv('genero.txt')
x = data.iloc[:, 1:3].values.reshape(-1, 2)
y = data.iloc[:, 0].values.reshape(-1, 1)
xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size=0.2)

print(” ~ Creating sklearn's logistic regression model”)
regressor = LogisticRegression()
regressor.fit(xTrain, yTrain.ravel())

print(” ~ Testing sklearn's logistic regression model”)
yPredicted = regressor.predict(xTest)
accuracy = accuracy_score(yTest, yPredicted)
print(” →   Accuracy:”, accuracy)

# TODO
graphGENERO(xTest, yTest, yPredicted, ”Logistic Regression using sklearn”,
            ”genero_sklearn.png”)
```

C NUESTRA IMPLEMENTACIÓN DE GRADIENTE DESCENDENTE

```python
from typing import Tuple
import numpy as np
import pandas as pd
from sklearn.metrics import mean_squared_error


def normalize(x: pd.DataFrame) -> Tuple[np.ndarray, float, float]:
    mu = np.mean(x, axis=0)
    sigma = np.std(x, axis=0, ddof=1)
    x_norm = (x - mu) / sigma
    return x_norm, mu, sigma


class GradientDescent:
    def __init__(self, learning_rate: float = 0.1, max_iterations: int = 200,
                 precision: float = 0.00001):
        self.learning_rate = learning_rate
        self.max_iterations = max_iterations
        self.precision = precision
        self.theta = None
        self.mu = None
        self.sigma = None

    def fit(self, x: pd.DataFrame, y: pd.DataFrame):
        x, self.mu, self.sigma = normalize(x)
        x = np.hstack((x, np.ones((x.shape[0], 1))))
        self.theta = np.zeros(x.shape[1])
        prev_cost = -1
        for _ in range(self.max_iterations):
            predictions = x.dot(self.theta)

            # TODO

            cost = mean_squared_error(y, predictions)
            if abs(cost - prev_cost) < self.precision:
                break
            prev_cost = cost

    def predict(self, x: pd.DataFrame):
        if self.theta is None or self.mu is None or self.sigma is None:
            raise Exception(
                ”GradientDescent::predict() called before model was trained”)

        x = (x - self.mu) / self.sigma
        x = np.hstack((x, np.ones((x.shape[0], 1))))
        x.dot(self.theta)

        # TODO
```

D CÓDIGO DE REGRESIÓN LOGÍSTICA CON GRADIENTE DESCENDENTE DEL DATASET DEFAULT

```
# TODO
```

E CÓDIGO DE REGRESIÓN LOGÍSTICA CON GRADIENTE DESCENDENTE DEL DATASET GENERO

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

from GradientDescent import GradientDescent
from graphs import graphGENERO

print(” ~ Reading genero.txt and generating train and test sets”)
data = pd.read_csv('genero.txt')
x = data.iloc[:, 1:2].values.reshape(-1, 1)
y = data.iloc[:, 0].values.reshape(-1, 1)
xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size=0.2)

print(” ~ Creating our logistric regression model with gradient descent”)
regressor = GradientDescent()
regressor.fit(xTrain, yTrain.ravel())

print(” ~ Testing our logistric regression model with gradient descent”)
yPredicted = regressor.predict(xTest)
accuracy = accuracy_score(yTest, yPredicted)
print(” →  Accuracy:”, accuracy)

graphGENERO(xTest, yTest, yPredicted, ”Logistic Regression using Gradient Descent”,
            ”genero_gradient.png”)
```

F CÓDIGO DE GENERACIÓN DE GRÁFICAS

```python
import sys
import matplotlib.pyplot as plt
import numpy as np

shouldDisplay = ”--display-graphs” in sys.argv
shouldSave = ”--save-graphs” in sys.argv


def graphDEFAULT(height, weight, theta, title, filename):
    # TODO
    pass


def graphGENERO(height, weight, predictions, title, filename):
    fig, ax = plt.subplots()
    ax.set_title(”GENERO: ” + title)
    ax.set_xlabel(”Height”)
    ax.set_ylabel(”Weight”)

    # TODO

    if shouldSave:
        fig.savefig(filename)
    if shouldDisplay:
        plt.show()
```