Guía 0→100: MS in AI Pathway

De Python Básico a Machine Learning y Deep Learning
Preparación para CU Boulder MS in Artificial Intelligence

DUQUEOM · 2025 · Versión 2.0
⭐ Enfoque: Probabilidad, Estadística, ML, Deep Learning

Guía MS in AI Pathway

Guía MS in AI Pathway

DUQUEOM · 2025

Guía 0→100: MS in AI Pathway

> **De Python Básico a Candidato del MS in AI de CU Boulder**
> **6 meses | 6h/día | 100% enfocado en las 6 materias del Pathway**

🎯 Objetivo Único

Prepararte para aprobar las **6 materias obligatorias** del Performance-Based Admission Pathway:

⭐ Línea 1: Machine Learning (3 créditos)

- Introduction to ML: Supervised Learning
- Unsupervised Algorithms in ML
- Introduction to Deep Learning

⭐ Línea 2: Probabilidad y Estadística (3 créditos)

- Probability Fundamentals for Data Science and AI
- Discrete-Time Markov Chains and Monte Carlo Methods
- Statistical Estimation for Data Science and AI

📚 Índice Completo

👉 **Ver todos los módulos →**

📋 Los 10 Módulos Obligatorios

#	Módulo	Fase	Semanas
01	Python Profesional	Fundamentos	2
02	OOP desde Cero	Fundamentos	2
03	Álgebra Lineal para ML	Fundamentos	2
04	**Fundamentos de Probabilidad**	⭐ Pathway L2	3
05	**Estadística Inferencial**	⭐ Pathway L2	3
06	**Markov y Monte Carlo**	⭐ Pathway L2	2
07	**ML Supervisado**	⭐ Pathway L1	3
08	**ML No Supervisado**	⭐ Pathway L1	2
09	**Deep Learning**	⭐ Pathway L1	3
10	Proyecto Integrador	Final	4

Total: 26 semanas = 6 meses

📌 Restricciones

- ✅ **Python puro** - Sin numpy, pandas, sklearn
- ✅ **100% local** - Todo se ejecuta en tu máquina

- ✅ **Desde cero** - Cada algoritmo implementado manualmente

🔗 Enlaces

- MS in AI - CU Boulder
- Pathway de Admisión

🚀 Comenzar

→ Módulo 01: Python Profesional

> 💡 **Filosofía:** Si puedes implementar Naive Bayes, K-Means, MLP y Markov desde cero, estás listo para el Pathway.

Índice de Módulos

Guía MS in AI Pathway

DUQUEOM · 2025

📚 Guía 0→100: MS in AI Pathway

De Python Básico a Candidato del MS in AI de CU Boulder
6 meses | 6h/día | 100% enfocado en las 6 materias del Pathway

---

🎯 Objetivo Único de Esta Guía

Prepararte para aprobar las **6 materias obligatorias** del Performance-Based Admission Pathway:

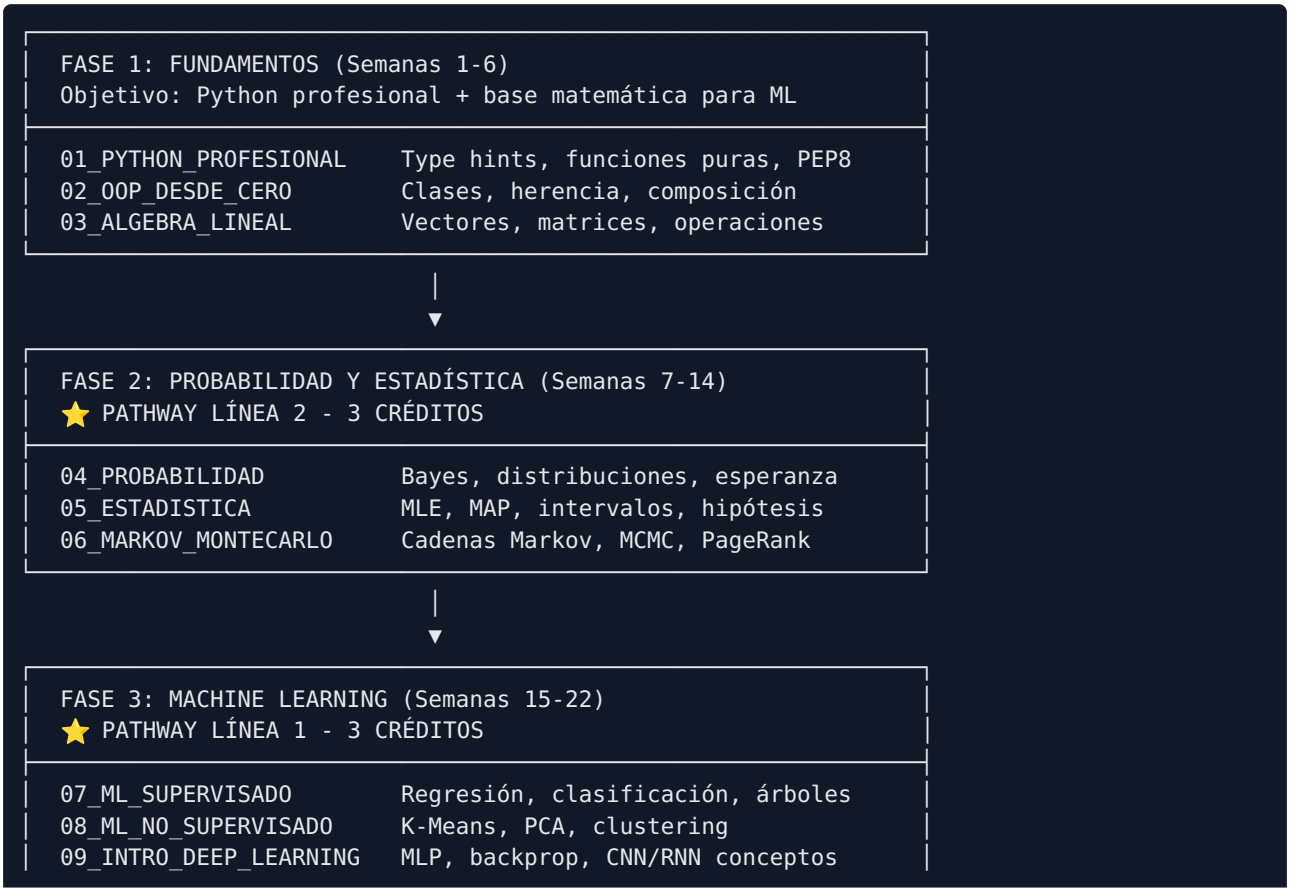
⭐ Línea 1: Aprendizaje Automático (3 créditos)

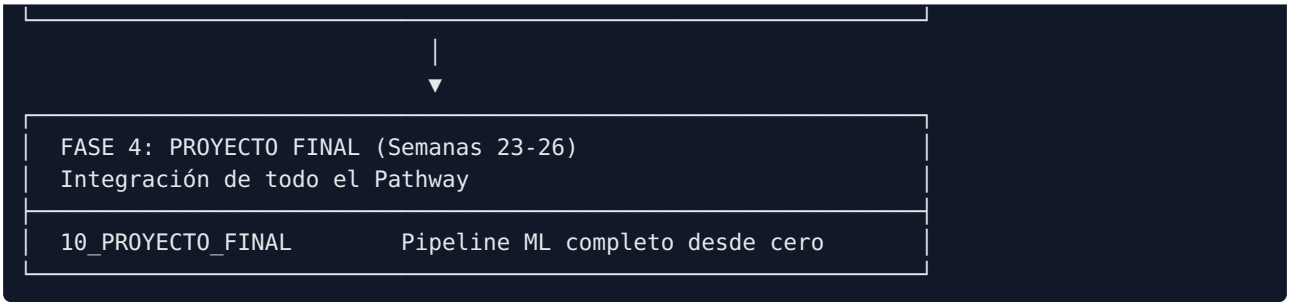
Curso del Pathway	Módulo de Esta Guía
Introduction to Machine Learning: Supervised Learning	**07_ML_SUPERVISADO**
Unsupervised Algorithms in Machine Learning	**08_ML_NO_SUPERVISADO**
Introduction to Deep Learning	**09_INTRO_DEEP_LEARNING**

⭐ Línea 2: Probabilidad y Estadística (3 créditos)

Curso del Pathway	Módulo de Esta Guía
Probability Fundamentals for Data Science and AI	**04_PROBABILIDAD**
Discrete-Time Markov Chains and Monte Carlo Methods	**06_MARKOV_MONTECARLO**
Statistical Estimation for Data Science and AI	**05_ESTADISTICA**

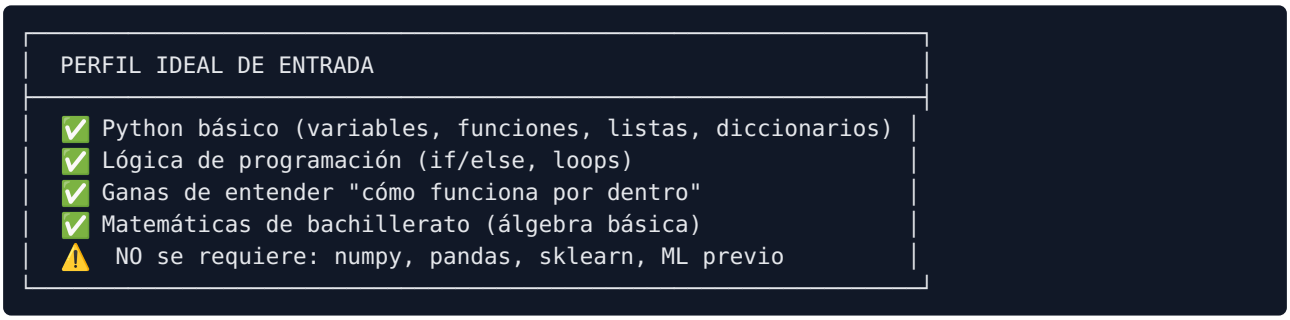
📋 Estructura del Programa (TODO OBLIGATORIO)





Total: 10 módulos obligatorios | 26 semanas | ~6 meses

👤 Perfil de Entrada



📖 Módulos Obligatorios

FASE 1: Fundamentos (Semanas 1-6)

Base de programación profesional necesaria para implementar ML

#	Módulo	Descripción	Tiempo	Archivo
01	Python Profesional	Type hints, funciones puras, PEP8	2 sem	01_PYTHON_PROFESIONAL.md
02	OOP desde Cero	Clases, herencia, composición	2 sem	02_OOP_DESDE_CERO.md
03	Álgebra Lineal para ML	Vectores, matrices, operaciones	2 sem	10_ALGEBRA_LINEAL.md

Entregable: Clase `Vector` y `Matrix` con operaciones básicas desde cero.

FASE 2: Probabilidad y Estadística (Semanas 7-14) ⭐ PATHWAY LÍNEA 2

Preparación directa para los 3 cursos de Probability & Statistics

#	Módulo	Curso del Pathway	Tiempo	Archivo
04	Fundamentos de Probabilidad	Probability Fundamentals for DS and AI	3 sem	19_PROBABILIDAD_FUNDAMENTOS.md
05	Estadística Inferencial	Statistical Estimation for DS and AI	3 sem	20_ESTADISTICA_INFERENCIAL.md
06	Markov y Monte Carlo	Discrete-Time Markov Chains and Monte Carlo	2 sem	21_CADENAS_MARKOV_MONTECARLO.md

Entregable: Implementación de Bayes, MLE, MCMC, PageRank desde cero.

FASE 3: Machine Learning (Semanas 15-22) ⭐ PATHWAY LÍNEA 1

Preparación directa para los 3 cursos de Machine Learning

#	Módulo	Curso del Pathway	Tiempo	Archivo
07	ML Supervisado	Introduction to ML: Supervised Learning	3 sem	22_ML_SUPERVISADO.md
08	ML No Supervisado	Unsupervised Algorithms in ML	2 sem	23_ML_NO_SUPERVISADO.md
09	Deep Learning	Introduction to Deep Learning	3 sem	24_INTRO_DEEP_LEARNING.md

Entregable: Regresión, Naive Bayes, K-Means, MLP con backprop desde cero.

FASE 4: Proyecto Final (Semanas 23-26)

Integración de todo lo aprendido en un pipeline ejecutable

#	Módulo	Descripción	Tiempo	Archivo
10	Proyecto Integrador	Pipeline ML completo	4 sem	12_PROYECTO_INTEGRADOR.md

Entregable: Sistema que clasifica texto usando NB, KMeans, MLP y genera texto con Markov.

