UNIVERSIDAD NACIONAL DE COLOMBIA

Fundamentos de Computación Científica

Lógica Algorítmica y Vectorización para IA Agroindustrial

Curso de IA Aplicada al Agro

11 de enero de 2026

Índice

Prefacio: El Ingeniero vs. El Programador

En la Semana 1 dominaste la terminal. Ahora, en la Semana 2, dejaremos de escribir "scripts" para empezar a construir **Ingeniería de Datos**.

Python es el lenguaje estándar de la IA, no por su velocidad (es lento), sino por su capacidad de conectarse con librerías de alto rendimiento escritas en C, como **NumPy**.

Al finalizar este módulo, entenderás:

- La matemática detrás de las decisiones lógicas ($A \wedge B$).
- Por qué los bucles `for` destruyen el rendimiento en Big Data.
- Cómo procesar imágenes satelitales usando matrices (Tensores).

1. Capítulo I: Lógica Algorítmica y Complejidad

La programación es la automatización de la lógica matemática. Un ingeniero no solo escribe código que funciona; escribe código que *escala*.

1.1. Variables y Tipos de Datos

En álgebra, x es una incógnita. En computación, una variable es una **dirección de memoria**.

Listing 1: Tipos de datos en Agro-IA

```
ph_suelo = 5.8          # float (Reales)
id_sensor = 1024        # int (Enteros)
zona = "Lote Norte"     # str (Cadenas)
riego_activo = True     # bool (Lógica Binaria)
```

1.2. Lógica Booleana (El cerebro de la IA)

Las decisiones se basan en tablas de verdad. Los operadores Python tienen equivalentes matemáticos exactos:

- `and` $\leftrightarrow$ Conjunción ($A \wedge B$)
- `or` $\leftrightarrow$ Disyunción ($A \vee B$)
- `not` $\leftrightarrow$ Negación ($\neg A$)

Listing 2: Lógica de Control de Riego

```
# Regla: Regar SI (Humedad < 30%) Y (No está lloviendo)
if (humedad < 30) and (not lloviendo):
    activar_bombas()
```

1.3. Complejidad Computacional (Big O)

Aquí es donde se separa al novato del experto. ¿Qué pasa si tienes que analizar 1 millón de plantas?

El Peligro de los Bucles Anidados ($O(N^2)$)

Si tienes una lista de N parcelas y usas un bucle dentro de otro bucle, el tiempo de ejecución crece cuadráticamente.

$$N = 10,000 \rightarrow \text{Operaciones} = 100,000,000$$

Regla: En Python para IA, evita los bucles `for` siempre que sea posible.

Nota de Ingeniería: En Big Data, ignoramos las constantes. Nos importa la tendencia cuando $N \rightarrow \infty$.

2. Capítulo II: Estructuras de Datos y Memoria

2.1. Listas vs. Arrays (La verdad oculta)

Una lista de Python `[1, 2, 3]` es flexible pero ineficiente. En la memoria RAM, es una colección de *punteros* dispersos. El procesador pierde tiempo buscando cada dato.

2.2. NumPy: La base del Deep Learning

NumPy crea **Arrays**: bloques contiguos de memoria optimizada (como en C o Fortran). Esto permite operaciones SIMD (Single Instruction, Multiple Data).

Comparativa de Velocidad

Procesar 1 millón de datos:

- **Python List:** $\approx$ 300 ms (Milisegundos)
- **NumPy Array:** $\approx$ 5 ms

¡Es 60 veces más rápido! Fundamental para entrenar IAs.

2.3. Vectorización y Broadcasting

En lugar de iterar por cada elemento, operamos sobre toda la matriz a la vez.

Listing 3: Vectorización: El estilo NumPy

```python
import numpy as np

# Matriz 1000x1000 simulando humedad del terreno
terreno = np.random.rand(1000, 1000)

# PROBLEMA: Calibrar sensores sumando 0.1 a todo el terreno
# FORMA INCORRECTA (Lenta - O(N^2)):
# for i in rows: for j in cols: terreno[i][j] += 0.1

# FORMA CORRECTA (Vectorizada - Instantánea):
terreno_calibrado = terreno + 0.1
```

Caso de Uso Real: Imágenes Satelitales

Una imagen satelital no es más que una matriz de números (R, G, B, Infrarrojo). Detectar vegetación (NDVI) es simplemente una operación matricial:

$$NDVI = \frac{(NIR - RED)}{(NIR + RED)}$$

Con NumPy, esta fórmula se aplica a millones de píxeles en milisegundos.

3. Capítulo III: Taller Práctico - Simulación de Cultivos

Vamos a crear un sistema que simule el crecimiento de cultivos en una matriz, aplicando factores aleatorios (clima).

3.1. Requerimientos

Crear un script `simulacion.py` que:

1. Genere un campo de 50×50 parcelas con salud inicial aleatoria (0.0 a 1.0).
2. Simule una "plaga"que reduce la salud en un 20 % en zonas aleatorias.
3. Calcule estadísticas finales (promedio, desviación estándar).

Listing 4: simulacion.py (Snippet)

```python
import numpy as np

# 1. Crear campo (50x50)
campo = np.random.uniform(0.5, 1.0, (50, 50))

# 2. Simular Plaga (Máscara Booleana)
# La plaga afecta al 10% del campo aleatoriamente
zona_plaga = np.random.rand(50, 50) < 0.1
campo[zona_plaga] = campo[zona_plaga] - 0.2

# 3. Reporte
print(f"Salud Promedio: {np.mean(campo):.2f}")
print(f"Parcelas Críticas (<0.4): {np.sum(campo < 0.4)}")
```

Próximos Pasos

En la Semana 3, conectaremos esto con el mundo real:

- Leer archivos CSV gigantes con **Pandas**.
- Limpieza de datos (Data Cleaning).
- Visualización científica con **Matplotlib**.