

Tarea 4 - Análisis de Datos II

Universidad de Costa Rica

Material del curso CA0305 I-2024

Tarea 4: Uso de SQL - Python.

Encargado: Potoy Juárez Luis Alberto

Correo: luis.juarezpotoy@ucr.ac.cr

En la resolución de la tarea considere:

Indicaciones


1. La tarea debe llevar el nombre: *Tarea_4 - Carnet - NombreApellidos*.
2. La tarea es de forma individual. En caso de utilizar alguna fuente o referencia documento. Todo acto de plagio será motivo invalidación de la tarea
3. La entrega de la tarea es por medio de mediación virtual y con fecha límite del día **4 Junio 2024 11:59 pm**. En caso no entregar la tarea en la hora y fecha señalada se calificará con base a 70%. Si la entrega es después de 3 días se asignará un 0%.
4. El desarrollo de la tarea debe realizarlo Jupyter Notebook. Debe adjuntar solo un documento html.
5. Al no cumplir algunas de las normas estándares de programación vista en el curso y items anteriores será motivo de reducción de 0 pts a 5 pts, por ejercicio.

SQL - Python

 (50 pts) Usando la base de datos [Seguro de autos](#) realice lo siguiente:

- Cargue la base en un objeto DataFrame Python. Asegúrese que la base tenga un *Id* único por observación, si no lo tiene cree una columna con ello.
- Realice conexión Lite y cargue la base en SQL.
- Cree datos nulos en la columna *PREMIUM* (costo de la prima) a todas las filas que contengan un *9* en su numeración (Id). Sugerencia use *CASE* y *LIKE* para modificar las filas.
- Discretice la variable *INSURED_VALUE* (nueva variable *rango_asegurado*) con los siguientes cortes 0, 1e3, 1e6, 1e9, 1e12.
- Utilice 2 métodos de imputación de datos nulos por medio de agrupación (con base a Grupos 1: *SEX*, *PROD_YEAR*, *SEATS_NUM*, *rango_asegurado*, Grupo 2: Puede elegir una agrupación que considere óptima). Estos grupos son fijos.
- Convierta la base resultante a python DataFrame y cuantifique cuál de los métodos y agrupación genera menor error con base a las métricas *RSS*, *RMSE* y *R²*. Comente los resultados.

Optimización de recursos

 (50 pts) Seguidamente se le facilitará la programación de uno de los modelos más común en el análisis de datos, *Regresión logística* (ver anexos). Su labor consiste en:

- Entender, documentar y comentar el código.
- Realice 10 corridas del código y guarde en una tabla resumen los resultados del tiempo ejecución.
- Haga modificaciones al código con el fin de acelerar el tiempo de ejecución (las modificaciones pueden ser las que guste). Guarde los resultados de 10 corridas de la mejor versión que usted realizó.
- Utilice el módulo `sklearn.linear_model` e importe `LogisticRegression`. Pruebe los mismo datos y guarde los resultados de tiempo de ejecución.
- Haga una tabla resumen final donde unifique las tablas de los 3 items anteriores. Saque sus conclusiones y comente las modificaciones que dieron paso a reducir el tiempo de ejecución.

Nota:

- En caso de no reducir el tiempo de ejecución con sus modificaciones perderá 5 puntos del ejercicio 2.
- Las pruebas deben realizarse con la base de datos facilitada en anexos "Ejemplo de uso".

Anexos

Regresión logística

```
def kernel_sigmoid(z):
    return 1 / (1 + np.exp(-z))

def forward_propagation(W, b, X, y):
    data_number = X.shape[0]
    Z = np.dot(W, X.T) + b
    A = kernel_sigmoid(Z)
    cost = (- 1 / data_number) * np.sum(y * np.log(A) + (1 - y) * (np.log(1 - A)))

    return A, cost

def backward_propagation(X, A, y):
    data_number = X.shape[0]
    dW = (1 / data_number) * np.dot((A - y), X)
    db = (1 / data_number) * np.sum(A - y)

    return dW, db

def optimize(W, b, X, y, num_iterations, learning_rate):
    costs = []

    for i in range(num_iterations):
        A, cost = forward_propagation(W, b, X, y)
        dW, db = backward_propagation(X, A, y)

        W = W - learning_rate * dW
        b = b - learning_rate * db

        if i % 100 == 0:
            costs.append(cost)

    params = {
        "W": W,
```

```

        "b": b
    }

    gradients = {
        "dW": dW,
        "db": db
    }
    return params, gradients, costs

def predict(W, b, X):
    data_number = X.shape[0]
    y_prediction = np.zeros((1, data_number))

    Z = np.dot(W, X.T) + b
    A = kernel_sigmoid(Z)

    for i in range(A.shape[1]):
        y_prediction[0, i] = 1 if A[0, i] > 0.5 else 0

    return y_prediction

def model_regresion_logistico(
    X_train,
    y_train,
    X_val,
    Y_val,
    num_iterations=2000,
    learning_rate=0.5):

    dimensions = X.shape[1]

    W = np.zeros(shape=(1, dimensions))
    b = 0

    params, gradients, costs = optimize(W, b, X_train, y_train, num_iterations,
    learning_rate)

    W = params["W"]
    b = params["b"]

    y_prediction_validation = predict(W, b, X_val)

    y_prediction_train = predict(W, b, X_train)

    lista = {
        "Ajuste entrenamiento": 100 - np.mean(np.abs(y_prediction_train - y_train)) * 100,
        "Ajuste testeo": 100 - np.mean(np.abs(y_prediction_validation - y_val)) * 100,
        "Costo en proceso": costs
    }

    return lista

```

Ejemplo de uso

```

# Librerias
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd

# Datos y transformaciones
datos = pd.read_csv("Diabetes.txt")
X = np.array(datos.drop(["Outcome"], axis=1))
y = np.array(datos["Outcome"])
#
X = (X - np.min(X)) / (np.max(X) - np.min(X))

# Separa la muestra
X_train, X_val, y_train, y_val = train_test_split(
    X, # Covariables predictoras
    y, # Variable a predecir
    random_state = None, #(None: escoge una diferente en cada corrida)
    test_size = 0.20 # Cantidad de datos de entrenamiento y prueba
)

# inicio Toma de tiempos -----

model_regresion_logistico(
    X_train,
    y_train, X_val,
    y_val,
    num_iterations = 1000,
    learning_rate = 0.003
)

# Fin Toma de tiempo -----

```