🔨 Proyecto Final: ML Pipeline

```
ml-pathway-project/
├── src/
│   ├── __init__.py
│   ├── vector.py              # Álgebra lineal (Módulo 03)
│   ├── probability.py         # Bayes, distribuciones (Módulo 04)
│   ├── statistics.py          # MLE, intervalos (Módulo 05)
│   ├── markov.py              # Cadenas Markov, MCMC (Módulo 06)
│   ├── naive_bayes.py         # Clasificador NB (Módulo 07)
│   ├── linear_regression.py   # Regresión (Módulo 07)
│   ├── kmeans.py              # Clustering (Módulo 08)
│   ├── pca.py                 # Reducción dim (Módulo 08)
│   ├── neural_network.py      # MLP + backprop (Módulo 09)
│   ├── activations.py         # Funciones activación (Módulo 09)
│   └── pipeline.py            # Integración (Módulo 10)
├── tests/
│   └── test_*.py              # Tests para cada módulo
├── data/
│   └── sample_texts/          # Datos de prueba
├── notebooks/
│   └── demo.ipynb             # Demo interactivo
├── README.md
└── requirements.txt           # Solo pytest (sin numpy/sklearn)
```

⏱️ Tiempo Total

Fase	Semanas	Horas (~36h/sem)
Fundamentos (01-03)	6	~216h
Probabilidad (04-06)	8	~288h

Fase	Semanas	Horas (~36h/sem)
Machine Learning (07-09)	8	~288h
Proyecto Final (10)	4	~144h
TOTAL	**26**	**~936h**

Duración: 6 meses con 6h/día (L-S)

📦 Material Complementario (Opcional)

Documento	Descripción	Obligatorio
EJERCICIOS.md	Práctica adicional por módulo	Recomendado
GLOSARIO.md	Definiciones técnicas	Consulta
SIMULACRO_ENTREVISTA.md	Preguntas tipo Pathway	Recomendado
RECURSOS.md	Cursos y libros externos	Consulta

DSA Avanzado (Solo si necesitas para entrevistas técnicas)

Estos módulos **NO son necesarios para el Pathway**, pero pueden ser útiles para entrevistas de trabajo:

Documento	Tema
04_ARRAYS_STRINGS.md	Arrays y manipulación
05_HASHMAPS_SETS.md	Hash tables
07_RECURSION.md	Recursión
08_SORTING.md	Ordenamiento
14_TREES.md	Árboles y BST
15_GRAPHS.md	Grafos, BFS, DFS
16_DYNAMIC_PROGRAMMING.md	DP

🚀 Comenzar

→ Módulo 01: Python Profesional

📌 Restricciones del Proyecto

- ✅ **Python puro** - Sin numpy, pandas, sklearn, tensorflow
- ✅ **100% local** - Todo se ejecuta en tu máquina
- ✅ **Desde cero** - Cada algoritmo implementado manualmente
- ✅ **Enfocado** - Solo lo necesario para el Pathway

> 💡 **Filosofía:** Si puedes implementar Naive Bayes, K-Means, MLP y Markov desde cero, estás listo para los cursos del Pathway. DSA avanzado es útil para entrevistas, pero **no es el objetivo de esta guía**.

Módulo 01 - Python Profesional

Guía MS in AI Pathway

DUQUEOM · 2025

Módulo 01 - Python Profesional

> 🎯 **Objetivo:** Transformar código Python funcional en código profesional con type hints, funciones puras y estándares de la industria
> **Fase:** Fundamentos | **Prerrequisito para:** Todos los módulos siguientes

---

🧠 Analogía: El Arquitecto vs El Albañil

```
ALBAÑIL                          ARQUITECTO
────────                         ──────────
"Pon ladrillos aquí"             "Plano estructural con medidas"
Funciona, pero no escala         Cualquiera puede construirlo
Solo él sabe cómo                Verificable y mantenible

def process(x):                  def process(data: list[int]) -> int:
    return x + 1                     """Sum all positive numbers."""
                                     return sum(n for n in data if n>0)
```

El código profesional es como un plano arquitectónico: cualquier ingeniero puede leerlo, entenderlo y construir a partir de él.

📋 Contenido

---

1. Type Hints: Documentación Ejecutable {#1-type-hints}

1.1 ¿Por qué Type Hints?

```
SIN TYPE HINTS
──────────────
def tokenize(text):     # ¿text es str? ¿bytes? ¿list?
    return text.split() # ¿Retorna list? ¿set? ¿generator?

CON TYPE HINTS
──────────────
def tokenize(text: str) -> list[str]:
    return text.split()  # Claro: recibe str, retorna list
```

Beneficios:
- 📖 Documentación que no se desactualiza
- 🐛 Errores detectados antes de ejecutar (con `mypy`)

- 🤖 Autocompletado inteligente en el IDE
- 🔍 Código más fácil de leer y mantener

1.2 Tipos Básicos

```
# Tipos primitivos
name: str = "Archimedes"
count: int = 42
ratio: float = 3.14159
is_active: bool = True
nothing: None = None

# Colecciones (Python 3.9+)
words: list[str] = ["hello", "world"]
scores: dict[str, float] = {"doc1": 0.85, "doc2": 0.92}
unique_words: set[str] = {"the", "and", "or"}
coordinates: tuple[float, float] = (10.5, 20.3)
```

1.3 Tipos en Funciones

```
# ❌ ANTES: ¿Qué recibe? ¿Qué retorna?
def clean(text):
    return text.lower().strip()

# ✅ DESPUÉS: Claro y verificable
def clean(text: str) -> str:
    """Remove whitespace and convert to lowercase."""
    return text.lower().strip()
```

1.4 Tipos Avanzados

```
from typing import Optional, Union

# Optional: puede ser el tipo o None
def find_document(doc_id: int) -> Optional[str]:
    """Return document content or None if not found."""
    if doc_id in documents:
        return documents[doc_id]
    return None

# Union: puede ser uno de varios tipos (Python 3.10+ usa |)
def process(data: Union[str, list[str]]) -> list[str]:
    """Accept string or list of strings."""
    if isinstance(data, str):
        return [data]
    return data

# Python 3.10+ syntax
def process_modern(data: str | list[str]) -> list[str]:
    if isinstance(data, str):
        return [data]
    return data
```

1.5 Type Hints para Clases

```
class Document:
    def __init__(self, doc_id: int, content: str) -> None:
        self.doc_id: int = doc_id
        self.content: str = content
        self.tokens: list[str] = []
```

```
    def tokenize(self) -> list[str]:
        """Split content into tokens."""
        self.tokens = self.content.lower().split()
        return self.tokens

    def word_count(self) -> int:
        """Return number of tokens."""
        return len(self.tokens)
```

1.6 Verificación con mypy

```
# Instalar mypy
pip install mypy

# Verificar un archivo
mypy src/document.py

# Verificar todo el proyecto
mypy src/

# Configuración en pyproject.toml
```

```
# pyproject.toml
[tool.mypy]
python_version = "3.11"
warn_return_any = true
warn_unused_ignores = true
disallow_untyped_defs = true
```

---

2. Funciones Puras vs Impuras {#2-funciones-puras}

2.1 ¿Qué es una Función Pura?

```
┌─────────────────────────────────────────────────────────┐
│  FUNCIÓN PURA                                           │
│  ────────────                                           │
│  1. Mismo input → siempre mismo output                  │
│  2. Sin efectos secundarios (no modifica estado externo)│
│                                                         │
│  VENTAJAS:                                              │
│  ✅ Fácil de testear                                    │
│  ✅ Fácil de entender                                   │
│  ✅ Paralelizable                                       │
│  ✅ Cacheable (memoization)                             │
└─────────────────────────────────────────────────────────┘
```

2.2 Ejemplos Comparativos

```
# ❌ IMPURA: modifica estado externo
results = []

def add_result_impure(value):
    results.append(value)  # Modifica lista externa
    return len(results)

# ✅ PURA: retorna nuevo valor sin modificar nada
def add_result_pure(results: list[int], value: int) -> list[int]:
    return results + [value]  # Retorna nueva lista
```

```
# ❌ IMPURA: depende de estado externo
multiplier = 2

def multiply_impure(x):
    return x * multiplier  # Depende de variable externa

# ✅ PURA: todo lo necesario viene como parámetro
def multiply_pure(x: int, multiplier: int) -> int:
    return x * multiplier
```

2.3 Evitar Mutación de Argumentos

```
# ❌ PELIGROSO: modifica el argumento original
def remove_stopwords_bad(tokens: list[str], stopwords: set[str]) -> list[str]:
    for word in list(tokens):  # Itera sobre copia para poder modificar
        if word in stopwords:
            tokens.remove(word)  # ¡Modifica la lista original!
    return tokens

# ✅ SEGURO: crea nueva lista
def remove_stopwords_good(tokens: list[str], stopwords: set[str]) -> list[str]:
    return [word for word in tokens if word not in stopwords]
```

2.4 Cuándo las Funciones Impuras Son Necesarias

Algunas operaciones requieren efectos secundarios:
- Escribir a disco
- Imprimir a consola
- Conectar a base de datos
- Generar números aleatorios

Estrategia: Aislar las funciones impuras y mantener la lógica de negocio pura.

```
# Lógica pura (testeable)
def prepare_document(content: str) -> dict[str, any]:
    tokens = content.lower().split()
    return {
        "tokens": tokens,
        "word_count": len(tokens),
        "char_count": len(content)
    }

# Función impura aislada
def save_document(doc_data: dict[str, any], filepath: str) -> None:
    with open(filepath, 'w') as f:
        json.dump(doc_data, f)
```

---

3. PEP8 y Estilo Consistente {#3-pep8}

3.1 Reglas Esenciales

Regla	Ejemplo Correcto
Indentación: 4 espacios	`def func():⏎····code`
Línea máxima: 88-100 caracteres	Configurar en linter
Espacios alrededor de operadores	`x = 1 + 2` (no `x=1+2`)

Regla	Ejemplo Correcto
Nombres de variables: snake_case	`word_count`, `doc_id`
Nombres de clases: PascalCase	`Document`, `InvertedIndex`
Constantes: UPPER_CASE	`MAX_TOKENS = 1000`

3.2 Nombres Descriptivos

```
# ❌ Nombres crípticos
def proc(d):
    r = []
    for i in d:
        if len(i) > 3:
            r.append(i)
    return r

# ✅ Nombres descriptivos
def filter_short_words(tokens: list[str], min_length: int = 3) -> list[str]:
    """Remove tokens shorter than min_length."""
    return [token for token in tokens if len(token) > min_length]
```

3.3 Configurar Linter (ruff)

```
# Instalar ruff (rápido y moderno)
pip install ruff

# Verificar código
ruff check src/

# Corregir automáticamente
ruff check --fix src/
```

```
# pyproject.toml
[tool.ruff]
line-length = 88
select = ["E", "F", "W", "I", "N", "UP"]

[tool.ruff.per-file-ignores]
"tests/*" = ["S101"]  # Permitir assert en tests
```

---

4. Docstrings Profesionales {#4-docstrings}

4.1 Formato Google Style

```
def compute_tf(term: str, document: list[str]) -> float:
    """Compute Term Frequency for a term in a document.

    Term Frequency measures how often a term appears in a document,
    normalized by the total number of terms.

    Args:
        term: The word to search for.
        document: List of tokens in the document.

    Returns:
        The term frequency as a float between 0 and 1.

    Raises:
```

```
        ValueError: If document is empty.

    Example:
        >>> compute_tf("hello", ["hello", "world", "hello"])
        0.6666666666666666
    """
    if not document:
        raise ValueError("Document cannot be empty")

    count = document.count(term)
    return count / len(document)
```

4.2 Docstrings para Clases

```
class Document:
    """Represents a text document with metadata.

    A Document holds the original content along with processed
    tokens and provides methods for text analysis.

    Attributes:
        doc_id: Unique identifier for the document.
        content: Original text content.
        tokens: List of processed tokens (populated after tokenize()).

    Example:
        >>> doc = Document(1, "Hello World")
        >>> doc.tokenize()
        ['hello', 'world']
    """

    def __init__(self, doc_id: int, content: str) -> None:
        """Initialize a Document.

        Args:
            doc_id: Unique identifier.
            content: Raw text content.
        """
        self.doc_id = doc_id
        self.content = content
        self.tokens: list[str] = []
```

---

5. Configuración de Herramientas {#5-configuracion}

5.1 pyproject.toml Completo

```
[project]
name = "archimedes-indexer"
version = "0.1.0"
description = "A search engine built from scratch"
requires-python = ">=3.11"

[tool.mypy]
python_version = "3.11"
warn_return_any = true
warn_unused_ignores = true
disallow_untyped_defs = true
ignore_missing_imports = true

[tool.ruff]
```

```
line-length = 88
select = [
    "E",   # pycodestyle errors
    "F",   # pyflakes
    "W",   # pycodestyle warnings
    "I",   # isort
    "N",   # pep8-naming
    "UP",  # pyupgrade
]

[tool.pytest.ini_options]
testpaths = ["tests"]
python_files = "test_*.py"
```

5.2 Comandos de Verificación

```
# Verificar tipos
mypy src/

# Verificar estilo
ruff check src/

# Corregir estilo automáticamente
ruff check --fix src/

# Todo junto (crear en Makefile)
make check
```

5.3 Makefile Básico

```
.PHONY: check lint type-check test

check: lint type-check test

lint:
    ruff check src/ tests/

type-check:
    mypy src/

test:
    python -m pytest tests/ -v

fix:
    ruff check --fix src/ tests/
```

⚠️ Errores Comunes y Cómo Evitarlos

Error 1: Type hints incorrectos

```
# ❌ Error: list sin tipo genérico
def get_words(text: str) -> list:  # mypy warning
    return text.split()

# ✅ Correcto
def get_words(text: str) -> list[str]:
    return text.split()
```

Error 2: Mutar argumentos por defecto

```
# ❌ Bug clásico: lista mutable como default
def add_word(word: str, words: list[str] = []) -> list[str]:
    words.append(word)  # ¡Se acumula entre llamadas!
    return words

# ✅ Correcto: usar None como default
def add_word(word: str, words: list[str] | None = None) -> list[str]:
    if words is None:
        words = []
    return words + [word]
```

Error 3: Olvidar el return type en init

```
# ❌ Incompleto
def __init__(self, doc_id: int):
    self.doc_id = doc_id

# ✅ Completo (siempre -> None)
def __init__(self, doc_id: int) -> None:
    self.doc_id = doc_id
```

🔧 Ejercicios Prácticos

Ejercicio 1.1: Tipar una Función

Ver EJERCICIOS.md - Agregar type hints a función de tokenización.

