T3 CASOS DE ESTUDIO

December 6, 2025

Pregunta 1

En la administración de servidores web, el número de "Hits" o peticiones que recibe un servidor por minuto es una variable de conteo (números enteros no negativos), en este sentido, el tráfico web se modela mejor con una **distribución de Poisson**.

Deseamos predecir la tasa media de peticiones (λ) en función del número de usuarios activos simultáneos.

Utilizaremos un **modelo lineal generalizado (GLM)**. La probabilidad de observar y peticiones dado un vector de características x es:

$$\mathsf{P}(Y = y|x; \beta) = \frac{e^{-\lambda}\lambda^y}{y!}$$

Donde la tasa media λ se relaciona con la entrada mediante la función de enlace logarítmica (para garantizar $\lambda > 0$):

$$\lambda = e^{\beta_0+\beta_1 x}$$

El **objetivo de optimización** es maximizar la verosimilitud, o equivalentemente, minimizar la **log-verosimilitud negativa**. La función objetivo es:

$$J(\beta) = \sum_{i=1}^{m} \left(\lambda^{(i)} - y^{(i)} \log(\lambda^{(i)})\right) = \sum_{i=1}^{m} \left(e^{\beta_0+\beta_1 x^{(i)}} - y^{(i)}(\beta_0 + \beta_1 x^{(i)})\right)$$

Actividades:

1. Derivar matemáticamente el gradiente $\nabla J(\beta)$.
2. Implementar el método de **descenso de gradiente** para encontrar β.
3. Predecir la tasa de peticiones para una carga de usuarios inédita.

[1]:
```
# Datos simulados pregunta 1
# --------------------------
import numpy as np
import pandas as pd

np.random.seed(2025)
```

```python
m = 100

# Variable independiente (usuarios)
usuarios = np.random.uniform(10, 100, m)
# Variable dependiente (peticiones)
lambda_real = np.exp(0.5 + 0.03 * usuarios)
peticiones = np.random.poisson(lambda_real)

datos_pregunta_1 = pd.DataFrame({"Usuarios": usuarios,
                                 "Peticiones": peticiones})
print(datos_pregunta_1.head())
```

```
    Usuarios  Peticiones
0  22.193935           3
1  89.906653          36
2  93.934508          21
3  50.101135           9
4  44.941199           5
```

Pregunta 2

En Internet de las Cosas (IoT), a menudo necesitamos calibrar sensores baratos. Supongamos un sensor de distancia basado en la intensidad de señal WiFi (RSSI). La relación teórica sigue una ley de potencia inversa con un término de ruido base.

El modelo físico propuesto es:

$$h(x;\theta) = \frac{\theta_0}{x^{\theta_1}} + \theta_2$$

Donde x es la distancia real y $h(x)$ es la lectura del sensor.

El **objetivo de optimización** es minimizar el error cuadrático medio:

$$f(\theta) = \sum_{i=1}^{n} \left(y_i - \left(\frac{\theta_0}{x_i^{\theta_1}} + \theta_2 \right) \right)^2$$

Utilice el método **BFGS** provisto por `scipy` para manejar la curvatura sin calcular la Hessiana manualmente.

Actividades:

1. Definir la función objetivo.
2. Utilizar `scipy.optimize.minimize` con el método BFGS.
3. Comparar el resultado con diferentes inicializaciones aleatorias para observar la sensibilidad a los puntos de partida.

[2]:
```python
# Datos simulados pregunta 2
# --------------------------
import numpy as np
import pandas as pd

np.random.seed(2025)

# Variable independiente (metros)
distancia = np.linspace(1, 20, 50)
# Variable dependiente (lectura sensor)
lectura_sensor = (50 / (distancia ** 1.5)) +  2 + np.random.normal(0, 0.5, 50)

datos_pregunta_2 = pd.DataFrame({"Distancia": distancia,
                                 "Lectura_Sensor": lectura_sensor})
print(datos_pregunta_2.head())
```

```
   Distancia  Lectura_Sensor
0   1.000000       51.953805
1   1.387755       32.951598
2   1.775510       22.414731
3   2.163265       17.382961
4   2.551020       14.221177
```

Pregunta 3

Imagine un administrador de contenedores (como Kubernetes). Tiene una lista de N microservicios, cada uno con una carga computacional estimada (CPU load). Debe asignar cada servicio a uno de los K servidores disponibles. El objetivo es **minimizar el desequilibrio** (varianza) de la carga entre los servidores.

Sea $S = \{s_1, s_2, \dots, s_N\}$ el conjunto de cargas de los servicios. Sea A un vector de asignación donde $A_i \in \{0, 1, \dots, K-1\}$ indica el servidor del servicio i. La carga total del servidor k es:

$$L_k = \sum_{i:A_i=k} s_i$$

La función objetivo minimiza la desviación estándar de las cargas de los servidores:

$$f(A) = \sqrt{\frac{1}{K}\sum_{k=0}^{K-1}(L_k - \bar{L})^2}$$

Actividades:

1. Implemetar la función objetivo basada en la desviación estándar.
2. Adaptar el algoritmo **simulated annealing**:
    - **Estado:** Vector de enteros (asignación)

- **Vecino:** Mover un servicio aleatorio de un servidor a otro.

3. Analizar la reducción del desequilibrio.

[3]:
```python
# Datos simulados pregunta 3
# --------------------------
import numpy as np

np.random.seed(2025)

n_servicios = 50
n_servidores = 4
cargas_cpu = np.random.randint(1, 20, n_servicios)
```

Juan F. Olivares Pacheco (jfolivar@uda.l)

Universidad de Atacama, Facultad de Ingeniería, Departamento de Matemática