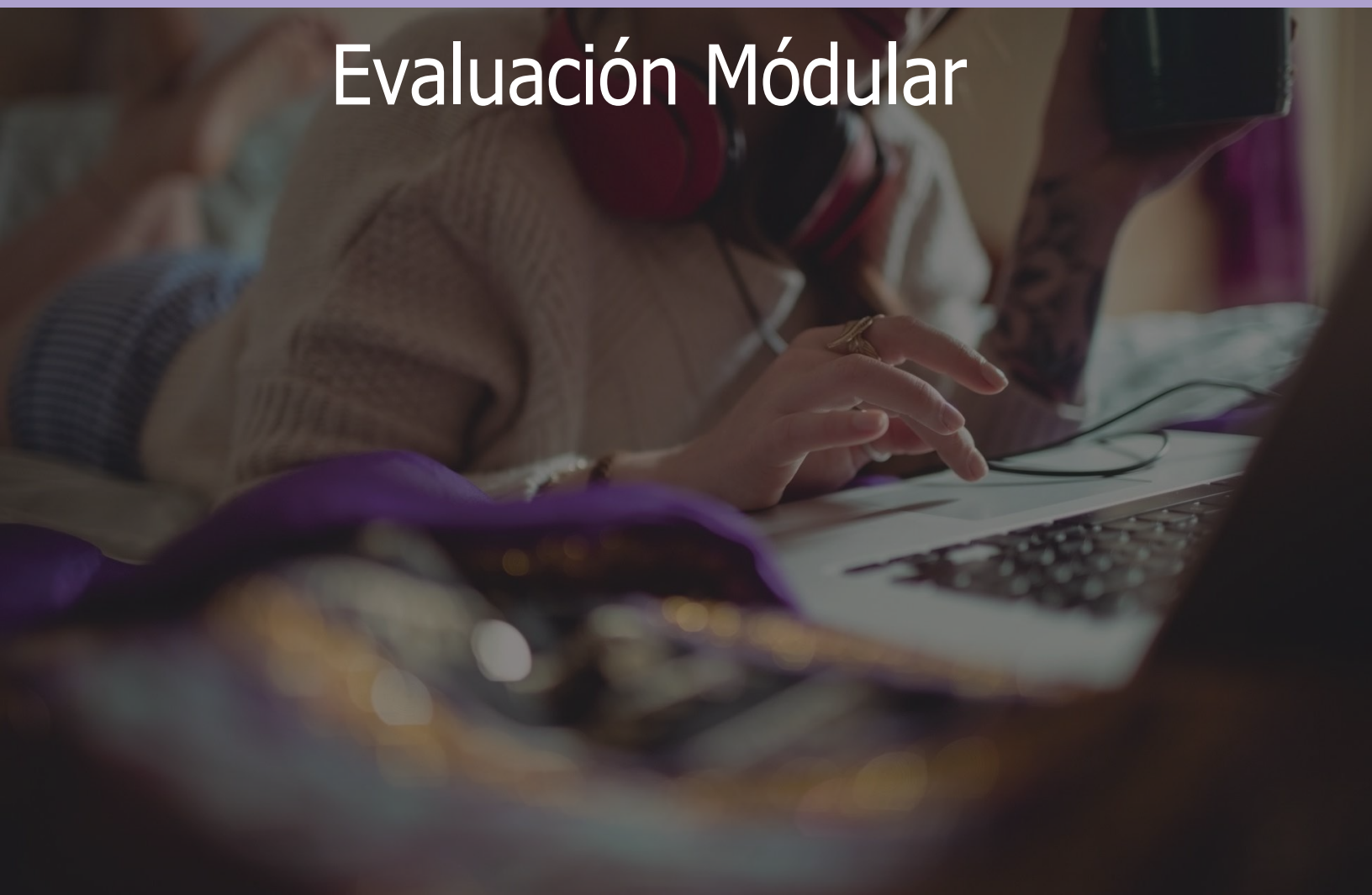
IT Academy
by KIBERNUM

Módulo 3

Evaluación Módular



ACTIVIDAD:

Optimización de un Modelo de Regresión Lineal Mediante Técnicas Matemáticas y Algorítmicas

Objetivo:

El estudiante debe demostrar, de forma individual, su capacidad para integrar y aplicar los conocimientos teóricos y prácticos adquiridos en el curso. Se espera que el alumno implemente un modelo de regresión lineal utilizando conceptos de álgebra matricial, resolución de sistemas de ecuaciones, cálculo diferencial (en una y múltiples variables) y métodos de optimización. Asimismo, deberá justificar el proceso de optimización mediante la derivación de la función de costo, el cálculo de gradientes y la aplicación de técnicas de optimización (por ejemplo, descenso de gradiente), validando y comparando los resultados obtenidos.