Ejercicio 1.2: Convertir a Función Pura

Ver EJERCICIOS.md - Refactorizar función impura.

Ejercicio 1.3: Configurar Linters

Ver EJERCICIOS.md - Crear pyproject.toml completo.

Ejercicio 1.4: Escribir Docstrings

Ver EJERCICIOS.md - Documentar módulo completo.

📚 Recursos Externos

Recurso	Tipo	Prioridad
Real Python: Type Checking	Tutorial	🔴 Obligatorio
PEP 8	Documentación	🔴 Obligatorio
mypy Documentation	Documentación	🟡 Recomendado
Google Python Style Guide	Guía	🟢 Complementario

🔗 Referencias del Glosario

- Type Hint
- Función Pura

- PEP8
- Docstring
- Linter

🧭 Navegación

← Anterior	Índice	Siguiente →
-	00_INDICE	02_OOP_DESDE_CERO

Módulo 02 - OOP desde Cero

Guía MS in AI Pathway

DUQUEOM · 2025

Módulo 02 - OOP desde Cero

> 🎯 **Objetivo:** Diseñar clases profesionales que representen documentos y colecciones, aplicando principios SOLID básicos
> **Fase:** Fundamentos | **Prerrequisito para:** Todos los módulos siguientes

---

🧠 Analogía: La Fábrica de Documentos

📋 Contenido

---

1. Clases y Objetos Básicos {#1-clases-basicas}

1.1 Anatomía de una Clase

```
class Document:
    """Represents a single document in the corpus."""

    # Atributo de clase (compartido por todas las instancias)
    document_count: int = 0

    def __init__(self, doc_id: int, content: str) -> None:
        """Initialize a new Document.

        Args:
            doc_id: Unique identifier for this document.
            content: Raw text content of the document.
        """
        # Atributos de instancia (únicos para cada objeto)
        self.doc_id: int = doc_id
        self.content: str = content
        self.tokens: list[str] = []
```

```
        # Incrementar contador de clase
        Document.document_count += 1

    def tokenize(self) -> list[str]:
        """Split content into lowercase tokens.

        Returns:
            List of tokens extracted from content.
        """
        self.tokens = self.content.lower().split()
        return self.tokens

    def word_count(self) -> int:
        """Return the number of tokens.

        Note:
            Must call tokenize() first, or returns 0.
        """
        return len(self.tokens)
```

1.2 Creando y Usando Objetos

```
# Crear instancias (objetos)
doc1 = Document(1, "Hello World")
doc2 = Document(2, "Goodbye World")

# Llamar métodos
doc1.tokenize()
print(doc1.tokens)  # ['hello', 'world']
print(doc1.word_count())  # 2

# Acceder al atributo de clase
print(Document.document_count)  # 2
```

1.3 Self: La Referencia al Objeto Actual

```
┌──────────────────────────────────────────────────────────────┐
│  self = "yo mismo"                                           │
│                                                              │
│  Cuando llamas doc1.tokenize(), Python traduce a:            │
│  Document.tokenize(doc1)                                     │
│                                                              │
│  self es simplemente el objeto sobre el que se llama el método  │
└──────────────────────────────────────────────────────────────┘
```

---

2. Métodos Mágicos (Dunder Methods) {#2-metodos-magicos}

2.1 Los Más Importantes

Método	Cuándo se llama	Propósito
`__init__`	Al crear objeto	Inicializar atributos
`__repr__`	`repr(obj)`, debugger	Representación técnica
`__str__`	`str(obj)`, `print(obj)`	Representación legible
`__eq__`	`obj1 == obj2`	Comparar igualdad
`__len__`	`len(obj)`	Retornar "longitud"

Método	Cuándo se llama	Propósito
`__iter__`	`for x in obj`	Hacer iterable

2.2 Implementación Completa

```
class Document:
    def __init__(self, doc_id: int, content: str) -> None:
        self.doc_id = doc_id
        self.content = content
        self.tokens: list[str] = []

    def __repr__(self) -> str:
        """Technical representation for debugging.

        Example:
            >>> doc = Document(1, "Hello World")
            >>> repr(doc)
            "Document(doc_id=1, content='Hello World')"
        """
        return f"Document(doc_id={self.doc_id}, content='{self.content[:20]}...')"

    def __str__(self) -> str:
        """Human-readable representation.

        Example:
            >>> print(doc)
            Document #1: Hello World (2 words)
        """
        word_count = len(self.tokens) if self.tokens else "not tokenized"
        return f"Document #{self.doc_id}: {self.content[:30]}... ({word_count} words)"

    def __eq__(self, other: object) -> bool:
        """Check equality based on doc_id.

        Two documents are equal if they have the same doc_id.
        """
        if not isinstance(other, Document):
            return NotImplemented
        return self.doc_id == other.doc_id

    def __len__(self) -> int:
        """Return number of tokens (after tokenization)."""
        return len(self.tokens)

    def __hash__(self) -> int:
        """Make Document hashable (usable in sets/dicts)."""
        return hash(self.doc_id)
```

2.3 Uso de Métodos Mágicos

```
doc = Document(1, "Hello World from Archimedes")
doc.tokenize()

# __repr__ (en debugger o consola)
>>> doc
Document(doc_id=1, content='Hello World from Arc...')

# __str__ (con print)
>>> print(doc)
Document #1: Hello World from Archimedes... (4 words)
```

```
# __len__
>>> len(doc)
4

# __eq__
doc2 = Document(1, "Different content")
>>> doc == doc2
True  # Mismo doc_id

# __hash__ permite usar en sets
>>> docs_set = {doc, doc2}
>>> len(docs_set)
1  # Son "iguales" por doc_id
```

---

3. Properties y Encapsulamiento {#3-properties}

3.1 ¿Por Qué Encapsular?

```
┌─────────────────────────────────────────────────────┐
│  PROBLEMA: Acceso directo sin validación            │
│                                                     │
│  doc.doc_id = -5     # ¿ID negativo? ¡Inválido!     │
│  doc.content = None  # ¿Contenido None? ¡Error futuro!
│                                                     │
│  SOLUCIÓN: Properties con validación                │
│                                                     │
│  doc.doc_id = -5     # Lanza ValueError             │
│  doc.content = None  # Lanza TypeError              │
└─────────────────────────────────────────────────────┘
```

3.2 Implementando Properties

```
class Document:
    def __init__(self, doc_id: int, content: str) -> None:
        # Usar los setters para validar desde el inicio
        self._doc_id: int = 0  # Atributo "privado" (convención)
        self._content: str = ""

        # Estos llaman a los setters
        self.doc_id = doc_id
        self.content = content
        self.tokens: list[str] = []

    @property
    def doc_id(self) -> int:
        """Get document ID."""
        return self._doc_id

    @doc_id.setter
    def doc_id(self, value: int) -> None:
        """Set document ID with validation."""
        if not isinstance(value, int):
            raise TypeError(f"doc_id must be int, got {type(value).__name__}")
        if value < 0:
            raise ValueError(f"doc_id must be non-negative, got {value}")
        self._doc_id = value

    @property
    def content(self) -> str:
        """Get document content."""
```

```
        return self._content

    @content.setter
    def content(self, value: str) -> None:
        """Set content with validation."""
        if not isinstance(value, str):
            raise TypeError(f"content must be str, got {type(value).__name__}")
        if not value.strip():
            raise ValueError("content cannot be empty or whitespace only")
        self._content = value

    @property
    def is_tokenized(self) -> bool:
        """Check if document has been tokenized (read-only)."""
        return len(self.tokens) > 0
```

3.3 Uso de Properties

```
doc = Document(1, "Hello World")

# Lectura transparente (parece atributo normal)
print(doc.doc_id)  # 1

# Escritura con validación automática
doc.doc_id = 5     # OK
doc.doc_id = -1    # ValueError: doc_id must be non-negative

# Property de solo lectura
print(doc.is_tokenized)  # False
doc.tokenize()
print(doc.is_tokenized)  # True
# doc.is_tokenized = True  # AttributeError: can't set attribute
```

---

4. Composición vs Herencia {#4-composicion}

4.1 La Regla de Oro

```
┌─────────────────────────────────────────────────────────┐
│                                                         │
│   "Favor composition over inheritance"                  │
│   (Prefiere composición sobre herencia)                 │
│                                                         │
│   HERENCIA: "ES UN" (is-a)                              │
│   ─────────────────────────                             │
│   Un Perro ES UN Animal                                 │
│   ✅ Tiene sentido                                      │
│                                                         │
│   COMPOSICIÓN: "TIENE UN" (has-a)                       │
│   ──────────────────────────────                        │
│   Un Corpus TIENE Documentos                            │
│   ✅ Más flexible                                       │
│                                                         │
└─────────────────────────────────────────────────────────┘
```

4.2 Composición: Corpus Contiene Documents

```
class Corpus:
    """A collection of documents."""

    def __init__(self, name: str) -> None:
```

```
        """Initialize an empty corpus.

        Args:
            name: Name of this corpus.
        """
        self.name: str = name
        self._documents: dict[int, Document] = {}  # Composición: contiene Documents

    def add_document(self, doc: Document) -> None:
        """Add a document to the corpus.

        Args:
            doc: Document to add.

        Raises:
            ValueError: If document with same ID already exists.
        """
        if doc.doc_id in self._documents:
            raise ValueError(f"Document with id {doc.doc_id} already exists")
        self._documents[doc.doc_id] = doc

    def get_document(self, doc_id: int) -> Document | None:
        """Retrieve a document by ID.

        Args:
            doc_id: ID of document to retrieve.

        Returns:
            The Document if found, None otherwise.
        """
        return self._documents.get(doc_id)

    def remove_document(self, doc_id: int) -> bool:
        """Remove a document by ID.

        Returns:
            True if document was removed, False if not found.
        """
        if doc_id in self._documents:
            del self._documents[doc_id]
            return True
        return False

    def __len__(self) -> int:
        """Return number of documents in corpus."""
        return len(self._documents)

    def __iter__(self):
        """Iterate over documents."""
        return iter(self._documents.values())

    def __contains__(self, doc_id: int) -> bool:
        """Check if document ID exists."""
        return doc_id in self._documents
```

4.3 Cuándo Usar Herencia

La herencia es apropiada cuando hay una relación "es un" clara:

```
from abc import ABC, abstractmethod

class Tokenizer(ABC):
```

```
    """Abstract base class for tokenizers."""

    @abstractmethod
    def tokenize(self, text: str) -> list[str]:
        """Tokenize text into words."""
        pass

class SimpleTokenizer(Tokenizer):
    """Basic whitespace tokenizer."""

    def tokenize(self, text: str) -> list[str]:
        return text.lower().split()

class AdvancedTokenizer(Tokenizer):
    """Tokenizer that also removes punctuation."""

    def __init__(self, min_length: int = 2) -> None:
        self.min_length = min_length

    def tokenize(self, text: str) -> list[str]:
        # Remove punctuation
        cleaned = ''.join(c if c.isalnum() or c.isspace() else ' ' for c in text)
        words = cleaned.lower().split()
        return [w for w in words if len(w) >= self.min_length]
```

---

5. Principios SOLID Básicos {#5-solid}

5.1 S - Single Responsibility Principle

```
┌─────────────────────────────────────────────────────────┐
│  PRINCIPIO: Una clase debe tener una sola razón para cambiar    │
│                                                         │
│  ❌ MAL: Document que hace todo                         │
│  ──────────────────────────────                         │
│  class Document:                                        │
│      def tokenize(self): ...                            │
│      def save_to_file(self): ...      # Persistencia    │
│      def compute_tfidf(self): ...     # Cálculo ML      │
│      def render_html(self): ...       # Presentación    │
│                                                         │
│  ✅ BIEN: Responsabilidades separadas                   │
│  ─────────────────────────────────────                  │
│  class Document:          # Solo datos del documento    │
│  class Tokenizer:         # Solo tokenización           │
│  class DocumentStorage:   # Solo persistencia           │
│  class TFIDFCalculator:   # Solo cálculos               │
└─────────────────────────────────────────────────────────┘
```

5.2 O - Open/Closed Principle

```
# ✅ Abierto para extensión, cerrado para modificación

class Tokenizer(ABC):
    @abstractmethod
    def tokenize(self, text: str) -> list[str]:
        pass

# Extender sin modificar la clase base
class SpanishTokenizer(Tokenizer):
    """Tokenizer with Spanish stop words."""
```

```
    STOP_WORDS = {"el", "la", "los", "las", "de", "en"}

    def tokenize(self, text: str) -> list[str]:
        words = text.lower().split()
        return [w for w in words if w not in self.STOP_WORDS]
```

5.3 Aplicación en el Proyecto

```
# Cada clase tiene una responsabilidad clara:

class Document:
    """Solo almacena datos de un documento."""
    pass

class Corpus:
    """Solo administra una colección de documentos."""
    pass

class Tokenizer:
    """Solo convierte texto en tokens."""
    pass

class InvertedIndex:
    """Solo indexa documentos para búsqueda."""
    pass

class SearchEngine:
    """Orquesta los demás componentes."""
    pass
```

---

6. Dataclasses {#6-dataclasses}

6.1 Simplificando Clases de Datos

```
from dataclasses import dataclass, field

# ❌ Mucho boilerplate
class DocumentOld:
    def __init__(self, doc_id: int, content: str, title: str = "") -> None:
        self.doc_id = doc_id
        self.content = content
        self.title = title

    def __repr__(self) -> str:
        return f"Document(doc_id={self.doc_id}, content='{self.content[:20]}...',
title='{self.title}')"

    def __eq__(self, other: object) -> bool:
        if not isinstance(other, DocumentOld):
            return NotImplemented
        return self.doc_id == other.doc_id and self.content == other.content

# ✅ Dataclass: automático
@dataclass
class Document:
    doc_id: int
    content: str
    title: str = ""
    tokens: list[str] = field(default_factory=list)
```

```
    # Puedes agregar métodos normalmente
    def tokenize(self) -> list[str]:
        self.tokens = self.content.lower().split()
        return self.tokens
```

6.2 Opciones de Dataclass

```
@dataclass(frozen=True)  # Inmutable (no se puede modificar)
class ImmutableDocument:
    doc_id: int
    content: str

@dataclass(order=True)  # Permite comparar <, >, etc.
class RankedDocument:
    score: float  # Primer campo = criterio de ordenamiento
    doc_id: int
    content: str

# Uso
docs = [RankedDocument(0.8, 1, "doc1"), RankedDocument(0.9, 2, "doc2")]
sorted_docs = sorted(docs, reverse=True)  # Ordenar por score
```

6.3 Cuándo Usar Dataclass

Usa Dataclass cuando...	**Usa Clase normal cuando...**
Principalmente almacena datos	Lógica compleja de validación
init, **repr**, **eq** estándar	Necesitas control total
Quieres código conciso	Properties con setters

⚠️ Errores Comunes y Cómo Evitarlos

Error 1: Olvidar self

```
# ❌ Error: NameError: name 'doc_id' is not defined
class Document:
    def __init__(self, doc_id: int) -> None:
        doc_id = doc_id  # ¡No guarda nada!

# ✅ Correcto
class Document:
    def __init__(self, doc_id: int) -> None:
        self.doc_id = doc_id
```

Error 2: Mutar lista compartida

```
# ❌ Bug: todos los documentos comparten la misma lista
class Document:
    tokens: list[str] = []  # ¡Atributo de clase!

# ✅ Correcto: inicializar en __init__
class Document:
    def __init__(self) -> None:
        self.tokens: list[str] = []  # Atributo de instancia
```

Error 3: eq sin hash

```
# ❌ Si defines __eq__, Python elimina __hash__ por defecto
class Document:
    def __eq__(self, other): ...
    # No se puede usar en sets/dicts

# ✅ Definir ambos
class Document:
    def __eq__(self, other): ...
    def __hash__(self): return hash(self.doc_id)
```

🔧 Ejercicios Prácticos

Ejercicio 2.1: Clase Document Básica

Ver EJERCICIOS.md

Ejercicio 2.2: Métodos Mágicos

Ver EJERCICIOS.md

Ejercicio 2.3: Properties con Validación

Ver EJERCICIOS.md

Ejercicio 2.4: Clase Corpus

Ver EJERCICIOS.md

Ejercicio 2.5: Refactorizar a SOLID

Ver EJERCICIOS.md

📚 Recursos Externos

Recurso	Tipo	Prioridad
Real Python: OOP	Tutorial	🔴 Obligatorio
Dataclasses Documentation	Docs	🟡 Recomendado
SOLID Principles	Tutorial	🟡 Recomendado

🔗 Referencias del Glosario

- Clase
- Instancia
- Método Mágico
- Property
- Composición
- SOLID

Módulo 03 - Álgebra Lineal para ML

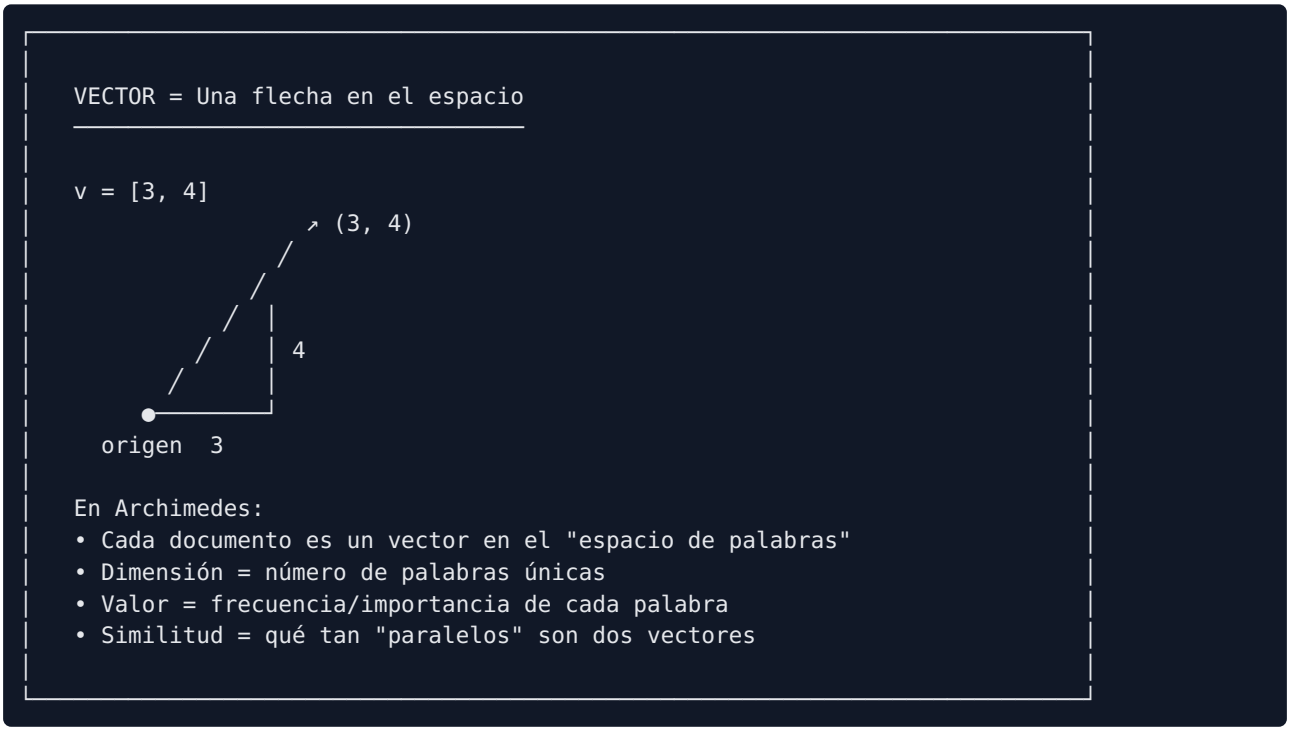
Guía MS in AI Pathway

DUQUEOM · 2025

Módulo 03 - Álgebra Lineal para ML

> 🎯 **Objetivo:** Implementar operaciones vectoriales y matriciales desde cero
> **Fase:** Fundamentos | **Prerrequisito para:** Módulos 04-09

🧠 Analogía: Vectores como Flechas, Matrices como Transformaciones



📋 Contenido

---

1. Vectores como Listas {#1-vectores}

1.1 Representación

```
# Un vector es simplemente una lista de números
Vector = list[float]

# Ejemplos
v1: Vector = [1.0, 2.0, 3.0]      # Vector 3D
v2: Vector = [0.5, 0.3, 0.8, 0.2] # Vector 4D

# En el contexto de TF-IDF:
# Cada posición corresponde a una palabra del vocabulario
# El valor es la importancia de esa palabra en el documento
```

```
vocabulary = ["python", "java", "code", "tutorial"]
doc_vector: Vector = [0.8, 0.0, 0.5, 0.3]
# Significa: mucho "python", nada de "java", algo de "code" y "tutorial"
```

1.2 Acceso y Longitud

```
def get_dimension(v: Vector) -> int:
    """Return the dimension (number of components) of a vector."""
    return len(v)


def get_component(v: Vector, index: int) -> float:
    """Get specific component of vector."""
    if index < 0 or index >= len(v):
        raise IndexError(f"Index {index} out of range for vector of dimension {len(v)}")
    return v[index]
```

---

2. Operaciones Vectoriales {#2-operaciones-vectoriales}

2.1 Suma de Vectores

```
┌─────────────────────────────────────────────────────┐
│  SUMA: Componente a componente                      │
│                                                     │
│  [1, 2, 3] + [4, 5, 6] = [1+4, 2+5, 3+6] = [5, 7, 9]  │
│                                                     │
│  Geométricamente: "poner una flecha al final de otra"  │
└─────────────────────────────────────────────────────┘
```

```
def add_vectors(v1: Vector, v2: Vector) -> Vector:
    """Add two vectors component-wise.

    Args:
        v1: First vector.
        v2: Second vector (must have same dimension as v1).

    Returns:
        New vector with sum of corresponding components.

    Raises:
        ValueError: If vectors have different dimensions.

    Example:
        >>> add_vectors([1, 2, 3], [4, 5, 6])
        [5, 7, 9]
    """
    if len(v1) != len(v2):
        raise ValueError(f"Dimension mismatch: {len(v1)} vs {len(v2)}")

    return [a + b for a, b in zip(v1, v2)]
```

2.2 Resta de Vectores

```
def subtract_vectors(v1: Vector, v2: Vector) -> Vector:
    """Subtract v2 from v1 component-wise.

    Example:
        >>> subtract_vectors([5, 7, 9], [4, 5, 6])
        [1, 2, 3]
```

```
    """
    if len(v1) != len(v2):
        raise ValueError(f"Dimension mismatch: {len(v1)} vs {len(v2)}")

    return [a - b for a, b in zip(v1, v2)]
```

2.3 Multiplicación por Escalar

```
┌─────────────────────────────────────────────────────────┐
│  ESCALAR × VECTOR: Multiplica cada componente           │
│                                                         │
│  3 × [1, 2, 3] = [3×1, 3×2, 3×3] = [3, 6, 9]           │
│                                                         │
│  Geométricamente: "estirar" o "encoger" la flecha       │
│  - Escalar > 1: estira                                  │
│  - 0 < Escalar < 1: encoge                              │
│  - Escalar < 0: invierte dirección                      │
└─────────────────────────────────────────────────────────┘
```

```
def scalar_multiply(scalar: float, v: Vector) -> Vector:
    """Multiply a vector by a scalar.

    Example:
        >>> scalar_multiply(3, [1, 2, 3])
        [3, 6, 9]
        >>> scalar_multiply(0.5, [4, 6])
        [2.0, 3.0]
    """
    return [scalar * component for component in v]
```

---

3. Producto Punto y Norma {#3-producto-punto}

3.1 Producto Punto (Dot Product)

```
┌─────────────────────────────────────────────────────────┐
│  PRODUCTO PUNTO: Suma de productos componente a componente │
│                                                         │
│  [1, 2, 3] · [4, 5, 6] = 1×4 + 2×5 + 3×6 = 4 + 10 + 18 = 32 │
│                                                         │
│  SIGNIFICADO GEOMÉTRICO:                                │
│  v1 · v2 = |v1| × |v2| × cos(θ)                         │
│                                                         │
│  Donde θ es el ángulo entre los vectores.               │
│                                                         │
│  Si cos(θ) = 1 (θ = 0°): vectores paralelos, misma dirección │
│  Si cos(θ) = 0 (θ = 90°): vectores perpendiculares      │
│  Si cos(θ) = -1 (θ = 180°): direcciones opuestas        │
│                                                         │
│  EN ARCHIMEDES: Mide qué tan "similares" son dos documentos │
└─────────────────────────────────────────────────────────┘
```

```
def dot_product(v1: Vector, v2: Vector) -> float:
    """Compute dot product of two vectors.

    Also known as inner product or scalar product.

    Args:
        v1: First vector.
        v2: Second vector (same dimension as v1).
```

```
    Returns:
        Scalar result of dot product.

    Example:
        >>> dot_product([1, 2, 3], [4, 5, 6])
        32
        >>> dot_product([1, 0], [0, 1])  # Perpendicular
        0
    """
    if len(v1) != len(v2):
        raise ValueError(f"Dimension mismatch: {len(v1)} vs {len(v2)}")

    return sum(a * b for a, b in zip(v1, v2))
```

3.2 Norma (Magnitud)

```
┌─────────────────────────────────────────────────────────┐
│  NORMA = Longitud del vector                            │
│                                                         │
│  ||v|| = √(v₁² + v₂² + ... + vₙ²)                       │
│                                                         │
│  Ejemplo: ||[3, 4]|| = √(9 + 16) = √25 = 5              │
│                                                         │
│  Nota: ||v||² = v · v  (producto punto consigo mismo)   │
└─────────────────────────────────────────────────────────┘
```

```
import math


def magnitude(v: Vector) -> float:
    """Compute the magnitude (length/norm) of a vector.

    Also known as Euclidean norm or L2 norm.

    Formula: ||v|| = sqrt(v1² + v2² + ... + vn²)

    Example:
        >>> magnitude([3, 4])
        5.0
        >>> magnitude([1, 0, 0])
        1.0
    """
    return math.sqrt(sum(component ** 2 for component in v))


def magnitude_squared(v: Vector) -> float:
    """Compute squared magnitude (avoids sqrt for comparisons).

    Useful when you only need to compare magnitudes.
    """
    return sum(component ** 2 for component in v)
```

3.3 Normalización (Vector Unitario)

```
def normalize(v: Vector) -> Vector:
    """Return unit vector (magnitude = 1) in same direction.

    Formula: v̂ = v / ||v||

    Example:
```

```
        >>> normalize([3, 4])
        [0.6, 0.8]  # magnitude = 1.0

    Raises:
        ValueError: If vector has zero magnitude.
    """
    mag = magnitude(v)

    if mag == 0:
        raise ValueError("Cannot normalize zero vector")

    return [component / mag for component in v]
```