Requerimientos:

1. **Planteamiento del Proyecto:**
   Desarrollar un sistema en Python para entrenar un modelo de regresión lineal simple utilizando un conjunto de datos sintéticos. El proyecto deberá abordar lo siguiente:
   - Representación de Datos:
     - Generar un conjunto de datos que contenga una variable independiente x (por ejemplo, 100 datos generados aleatoriamente en un rango definido) y una variable dependiente y, siguiendo un modelo lineal con cierto nivel de ruido: $\boldsymbol{y = 4 + 3x + \epsilon}$
     - Representar el conjunto de datos mediante matrices y vectores utilizando NumPy.
   - Resolución del Modelo (Cálculo Cerrado):
     - Utilizar la fórmula normal de regresión lineal ($\beta = (X^T X)^{-1} X^T y$ para calcular los parámetros óptimos del modelo.
     - Implementar validaciones en caso de que la matriz $X^T X$ sea singular, utilizando manejo de excepciones.
   - Optimización Iterativa (Método de Descenso de Gradiente):
     - Definir la función de costo (por ejemplo, el error cuadrático medio) y sus derivadas parciales mediante cálculo diferencial.
     - Implementar el algoritmo de descenso de gradiente para minimizar la función de costo, actualizando iterativamente los parámetros (pendiente y sesgo).
     - Incluir una comparación entre el método del cálculo cerrado y el enfoque iterativo, presentando el valor final de la función de costo en cada caso.
   - Visualización y Análisis:
     - Graficar el conjunto de datos junto con la recta de regresión obtenida mediante ambos métodos, utilizando Matplotlib.
     - Generar un gráfico que muestre la evolución de la función de costo a lo largo de las iteraciones del descenso de gradiente.
     - Incluir un breve análisis en el que se explique la notación Big O de ambos algoritmos y se justifique la elección del método de optimización en función del tamaño y la naturaleza de los datos.
   - Documentación y Manejo de Excepciones:
     - Manejar de manera robusta las excepciones que puedan surgir durante la ejecución (por ejemplo, archivos no encontrados, errores de conversión de datos, problemas de singularidad) utilizando bloques try-except, else y finally.
     - Crear al menos una excepción personalizada para validar condiciones específicas, por ejemplo, que los datos de entrada tengan el formato correcto.

Requerimientos:

2. **Informe y Evidencias:**
    - Elaborar un documento (README o informe) que describa:
        - La estructura del proyecto, la metodología empleada y las decisiones de diseño.
        - Los resultados obtenidos con la solución analítica y el método iterativo, incluyendo tablas o gráficos comparativos.
        - Un análisis crítico sobre la optimización de la función de costo, la convergencia del algoritmo y la eficiencia computacional.
    - Incluir capturas de pantalla del código en el editor y de la salida (gráficos y mensajes en consola) que evidencien la ejecución del proyecto.
3. **Tiempo Estimado para la Realización de la Evaluación**
    a. Se estima que la realización del proyecto y la documentación requerida tomará 3 horas.

Anexo:

A continuación, se muestra una función en Python que genera datos aleatorios para el proyecto de Regresión Lineal. La función crea un conjunto de puntos (x, y) donde x se genera de forma aleatoria en un intervalo definido (por ejemplo, de 0 a 10) y y se genera aplicando un modelo lineal conocido (en este ejemplo, $y = 4 + 3x$) al que se añade ruido gaussiano para simular datos reales. Este código se ubicará en la sección de "Representación de Datos" del proyecto y es el mismo para todos los alumnos.

```
import numpy as np
import matplotlib.pyplot as plt
def generar_datos_sinteticos(n=100, intervalo=(0, 10), coeficiente=3, intercepto=4, ruido_std=1,
seed=None):
"""
Genera datos sintéticos para un problema de regresión lineal.
Parámetros:
    - n: Número de datos a generar (por defecto 100)
    - intervalo: Tupla que define el intervalo de los valores de x (por ejemplo, (0, 10))
    - coeficiente: Coeficiente de x en la relación lineal (por defecto 3)
    - intercepto: Término independiente de la relación lineal (por defecto 4)
    - ruido_std: Desviación estándar del ruido gaussiano (por defecto 1)
    - seed: Valor para la semilla del generador de números aleatorios (opcional)
Retorna:
    - x: Array de valores de la variable independiente.
    - y: Array de valores de la variable dependiente calculados como: y = intercepto + coeficiente * x +
ruido.
"""
    if seed is not None:
        np.random.seed(seed)
    # Generar n datos de x de forma uniforme dentro del intervalo especificado
    x = np.random.uniform(intervalo[0], intervalo[1], n)
    # Generar ruido gaussiano
    ruido = np.random.normal(0, ruido_std, n)
    # Calcular y aplicando el modelo lineal y agregando el ruido
    y = intercepto + coeficiente * x + ruido
    return x, y
# Ejemplo de uso:
if __name__ == '__main__':
    # Generar 100 datos con parámetros por defecto y semilla para reproducibilidad
    x, y = generar_datos_sinteticos(n=100, seed=42)
    # Visualizar los datos generados
    plt.scatter(x, y, color="blue", label="Datos Sintéticos")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.title("Datos Sintéticos para Regresión Lineal: y = 4 + 3x + ruido")
    plt.legend()
    plt.show()
```