3.4 Similitud de Coseno (¡Crucial para Archimedes!)

```
┌─────────────────────────────────────────────────────────┐
│  SIMILITUD DE COSENO                                    │
│                                                         │
│                    v1 · v2                              │
│  cos(θ) = ──────────────────────                        │
│            ||v1|| × ||v2||                              │
│                                                         │
│  Resultado:                                             │
│  • 1: Vectores idénticos en dirección (muy similares)   │
│  • 0: Vectores perpendiculares (nada en común)          │
│  • -1: Direcciones opuestas (para TF-IDF, raro)         │
│                                                         │
│  EN ARCHIMEDES: Documentos con palabras similares tendrán│
│  coseno cercano a 1                                     │
└─────────────────────────────────────────────────────────┘
```

```
def cosine_similarity(v1: Vector, v2: Vector) -> float:
    """Compute cosine similarity between two vectors.

    Measures the cosine of the angle between vectors.

    Returns:
        Value between -1 and 1 (usually 0 to 1 for TF-IDF).
        1 = identical direction, 0 = perpendicular.

    Example:
        >>> cosine_similarity([1, 0], [1, 0])
        1.0
        >>> cosine_similarity([1, 0], [0, 1])
        0.0
        >>> cosine_similarity([1, 1], [1, 1])
        1.0
    """
    if len(v1) != len(v2):
        raise ValueError(f"Dimension mismatch: {len(v1)} vs {len(v2)}")

    dot = dot_product(v1, v2)
    mag1 = magnitude(v1)
    mag2 = magnitude(v2)

    # Handle zero vectors
    if mag1 == 0 or mag2 == 0:
        return 0.0

    return dot / (mag1 * mag2)
```

---

4. Matrices como Listas de Listas {#4-matrices}

4.1 Representación

```
# Matriz = lista de filas, cada fila es un vector
Matrix = list[list[float]]

# Ejemplo: matriz 2x3 (2 filas, 3 columnas)
m: Matrix = [
    [1, 2, 3],  # Fila 0
    [4, 5, 6]   # Fila 1
]

# Acceso: m[fila][columna]
# m[0][0] = 1, m[0][2] = 3, m[1][1] = 5
```

4.2 Funciones de Información

```
def get_shape(m: Matrix) -> tuple[int, int]:
    """Return (rows, columns) of matrix.

    Example:
        >>> get_shape([[1, 2, 3], [4, 5, 6]])
        (2, 3)
    """
    if not m:
        return (0, 0)
    return (len(m), len(m[0]))


def get_element(m: Matrix, row: int, col: int) -> float:
    """Get element at (row, col)."""
    return m[row][col]


def get_row(m: Matrix, row: int) -> Vector:
    """Get a row as vector."""
    return m[row].copy()


def get_column(m: Matrix, col: int) -> Vector:
    """Get a column as vector."""
    return [row[col] for row in m]
```

---

5. Operaciones Matriciales {#5-operaciones-matriciales}

5.1 Suma de Matrices

```
def add_matrices(m1: Matrix, m2: Matrix) -> Matrix:
    """Add two matrices element-wise.

    Matrices must have same dimensions.

    Example:
        >>> add_matrices([[1, 2], [3, 4]], [[5, 6], [7, 8]])
        [[6, 8], [10, 12]]
    """
    rows1, cols1 = get_shape(m1)
    rows2, cols2 = get_shape(m2)
```

```
    if (rows1, cols1) != (rows2, cols2):
        raise ValueError(f"Shape mismatch: {(rows1, cols1)} vs {(rows2, cols2)}")

    return [
        [m1[i][j] + m2[i][j] for j in range(cols1)]
        for i in range(rows1)
    ]
```

5.2 Multiplicación por Escalar

```
def scalar_multiply_matrix(scalar: float, m: Matrix) -> Matrix:
    """Multiply matrix by scalar.

    Example:
        >>> scalar_multiply_matrix(2, [[1, 2], [3, 4]])
        [[2, 4], [6, 8]]
    """
    return [
        [scalar * element for element in row]
        for row in m
    ]
```

5.3 Transpuesta

```
┌─────────────────────────────────────────────────────┐
│  TRANSPUESTA: Intercambiar filas y columnas         │
│                                                     │
│  Original (2x3):        Transpuesta (3x2):          │
│  [1, 2, 3]              [1, 4]                      │
│  [4, 5, 6]              [2, 5]                      │
│                         [3, 6]                      │
│                                                     │
│  Fórmula: T[i][j] = M[j][i]                         │
└─────────────────────────────────────────────────────┘
```

```
def transpose(m: Matrix) -> Matrix:
    """Transpose a matrix (swap rows and columns).

    Example:
        >>> transpose([[1, 2, 3], [4, 5, 6]])
        [[1, 4], [2, 5], [3, 6]]
    """
    if not m:
        return []

    rows, cols = get_shape(m)

    return [
        [m[i][j] for i in range(rows)]
        for j in range(cols)
    ]
```

5.4 Producto Matriz-Vector

```
┌─────────────────────────────────────────────────────┐
│  MATRIZ × VECTOR                                    │
│                                                     │
│  [1, 2, 3]     [1]     [1×1 + 2×2 + 3×3]     [14]   │
│  [4, 5, 6]  ×  [2]  =  [4×1 + 5×2 + 6×3]  =  [32]   │
│                [3]                                  │
```

```
┌                                                                    ┐
│  Cada elemento del resultado = producto punto de una fila          │
│  con el vector                                                     │
└                                                                    ┘
```

```
def matrix_vector_multiply(m: Matrix, v: Vector) -> Vector:
    """Multiply matrix by vector.

    Matrix must have columns = len(v).
    Result has length = rows of matrix.

    Example:
        >>> matrix_vector_multiply([[1, 2, 3], [4, 5, 6]], [1, 2, 3])
        [14, 32]
    """
    rows, cols = get_shape(m)

    if cols != len(v):
        raise ValueError(f"Dimension mismatch: matrix has {cols} cols, vector has {len(v)}
elements")

    return [dot_product(row, v) for row in m]
```

5.5 Producto Matriz-Matriz (Opcional pero útil)

```
def matrix_multiply(m1: Matrix, m2: Matrix) -> Matrix:
    """Multiply two matrices.

    m1 must have cols = m2 rows.
    Result shape: (m1 rows, m2 cols).

    Example:
        >>> matrix_multiply([[1, 2], [3, 4]], [[5, 6], [7, 8]])
        [[19, 22], [43, 50]]
    """
    rows1, cols1 = get_shape(m1)
    rows2, cols2 = get_shape(m2)

    if cols1 != rows2:
        raise ValueError(f"Cannot multiply: m1 has {cols1} cols, m2 has {rows2} rows")

    # Result[i][j] = dot product of m1 row i with m2 column j
    result = []
    for i in range(rows1):
        row = []
        for j in range(cols2):
            val = sum(m1[i][k] * m2[k][j] for k in range(cols1))
            row.append(val)
        result.append(row)

    return result
```

⚠️ Errores Comunes

Error 1: Modificar vectores originales

```
# ❌ Modifica el vector original
def normalize_bad(v: Vector) -> Vector:
    mag = magnitude(v)
    for i in range(len(v)):
```

```
        v[i] /= mag  # Modifica v!
    return v

# ✅ Crear nuevo vector
def normalize_good(v: Vector) -> Vector:
    mag = magnitude(v)
    return [x / mag for x in v]
```

Error 2: División por cero en normalización

```
# ❌ Falla con vector cero
def normalize_bad(v):
    mag = magnitude(v)
    return [x / mag for x in v]  # ZeroDivisionError!

# ✅ Manejar caso especial
def normalize_good(v):
    mag = magnitude(v)
    if mag == 0:
        raise ValueError("Cannot normalize zero vector")
    return [x / mag for x in v]
```

Error 3: Comparar floats con ==

```
# ❌ Puede fallar por precisión de punto flotante
if magnitude(v) == 1.0:
    print("Unit vector")

# ✅ Usar tolerancia
if abs(magnitude(v) - 1.0) < 1e-9:
    print("Unit vector")
```

🔧 Ejercicios Prácticos

Ejercicio 10.1: Operaciones Vectoriales

Ver EJERCICIOS.md

Ejercicio 10.2: Producto Punto y Norma

Ver EJERCICIOS.md

Ejercicio 10.3: Similitud de Coseno

Ver EJERCICIOS.md

📚 Recursos Externos

Recurso	Tipo	Prioridad
3Blue1Brown: Linear Algebra	Video	🔴 Obligatorio
Mathematics for ML: Linear Algebra	Curso	🔴 Obligatorio
Khan Academy Linear Algebra	Curso	🟡 Recomendado

🔗 Referencias del Glosario

- Vector
- Matriz
- Producto Punto
- Norma
- Similitud de Coseno

🧭 Navegación

← Anterior	Índice	Siguiente →
09_BINARY_SEARCH	00_INDICE	11_TFIDF_COSENO

Módulo 04 - Fundamentos de Probabilidad ⭐

Guía MS in AI Pathway

DUQUEOM · 2025

Módulo 04 - Fundamentos de Probabilidad para IA

> 🎯 **Objetivo:** Dominar los conceptos probabilísticos esenciales para ML/IA
> ⭐ **PATHWAY LÍNEA 2:** Probability Fundamentals for Data Science and AI

🧠 Analogía: La Probabilidad como Lenguaje de la Incertidumbre

```
┌─────────────────────────────────────────────────────────────┐
│                                                             │
│   LA IA VIVE EN LA INCERTIDUMBRE                            │
│   ──────────────────────────────                            │
│                                                             │
│   Determinístico (Algoritmos clásicos):                     │
│   if x > 5: return "grande"  → SIEMPRE la misma respuesta   │
│                                                             │
│   Probabilístico (Machine Learning):                        │
│   P(spam | email) = 0.87     → "Probablemente spam, 87% seguro"
│                                                             │
│   ¿POR QUÉ PROBABILIDAD?                                    │
│   • Datos ruidosos e incompletos                            │
│   • Predicciones sobre el futuro                            │
│   • Cuantificar confianza en decisiones                     │
│   • Generalizar de muestras a poblaciones                   │
│                                                             │
└─────────────────────────────────────────────────────────────┘
```

📋 Contenido

---

1. Fundamentos de Probabilidad {#1-fundamentos}

1.1 Espacio Muestral y Eventos

```python
from typing import Set, Dict
import math

# Espacio muestral: todos los resultados posibles
# Evento: subconjunto del espacio muestral

def probability_basic(favorable: int, total: int) -> float:
    """Basic probability: P(A) = favorable outcomes / total outcomes.

    Example:
        >>> probability_basic(1, 6)  # Sacar un 6 en un dado
        0.16666666666666666
    """
    if total == 0:
        raise ValueError("Total outcomes cannot be zero")
    return favorable / total
```

```
def complement_probability(p_a: float) -> float:
    """P(not A) = 1 - P(A).

    Example:
        >>> complement_probability(0.3)  # P(no llueve) si P(llueve) = 0.3
        0.7
    """
    return 1.0 - p_a
```

1.2 Axiomas de Kolmogorov

```
AXIOMAS DE PROBABILIDAD:
─────────────────────────
1. P(A) ≥ 0          (No negativas)
2. P(Ω) = 1          (Espacio muestral tiene prob. 1)
3. P(A ∪ B) = P(A) + P(B)   si A ∩ B = ∅  (Aditividad)

PROPIEDADES DERIVADAS:
─────────────────────────
• P(∅) = 0
• P(A') = 1 - P(A)
• P(A ∪ B) = P(A) + P(B) - P(A ∩ B)
• Si A ⊆ B, entonces P(A) ≤ P(B)
```

1.3 Operaciones con Eventos

```
def union_probability(p_a: float, p_b: float, p_intersection: float) -> float:
    """P(A ∪ B) = P(A) + P(B) - P(A ∩ B).

    Inclusion-exclusion principle.

    Example:
        >>> # P(rey o corazón) en baraja
        >>> union_probability(4/52, 13/52, 1/52)
        0.3076923076923077
    """
    return p_a + p_b - p_intersection


def intersection_independent(p_a: float, p_b: float) -> float:
    """P(A ∩ B) = P(A) × P(B) for independent events.

    Example:
        >>> # Dos monedas, ambas cara
        >>> intersection_independent(0.5, 0.5)
        0.25
    """
    return p_a * p_b
```

---

2. Probabilidad Condicional y Bayes {#2-bayes}

2.1 Probabilidad Condicional

```
def conditional_probability(p_a_and_b: float, p_b: float) -> float:
    """P(A|B) = P(A ∩ B) / P(B).

    Probability of A given B has occurred.

    Example:
```

```
        >>> # P(llueve | nublado)
        >>> conditional_probability(0.3, 0.4)
        0.75
    """
    if p_b == 0:
        raise ValueError("P(B) cannot be zero")
    return p_a_and_b / p_b
```

2.2 Teorema de Bayes ⭐ FUNDAMENTAL PARA ML

```
┌──────────────────────────────────────────────────────────────────┐
│                                                                  │
│   TEOREMA DE BAYES                                               │
│   ────────────────                                               │
│                                                                  │
│                     P(B|A) × P(A)                                │
│   P(A|B) = ──────────────────────────                            │
│                      P(B)                                        │
│                                                                  │
│   Donde:                                                         │
│   • P(A|B) = POSTERIOR (lo que queremos saber)                   │
│   • P(B|A) = LIKELIHOOD (evidencia dado la hipótesis)            │
│   • P(A)   = PRIOR (creencia inicial)                            │
│   • P(B)   = EVIDENCE (normalizador)                             │
│                                                                  │
│   EJEMPLO SPAM:                                                  │
│   P(spam | "gratis") = P("gratis"|spam) × P(spam) / P("gratis")  │
│                                                                  │
└──────────────────────────────────────────────────────────────────┘
```

```
def bayes_theorem(
    p_b_given_a: float,  # Likelihood
    p_a: float,          # Prior
    p_b: float           # Evidence
) -> float:
    """Bayes' Theorem: P(A|B) = P(B|A) × P(A) / P(B).

    The foundation of probabilistic machine learning.

    Example:
        >>> # Test médico: P(enfermo | test positivo)
        >>> # P(test+|enfermo) = 0.99, P(enfermo) = 0.01, P(test+) = 0.02
        >>> bayes_theorem(0.99, 0.01, 0.02)
        0.495
    """
    return (p_b_given_a * p_a) / p_b


def bayes_with_total_probability(
    p_b_given_a: float,
    p_a: float,
    p_b_given_not_a: float
) -> float:
    """Bayes with P(B) calculated via total probability.

    P(B) = P(B|A)P(A) + P(B|¬A)P(¬A)

    Example:
        >>> # Spam classifier
        >>> # P("free"|spam)=0.7, P(spam)=0.3, P("free"|not spam)=0.1
        >>> bayes_with_total_probability(0.7, 0.3, 0.1)
        0.75
```

```
    """
    p_not_a = 1 - p_a
    p_b = p_b_given_a * p_a + p_b_given_not_a * p_not_a
    return (p_b_given_a * p_a) / p_b
```

2.3 Aplicación: Clasificador Naive Bayes

```
from collections import defaultdict
from typing import List, Tuple

class NaiveBayesClassifier:
    """Simple Naive Bayes for text classification.

    Assumes features are conditionally independent given class.

    P(class|features) ∝ P(class) × ∏ P(feature|class)
    """

    def __init__(self) -> None:
        self.class_counts: Dict[str, int] = defaultdict(int)
        self.feature_counts: Dict[str, Dict[str, int]] = defaultdict(lambda:
defaultdict(int))
        self.total_samples: int = 0

    def fit(self, X: List[List[str]], y: List[str]) -> None:
        """Train the classifier.

        Args:
            X: List of feature lists (e.g., words in documents)
            y: List of class labels
        """
        for features, label in zip(X, y):
            self.class_counts[label] += 1
            self.total_samples += 1
            for feature in features:
                self.feature_counts[label][feature] += 1

    def predict(self, features: List[str]) -> str:
        """Predict class for given features.

        Returns class with highest posterior probability.
        """
        best_class = None
        best_score = float('-inf')

        for cls in self.class_counts:
            # Log probability to avoid underflow
            score = math.log(self.class_counts[cls] / self.total_samples)

            total_features_in_class = sum(self.feature_counts[cls].values())
            vocab_size = len(set(
                f for counts in self.feature_counts.values()
                for f in counts
            ))

            for feature in features:
                # Laplace smoothing
                count = self.feature_counts[cls].get(feature, 0) + 1
                prob = count / (total_features_in_class + vocab_size)
                score += math.log(prob)

            if score > best_score:
```

```
                best_score = score
                best_class = cls

        return best_class
```

---

3. Variables Aleatorias {#3-variables-aleatorias}

3.1 Discretas vs Continuas

```
VARIABLES ALEATORIAS:
─────────────────────

DISCRETAS: Valores contables
• Número de emails spam
• Cara o cruz
• Clasificación (0, 1, 2, ...)

CONTINUAS: Valores en un rango
• Temperatura
• Altura
• Probabilidad predicha

FUNCIÓN DE PROBABILIDAD:
• Discreta: PMF (Probability Mass Function)
  P(X = x)

• Continua: PDF (Probability Density Function)
  P(a ≤ X ≤ b) = ∫[a,b] f(x)dx
```

3.2 Función de Distribución Acumulativa (CDF)

```
def cdf_from_pmf(pmf: Dict[int, float], x: int) -> float:
    """CDF: F(x) = P(X ≤ x) = Σ P(X = k) for k ≤ x.

    Example:
        >>> pmf = {1: 1/6, 2: 1/6, 3: 1/6, 4: 1/6, 5: 1/6, 6: 1/6}
        >>> cdf_from_pmf(pmf, 3)
        0.5
    """
    return sum(prob for val, prob in pmf.items() if val <= x)
```

---

4. Distribuciones de Probabilidad {#4-distribuciones}

4.1 Distribución Bernoulli

```
def bernoulli_pmf(k: int, p: float) -> float:
    """Bernoulli: single trial with success probability p.

    P(X = k) = p^k × (1-p)^(1-k) for k ∈ {0, 1}

    Example:
        >>> bernoulli_pmf(1, 0.7)  # P(success) with p=0.7
        0.7
    """
    if k == 1:
        return p
    elif k == 0:
        return 1 - p
```

```
    else:
        return 0.0
```

4.2 Distribución Binomial

```
def factorial(n: int) -> int:
    """Calculate n! iteratively."""
    result = 1
    for i in range(2, n + 1):
        result *= i
    return result


def binomial_coefficient(n: int, k: int) -> int:
    """C(n, k) = n! / (k! × (n-k)!)."""
    return factorial(n) // (factorial(k) * factorial(n - k))


def binomial_pmf(k: int, n: int, p: float) -> float:
    """Binomial: k successes in n independent trials.

    P(X = k) = C(n,k) × p^k × (1-p)^(n-k)

    Example:
        >>> # P(3 caras en 5 lanzamientos)
        >>> binomial_pmf(3, 5, 0.5)
        0.3125
    """
    return binomial_coefficient(n, k) * (p ** k) * ((1 - p) ** (n - k))
```

4.3 Distribución Normal (Gaussiana) ⭐ FUNDAMENTAL

```
┌─────────────────────────────────────────────────────────────────┐
│                                                                 │
│   DISTRIBUCIÓN NORMAL                                           │
│   ═══════════════════                                           │
│                                                                 │
│                      1              (x - μ)²                    │
│   f(x) = ──────────────────── × exp(- ──────────)               │
│          σ × √(2π)                    2σ²                       │
│                                                                 │
│   Parámetros:                                                   │
│   • μ (mu) = media (centro de la campana)                       │
│   • σ (sigma) = desviación estándar (ancho)                     │
│                                                                 │
│               .----.                                           │
│            .'      '.            68% dentro de 1σ             │
│           /    μ     \           95% dentro de 2σ             │
│         _/             \_        99.7% dentro de 3σ            │
│   ─────/─────────────────\─────                                 │
│        μ-2σ  μ-σ   μ   μ+σ  μ+2σ                               │
│                                                                 │
│   ¿POR QUÉ ES TAN IMPORTANTE?                                   │
│   • Teorema del Límite Central                                  │
│   • Muchos fenómenos naturales                                  │
│   • Base de modelos lineales                                    │
│                                                                 │
└─────────────────────────────────────────────────────────────────┘
```

```
def normal_pdf(x: float, mu: float = 0.0, sigma: float = 1.0) -> float:
    """Probability density function of normal distribution.
```

```
    Example:
        >>> normal_pdf(0, 0, 1)  # Standard normal at mean
        0.3989422804014327
    """
    coefficient = 1 / (sigma * math.sqrt(2 * math.pi))
    exponent = -((x - mu) ** 2) / (2 * sigma ** 2)
    return coefficient * math.exp(exponent)


def standard_normal_cdf_approx(x: float) -> float:
    """Approximation of standard normal CDF using error function.

    Uses the relationship: Φ(x) = 0.5 × (1 + erf(x/√2))
    """
    return 0.5 * (1 + math.erf(x / math.sqrt(2)))


def z_score(x: float, mu: float, sigma: float) -> float:
    """Standardize a value: z = (x - μ) / σ.

    Converts any normal distribution to standard normal.

    Example:
        >>> z_score(85, 70, 10)  # Score of 85 with mean 70, std 10
        1.5
    """
    return (x - mu) / sigma
```

4.4 Otras Distribuciones Importantes

```
def poisson_pmf(k: int, lam: float) -> float:
    """Poisson: events in fixed interval.

    P(X = k) = (λ^k × e^(-λ)) / k!

    Used for: emails per hour, arrivals per minute.

    Example:
        >>> poisson_pmf(3, 2.5)  # 3 events when average is 2.5
        0.21376...
    """
    return (lam ** k * math.exp(-lam)) / factorial(k)


def exponential_pdf(x: float, lam: float) -> float:
    """Exponential: time between Poisson events.

    f(x) = λ × e^(-λx) for x ≥ 0

    Used for: time until next event.
    """
    if x < 0:
        return 0.0
    return lam * math.exp(-lam * x)
```

---

5. Esperanza, Varianza y Momentos {#5-momentos}

5.1 Valor Esperado (Media)

```
def expected_value_discrete(pmf: Dict[float, float]) -> float:
    """E[X] = Σ x × P(X = x).

    The "center of mass" of the distribution.

    Example:
        >>> pmf = {1: 1/6, 2: 1/6, 3: 1/6, 4: 1/6, 5: 1/6, 6: 1/6}
        >>> expected_value_discrete(pmf)
        3.5
    """
    return sum(x * prob for x, prob in pmf.items())


def expected_value_sample(data: List[float]) -> float:
    """Sample mean as estimate of E[X].

    x̄ = (1/n) × Σ xᵢ
    """
    return sum(data) / len(data)
```

5.2 Varianza y Desviación Estándar

```
def variance_discrete(pmf: Dict[float, float]) -> float:
    """Var(X) = E[(X - μ)²] = E[X²] - (E[X])².

    Measures spread around the mean.
    """
    mu = expected_value_discrete(pmf)
    return sum((x - mu) ** 2 * prob for x, prob in pmf.items())


def variance_sample(data: List[float]) -> float:
    """Sample variance (unbiased estimator).

    s² = (1/(n-1)) × Σ (xᵢ - x̄)²
    """
    n = len(data)
    mean = expected_value_sample(data)
    return sum((x - mean) ** 2 for x in data) / (n - 1)


def std_dev_sample(data: List[float]) -> float:
    """Sample standard deviation."""
    return math.sqrt(variance_sample(data))
```

5.3 Covarianza y Correlación

```
def covariance(x: List[float], y: List[float]) -> float:
    """Cov(X, Y) = E[(X - μₓ)(Y - μᵧ)].

    Measures linear relationship between variables.
    • Cov > 0: positive relationship
    • Cov < 0: negative relationship
    • Cov = 0: no linear relationship (not necessarily independent!)
    """
    n = len(x)
    mean_x = sum(x) / n
```

```
    mean_y = sum(y) / n
    return sum((xi - mean_x) * (yi - mean_y) for xi, yi in zip(x, y)) / (n - 1)


def correlation(x: List[float], y: List[float]) -> float:
    """Pearson correlation: ρ = Cov(X,Y) / (σₓ × σᵧ).

    Normalized to [-1, 1].
    • ρ = 1: perfect positive linear relationship
    • ρ = -1: perfect negative linear relationship
    • ρ = 0: no linear relationship

    IMPORTANT for ML: Correlation ≠ Causation!
    """
    cov = covariance(x, y)
    std_x = std_dev_sample(x)
    std_y = std_dev_sample(y)

    if std_x == 0 or std_y == 0:
        return 0.0

    return cov / (std_x * std_y)
```

⚠️ Conceptos Clave para ML

Independence vs Conditional Independence

```
INDEPENDENCIA:
P(A ∩ B) = P(A) × P(B)
P(A|B) = P(A)  (conocer B no cambia A)

INDEPENDENCIA CONDICIONAL (crucial para Naive Bayes):
P(A ∩ B | C) = P(A|C) × P(B|C)

Aunque A y B no sean independientes, pueden serlo dado C.
Naive Bayes ASUME que features son independientes dado la clase.
```

Law of Large Numbers

```
A medida que n → ∞:
• Sample mean → True mean
• Sample variance → True variance

Justifica usar estadísticas muestrales como estimadores.
```

Central Limit Theorem ⭐

```
La suma/promedio de muchas variables aleatorias independientes
tiende a una distribución NORMAL, sin importar la distribución original.

IMPLICACIÓN PARA ML:
• Muchos errores se distribuyen normalmente
• Justifica asumir normalidad en muchos modelos
• Base teórica de muchos métodos estadísticos
```

🔧 Ejercicios Prácticos

Ejercicio 19.1: Bayes para Diagnóstico

Un test tiene 99% sensibilidad, 95% especificidad. La enfermedad afecta al 1% de la población. ¿Cuál es P(enfermo | test+)?

Ejercicio 19.2: Distribución Binomial

Si 30% de emails son spam, ¿cuál es la probabilidad de recibir exactamente 4 spam en 10 emails?

Ejercicio 19.3: Naive Bayes

Implementar clasificador de sentimiento usando el código de ejemplo.