Anexo:

Explicación del Código

- **Importación de Módulos:**
  Se importan numpy y matplotlib.pyplot para generar los datos y graficarlos.
- **Función generar_datos_sinteticos:**
  - Parámetros:
    - n: Número de datos a generar.
    - intervalo: Rango de valores para la variable independiente x.
    - coeficiente e intercepto: Parámetros de la relación lineal.
    - ruido_std: La desviación estándar del ruido gaussiano que se añade a los valores de y.
    - seed: Permite fijar una semilla para obtener resultados reproducibles.
  - Se genera el vector x de forma uniforme en el intervalo dado.
  - Se genera un vector de ruido aleatorio con distribución normal.
  - Se calcula y aplicando la ecuación y=intercepto+coeficiente×x+ruidoy = \text{intercepto} + \text{coeficiente} \times x + \text{ruido}.
- **Visualización:**
  En la sección principal (if __name__ == '__main__':), se generan 100 datos y se grafica la dispersión para visualizar la relación y el ruido aplicado.

Esta función permite estandarizar la generación de datos para que todos los alumnos trabajen con el mismo conjunto de datos o utilicen parámetros controlables.

Rúbrica:

Indicador de logro / criterio	Insuficiente (0%-20%)	Por lograrlo (21%-40%)	Medianamente logrado (41%-60%)	Logrado (61%-80%)	Sobresaliente (81%-100%)
1. Representación y generación de datos	No se generan datos o son incorrectos.	Datos generados con errores y sin control de parámetros.	Se usan funciones de NumPy, pero con estructura poco clara.	Datos generados correctamente con parámetros definidos y visualización básica.	Datos bien estructurados, parametrizados, reproducibles y visualizados con claridad.
2. Resolución de sistemas y transformaciones	No se implementa o el resultado es erróneo.	Implementación parcial con errores.	Sistema resuelto y transformaciones básicas sin buena explicación.	Resolución correcta con funciones de NumPy y transformaciones explicadas.	Implementación robusta, validaciones y ejemplos claros con visualización.
3. Cálculo diferencial y optimización	No se derivan funciones ni se optimiza.	Derivación incompleta o con errores; sin uso correcto de librerías.	Derivación y gradiente aplicados con errores menores.	Derivación con SymPy y optimización implementada con análisis.	Aplicación completa con análisis detallado y visualización del proceso.
4. Optimización de código y profiling	Código ineficiente, sin análisis de rendimiento.	Optimización mínima sin evidencia clara.	Algunas técnicas aplicadas, análisis parcial.	Uso correcto de vectorización, Numba, y análisis de tiempos con gráficos.	Código optimizado con múltiples técnicas y mejoras claramente documentadas.
5. Manejo de excepciones y validaciones	No se manejan errores; el programa falla.	Manejo básico con cobertura limitada.	Se manejan algunos errores comunes con bloques try-except.	Buen manejo de errores y validaciones, con excepciones personalizadas.	Manejo completo con flujos bien definidos, excepciones personalizadas y mensajes claros.
6. Diseño POO y principios SOLID	No se usa POO o está mal implementada.	POO básica sin aplicar principios correctamente.	Uso de clases y algunos principios, pero con diseño poco eficiente.	Diseño modular con POO y algunos principios SOLID aplicados.	Diseño limpio, modular, con uso completo de POO y principios SOLID, respaldado por diagramas.
7. Documentación, presentación y evidencias	Sin documentación ni evidencias.	Documentación mínima y desorganizada.	Documentación aceptable con evidencias limitadas.	Documentación clara, con capturas y justificación de decisiones.	Informe completo, con evidencias gráficas de calidad y explicación detallada.