📚 Recursos Externos

Recurso	Tipo	Prioridad
Probability for Data Science	Curso	🔴 Obligatorio
3Blue1Brown: Bayes	Video	🔴 Obligatorio
Khan Academy: Statistics	Curso	🟡 Recomendado

🔗 Referencias del Glosario

- Probabilidad Condicional
- Teorema de Bayes
- Distribución Normal
- Esperanza Matemática
- Varianza

🧭 Navegación

← Anterior	Índice	Siguiente →
18_HEAPS	00_INDICE	20_ESTADISTICA_INFERENCIAL

Módulo 05 - Estadística Inferencial

⭐

Guía MS in AI Pathway

DUQUEOM · 2025

Módulo 05 - Estadística Inferencial para IA

> 🎯 **Objetivo:** Dominar estimación, pruebas de hipótesis e inferencia
> ⭐ **PATHWAY LÍNEA 2:** Statistical Estimation for Data Science and AI

🧠 Analogía: Inferir la Población desde la Muestra

```
┌─────────────────────────────────────────────────────────────────┐
│                                                                 │
│   INFERENCIA ESTADÍSTICA                                        │
│   ──────────────────────                                        │
│                                                                 │
│   POBLACIÓN (desconocida) ──────────────────────────────────    │
│   • Parámetros verdaderos: μ, σ², θ                             │
│   • Imposible medir todos los individuos                        │
│                                                                 │
│              ↓ Muestreo                                         │
│                                                                 │
│   MUESTRA (observada) ──────────────────────────────────────    │
│   • Estadísticos: x̄, s², θ̂                                      │
│   • n observaciones                                             │
│                                                                 │
│              ↓ Inferencia                                       │
│                                                                 │
│   ESTIMACIÓN ───────────────────────────────────────────────    │
│   • Punto: θ̂ ≈ θ                                                │
│   • Intervalo: [θ̂ - error, θ̂ + error] contiene θ con 95% confianza │
│                                                                 │
│   APLICACIÓN EN ML:                                             │
│   • Train set = muestra                                         │
│   • Performance en test = estimación de performance real        │
│   • Cross-validation = reducir varianza de la estimación        │
│                                                                 │
└─────────────────────────────────────────────────────────────────┘
```

📋 Contenido

---

1. Estimación Puntual {#1-estimacion-puntual}

1.1 Propiedades de Buenos Estimadores

```
PROPIEDADES DESEABLES:
──────────────────────

1. INSESGADO (Unbiased):
   E[θ̂] = θ
   El estimador acierta "en promedio"
```

```
2. CONSISTENTE:
   θ̂ → θ cuando n → ∞
   Mejora con más datos

3. EFICIENTE:
   Mínima varianza entre estimadores insesgados
   Menor incertidumbre

SESGO-VARIANZA TRADE-OFF (crucial para ML):
• Sesgo alto → underfitting (modelo muy simple)
• Varianza alta → overfitting (modelo muy complejo)
• Objetivo: minimizar error total = sesgo² + varianza
```

1.2 Estimadores Comunes

```python
from typing import List
import math

def sample_mean(data: List[float]) -> float:
    """Unbiased estimator of population mean.

    E[x̄] = μ
    """
    return sum(data) / len(data)


def sample_variance_unbiased(data: List[float]) -> float:
    """Unbiased estimator of population variance.

    Uses n-1 (Bessel's correction) for unbiasedness.
    E[s²] = σ²
    """
    n = len(data)
    mean = sample_mean(data)
    return sum((x - mean) ** 2 for x in data) / (n - 1)


def sample_variance_mle(data: List[float]) -> float:
    """MLE estimator of variance (biased but consistent).

    Uses n instead of n-1.
    """
    n = len(data)
    mean = sample_mean(data)
    return sum((x - mean) ** 2 for x in data) / n


def standard_error(data: List[float]) -> float:
    """Standard error of the mean: SE = s / √n.

    Measures uncertainty in our estimate of the mean.
    """
    return math.sqrt(sample_variance_unbiased(data) / len(data))
```

---

2. Maximum Likelihood Estimation (MLE) {#2-mle}

2.1 Concepto

```
┌─────────────────────────────────────────────────────────────────────────┐
│                                                                         │
```

```
┌──────────────────────────────────────────────────────────────────────────┐
│   MAXIMUM LIKELIHOOD ESTIMATION                                          │
│   ─────────────────────────────                                          │
│                                                                          │
│   Pregunta: ¿Qué parámetros θ hacen MÁS PROBABLE los datos observados?   │
│                                                                          │
│   Likelihood: L(θ|data) = P(data|θ)                                      │
│                                                                          │
│   MLE: θ̂MLE = argmax L(θ|data)                                           │
│                    θ                                                     │
│                                                                          │
│   Práctica: maximizar log-likelihood (más estable numéricamente):        │
│   θ̂MLE = argmax log L(θ|data)                                            │
│                    θ                                                     │
│                                                                          │
│   EJEMPLO - Moneda sesgada:                                              │
│   Datos: 7 caras en 10 lanzamientos                                      │
│   L(p) = C(10,7) × p^7 × (1-p)^3                                         │
│   MLE: p̂ = 7/10 = 0.7                                                    │
│                                                                          │
└──────────────────────────────────────────────────────────────────────────┘
```

2.2 MLE para Distribuciones Comunes

```python
def mle_bernoulli(successes: int, trials: int) -> float:
    """MLE for Bernoulli parameter p.

    p̂_MLE = number of successes / number of trials

    Example:
        >>> mle_bernoulli(7, 10)
        0.7
    """
    return successes / trials


def mle_normal_mean(data: List[float]) -> float:
    """MLE for normal distribution mean.

    μ̂MLE = sample mean
    """
    return sample_mean(data)


def mle_normal_variance(data: List[float]) -> float:
    """MLE for normal distribution variance.

    Note: This is BIASED (uses n, not n-1).
    σ²_MLE = (1/n) Σ(xᵢ - x̄)²
    """
    return sample_variance_mle(data)


def mle_poisson(data: List[int]) -> float:
    """MLE for Poisson rate parameter λ.

    λ̂MLE = sample mean
    """
    return sum(data) / len(data)
```

2.3 MLE con Gradient Descent (Logistic Regression)

```
def sigmoid(z: float) -> float:
    """Logistic sigmoid function."""
    if z < -500:  # Prevent overflow
        return 0.0
    elif z > 500:
        return 1.0
    return 1.0 / (1.0 + math.exp(-z))


def log_likelihood_logistic(
    X: List[List[float]],
    y: List[int],
    weights: List[float]
) -> float:
    """Log-likelihood for logistic regression.

  ℓ(w) = Σ [yᵢ log(σ(wᵀxᵢ)) + (1-yᵢ) log(1-σ(wᵀxᵢ))]
    """
    ll = 0.0
    for xi, yi in zip(X, y):
        z = sum(w * x for w, x in zip(weights, xi))
        p = sigmoid(z)
        # Avoid log(0)
        p = max(min(p, 1 - 1e-15), 1e-15)
        ll += yi * math.log(p) + (1 - yi) * math.log(1 - p)
    return ll


def gradient_log_likelihood(
    X: List[List[float]],
    y: List[int],
    weights: List[float]
) -> List[float]:
    """Gradient of log-likelihood for logistic regression.

    ∂ℓ/∂wⱼ = Σ (yᵢ - σ(wᵀxᵢ)) × xᵢⱼ
    """
    n_features = len(weights)
    gradient = [0.0] * n_features

    for xi, yi in zip(X, y):
        z = sum(w * x for w, x in zip(weights, xi))
        p = sigmoid(z)
        error = yi - p
        for j in range(n_features):
            gradient[j] += error * xi[j]

    return gradient
```

---

3. Maximum A Posteriori (MAP) {#3-map}

3.1 Concepto: MLE + Prior

```
┌─────────────────────────────────────────────────────────────────────────┐
│                                                                         │
│   MAXIMUM A POSTERIORI (MAP)                                            │
│   ──────────────────────────                                            │
│                                                                         │
```

```
│   MLE: Solo usa datos                                                        │
│   θ̂MLE = argmax P(data|θ)                                                   │
│                                                                              │
│   MAP: Incorpora conocimiento previo (prior)                                 │
│   θ̂MAP = argmax P(θ|data) = argmax P(data|θ) × P(θ)                         │
│                                                                              │
│   Usando Bayes:                                                              │
│   P(θ|data) ∝ P(data|θ) × P(θ)                                               │
│   posterior ∝ likelihood × prior                                             │
│                                                                              │
│   RELACIÓN CON REGULARIZACIÓN:                                               │
│   • Prior Gaussiano → L2 regularization (Ridge)                              │
│   • Prior Laplaciano → L1 regularization (Lasso)                             │
│                                                                              │
│   ¿CUÁNDO USAR MAP vs MLE?                                                   │
│   • Datos abundantes: MLE ≈ MAP (prior se vuelve irrelevante)                │
│   • Datos escasos: MAP más estable (prior regulariza)                        │
│   • Conocimiento previo: MAP permite incorporarlo                            │
│                                                                              │
└──────────────────────────────────────────────────────────────────────────────┘
```

3.2 Ejemplo: MAP con Prior Gaussiano

```python
def map_with_gaussian_prior(
    data: List[float],
    prior_mean: float,
    prior_variance: float,
    likelihood_variance: float
) -> float:
    """MAP estimate for normal mean with Gaussian prior.

    Conjugate prior: Normal prior + Normal likelihood = Normal posterior

    θ̂MAP = (n/σ² × x̄ + 1/τ² × μ₀) / (n/σ² + 1/τ²)

    where:
    - x̄: sample mean
    - n: sample size
    - σ²: likelihood variance
    - μ₀: prior mean
    - τ²: prior variance
    """
    n = len(data)
    sample_mean_val = sample_mean(data)

    precision_likelihood = n / likelihood_variance
    precision_prior = 1 / prior_variance

    numerator = precision_likelihood * sample_mean_val + precision_prior * prior_mean
    denominator = precision_likelihood + precision_prior

    return numerator / denominator
```

---

4. Intervalos de Confianza {#4-intervalos}

4.1 Concepto

```
INTERVALO DE CONFIANZA:
──────────────────────
```

```
Un intervalo [a, b] tal que:
P(a ≤ θ ≤ b) = 1 - α

Para 95% de confianza: α = 0.05

INTERPRETACIÓN CORRECTA:
Si repitiéramos el experimento muchas veces,
95% de los intervalos construidos contendrían θ.

INTERPRETACIÓN INCORRECTA:
"Hay 95% de probabilidad de que θ esté en [a,b]"
(θ es fijo, no aleatorio en estadística frecuentista)
```

4.2 Intervalo para la Media

```
def confidence_interval_mean(
    data: List[float],
    confidence: float = 0.95
) -> tuple[float, float]:
    """Confidence interval for population mean.

    Assumes large sample (n > 30) using Normal approximation.
    For small samples, use t-distribution.

    CI = x̄ ± z* × (s / √n)

    Example:
        >>> data = [23, 25, 27, 29, 31]
        >>> confidence_interval_mean(data, 0.95)
        (23.5..., 30.5...)
    """
    n = len(data)
    mean = sample_mean(data)
    se = standard_error(data)

    # z* values for common confidence levels
    z_values = {
        0.90: 1.645,
        0.95: 1.96,
        0.99: 2.576
    }
    z = z_values.get(confidence, 1.96)

    margin = z * se
    return (mean - margin, mean + margin)


def confidence_interval_proportion(
    successes: int,
    trials: int,
    confidence: float = 0.95
) -> tuple[float, float]:
    """Confidence interval for population proportion.

    Uses normal approximation (valid when np > 5 and n(1-p) > 5).

    CI = p̂ ± z* × √(p̂(1-p̂)/n)
    """
    p_hat = successes / trials
    se = math.sqrt(p_hat * (1 - p_hat) / trials)

    z_values = {0.90: 1.645, 0.95: 1.96, 0.99: 2.576}
```

```
    z = z_values.get(confidence, 1.96)

    margin = z * se
    return (p_hat - margin, p_hat + margin)
```

---

5. Pruebas de Hipótesis {#5-hipotesis}

5.1 Framework

```
ESTRUCTURA DE UNA PRUEBA:
─────────────────────────

1. HIPÓTESIS NULA (H₀): Lo que asumimos es verdad
   "No hay efecto" / "No hay diferencia"

2. HIPÓTESIS ALTERNATIVA (H₁): Lo que queremos probar
   "Hay efecto" / "Hay diferencia"

3. ESTADÍSTICO DE PRUEBA: Resume los datos

4. P-VALUE: P(observar datos tan extremos | H₀ es verdad)
   p < α → Rechazar H₀

5. DECISIÓN:
   • p < 0.05 → "Estadísticamente significativo"
   • p ≥ 0.05 → "No hay evidencia suficiente"

TIPOS DE ERROR:
• Tipo I (α): Rechazar H₀ cuando es verdadera (falso positivo)
• Tipo II (β): No rechazar H₀ cuando es falsa (falso negativo)
• Power = 1 - β: Probabilidad de detectar efecto real
```

5.2 Z-Test para la Media

```
def z_test_one_sample(
    data: List[float],
    population_mean: float,
    population_std: float,
    alternative: str = "two-sided"
) -> tuple[float, float]:
    """One-sample Z-test for population mean.

    H₀: μ = μ₀
    H₁: μ ≠ μ₀ (two-sided) / μ > μ₀ (greater) / μ < μ₀ (less)

    Returns:
        z_statistic, p_value
    """
    n = len(data)
    x_bar = sample_mean(data)

    z = (x_bar - population_mean) / (population_std / math.sqrt(n))

    # Calculate p-value using standard normal CDF approximation
    if alternative == "two-sided":
        p_value = 2 * (1 - standard_normal_cdf_approx(abs(z)))
    elif alternative == "greater":
        p_value = 1 - standard_normal_cdf_approx(z)
    else:  # less
        p_value = standard_normal_cdf_approx(z)
```

```
    return z, p_value


def standard_normal_cdf_approx(x: float) -> float:
    """Approximation of standard normal CDF."""
    return 0.5 * (1 + math.erf(x / math.sqrt(2)))
```

5.3 T-Test (Muestras Pequeñas)

```
def t_test_two_sample(
    group1: List[float],
    group2: List[float]
) -> tuple[float, float]:
    """Two-sample t-test (Welch's t-test).

    Tests if two groups have different means.
    Does not assume equal variances.

    H₀: μ₁ = μ₂
    H₁: μ₁ ≠ μ₂
    """
    n1, n2 = len(group1), len(group2)
    mean1, mean2 = sample_mean(group1), sample_mean(group2)
    var1 = sample_variance_unbiased(group1)
    var2 = sample_variance_unbiased(group2)

    # Welch's t-statistic
    se = math.sqrt(var1/n1 + var2/n2)
    t_stat = (mean1 - mean2) / se

    # Welch-Satterthwaite degrees of freedom
    num = (var1/n1 + var2/n2) ** 2
    denom = (var1/n1)**2/(n1-1) + (var2/n2)**2/(n2-1)
    df = num / denom

    # Approximate p-value (would need t-distribution for exact)
    # For large df, t approaches normal
    p_value = 2 * (1 - standard_normal_cdf_approx(abs(t_stat)))

    return t_stat, p_value
```

5.4 Chi-Square Test (Datos Categóricos)

```
def chi_square_test(
    observed: List[int],
    expected: List[float]
) -> tuple[float, int]:
    """Chi-square goodness of fit test.

    Tests if observed frequencies match expected.

    χ² = Σ (O - E)² / E

    Returns:
        chi_square_statistic, degrees_of_freedom
    """
    chi_sq = sum(
        (o - e) ** 2 / e
        for o, e in zip(observed, expected)
    )
    df = len(observed) - 1
```

```
    return chi_sq, df
```

---

6. Regresión Estadística {#6-regresion}

6.1 Regresión Lineal Simple

```
def linear_regression_ols(
    X: List[float],
    y: List[float]
) -> tuple[float, float]:
    """Ordinary Least Squares linear regression.

    y = β₀ + β₁x + ε

    Minimizes Σ(yᵢ - ŷᵢ)²

    Returns:
        intercept (β₀), slope (β₁)
    """
    n = len(X)
    mean_x = sum(X) / n
    mean_y = sum(y) / n

    # Slope: β₁ = Cov(x,y) / Var(x)
    numerator = sum((xi - mean_x) * (yi - mean_y) for xi, yi in zip(X, y))
    denominator = sum((xi - mean_x) ** 2 for xi in X)

    slope = numerator / denominator
    intercept = mean_y - slope * mean_x

    return intercept, slope


def r_squared(y_true: List[float], y_pred: List[float]) -> float:
    """Coefficient of determination (R²).

    R² = 1 - SS_res / SS_tot

    Proportion of variance explained by the model.
    0 ≤ R² ≤ 1 (for linear regression with intercept)
    """
    mean_y = sum(y_true) / len(y_true)

    ss_tot = sum((yi - mean_y) ** 2 for yi in y_true)
    ss_res = sum((yt - yp) ** 2 for yt, yp in zip(y_true, y_pred))

    return 1 - ss_res / ss_tot
```

6.2 Regresión Lineal Múltiple (Forma Matricial)

```
def matrix_multiply(A: List[List[float]], B: List[List[float]]) -> List[List[float]]:
    """Matrix multiplication A × B."""
    rows_a, cols_a = len(A), len(A[0])
    cols_b = len(B[0])

    result = [[0.0] * cols_b for _ in range(rows_a)]

    for i in range(rows_a):
        for j in range(cols_b):
```

```
            for k in range(cols_a):
                result[i][j] += A[i][k] * B[k][j]

    return result


def transpose(A: List[List[float]]) -> List[List[float]]:
    """Matrix transpose."""
    return [[A[j][i] for j in range(len(A))] for i in range(len(A[0]))]


# Note: Full OLS requires matrix inversion
# β = (XᵀX)⁻¹ Xᵀy
# In practice, use numerical libraries (numpy.linalg.lstsq)
```

⚠️ Conexión con Machine Learning

```
ESTADÍSTICA → MACHINE LEARNING:
──────────────────────────────

• MLE → Training neural networks (minimize cross-entropy)
• MAP → Regularization (L1/L2 penalties)
• Hypothesis testing → Model comparison, A/B testing
• Confidence intervals → Uncertainty quantification
• Bias-variance → Model selection, regularization tuning

DIFERENCIAS DE ENFOQUE:
• Estadística: explicar, inferir sobre parámetros
• ML: predecir, generalizar a nuevos datos

Pero los fundamentos matemáticos son los MISMOS.
```

🔧 Ejercicios Prácticos

Ejercicio 20.1: MLE para Datos Reales

Dado un dataset de tiempos de respuesta, estimar λ de distribución exponencial.

Ejercicio 20.2: A/B Testing

Implementar prueba de proporciones para comparar dos versiones.

Ejercicio 20.3: Regresión con Regularización

Comparar OLS vs Ridge (MAP con prior gaussiano).

📚 Recursos Externos

Recurso	Tipo	Prioridad
Statistical Learning	Libro (gratis)	🔴 Obligatorio
Seeing Theory	Interactivo	🟡 Recomendado
StatQuest: Statistics	Videos	🔴 Obligatorio

🧭 Navegación

← Anterior	Índice	Siguiente →
19_PROBABILIDAD_FUNDAMENTOS	00_INDICE	21_CADENAS_MARKOV_MONTECARLO

Módulo 06 - Markov y Monte Carlo

Guía MS in AI Pathway

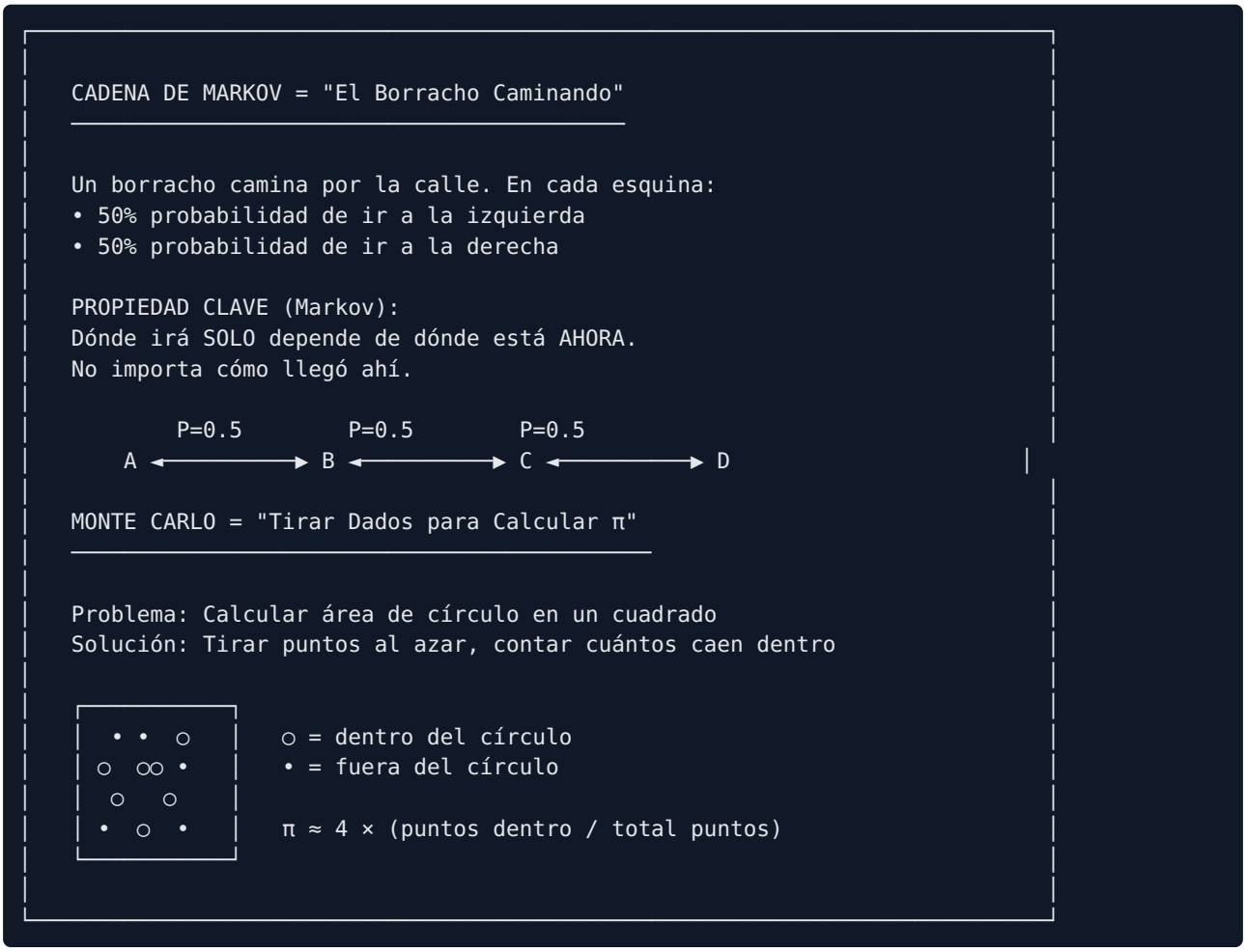
DUQUEOM · 2025

Módulo 06 - Cadenas de Markov y Métodos Monte Carlo

> 🎯 **Objetivo:** Dominar procesos estocásticos y métodos de muestreo
> ⭐ **PATHWAY LÍNEA 2:** Discrete-Time Markov Chains and Monte Carlo Methods

---

🧠 Analogía: Random Walks y Dados Infinitos



📋 Contenido

---

1. Cadenas de Markov Discretas {#1-markov-discretas}

1.1 Definición y Matriz de Transición

```
CADENA DE MARKOV:
```

Secuencia de estados X_0, X_1, X_2, ... donde:

$P(X_{n+1} = j \mid X_n = i, X_{n-1}, \ldots, X_0) = P(X_{n+1} = j \mid X_n = i)$

"El futuro solo depende del presente, no del pasado"

MATRIZ DE TRANSICIÓN P:
$P_{ij} = P(\text{ir a estado } j \mid \text{estoy en estado } i)$

• Cada fila suma 1 (probabilidades)
• $P_{ij} \geq 0$

```
from typing import List, Dict, Tuple
import random
import math


class MarkovChain:
    """Discrete-time Markov chain implementation.

    Example - Weather:
        states = ["sunny", "rainy"]
        transitions = {
            "sunny": {"sunny": 0.8, "rainy": 0.2},
            "rainy": {"sunny": 0.4, "rainy": 0.6}
        }
    """

    def __init__(
        self,
        states: List[str],
        transition_matrix: Dict[str, Dict[str, float]]
    ) -> None:
        """Initialize Markov chain.

        Args:
            states: List of state names
            transition_matrix: P[from_state][to_state] = probability
        """
        self.states = states
        self.transitions = transition_matrix
        self._validate()

    def _validate(self) -> None:
        """Validate that rows sum to 1."""
        for state in self.states:
            row_sum = sum(self.transitions[state].values())
            if abs(row_sum - 1.0) > 1e-10:
                raise ValueError(f"Row for state {state} sums to {row_sum}, not 1")

    def step(self, current_state: str) -> str:
        """Take one step from current state.

        Returns next state sampled from transition probabilities.
        """
        probs = self.transitions[current_state]
        r = random.random()

        cumulative = 0.0
        for state, prob in probs.items():
            cumulative += prob
            if r <= cumulative:
```

```
                return state

        return list(probs.keys())[-1]  # Edge case

    def simulate(self, start_state: str, n_steps: int) -> List[str]:
        """Simulate n steps of the Markov chain.

        Returns list of states visited.
        """
        trajectory = [start_state]
        current = start_state

        for _ in range(n_steps):
            current = self.step(current)
            trajectory.append(current)

        return trajectory

    def get_matrix(self) -> List[List[float]]:
        """Return transition matrix as 2D list."""
        n = len(self.states)
        matrix = [[0.0] * n for _ in range(n)]

        state_to_idx = {s: i for i, s in enumerate(self.states)}

        for from_state, to_probs in self.transitions.items():
            i = state_to_idx[from_state]
            for to_state, prob in to_probs.items():
                j = state_to_idx[to_state]
                matrix[i][j] = prob

        return matrix


# Example: Weather model
weather_chain = MarkovChain(
    states=["sunny", "rainy"],
    transition_matrix={
        "sunny": {"sunny": 0.8, "rainy": 0.2},
        "rainy": {"sunny": 0.4, "rainy": 0.6}
    }
)
```

1.2 Distribución después de n pasos

```
def matrix_power(P: List[List[float]], n: int) -> List[List[float]]:
    """Compute P^n (transition matrix to the nth power).

    P^n[i][j] = probability of being in state j after n steps,
                starting from state i.
    """
    size = len(P)

    # Start with identity matrix
    result = [[1.0 if i == j else 0.0 for j in range(size)] for i in range(size)]

    # Matrix multiplication n times
    current = [row[:] for row in P]  # Copy P

    while n > 0:
        if n % 2 == 1:
            result = matrix_multiply(result, current)
```

```
        current = matrix_multiply(current, current)
        n //= 2

    return result


def matrix_multiply(A: List[List[float]], B: List[List[float]]) -> List[List[float]]:
    """Multiply two matrices."""
    n = len(A)
    result = [[0.0] * n for _ in range(n)]

    for i in range(n):
        for j in range(n):
            for k in range(n):
                result[i][j] += A[i][k] * B[k][j]

    return result


def distribution_after_n_steps(
    initial_dist: List[float],
    P: List[List[float]],
    n: int
) -> List[float]:
    """Compute probability distribution after n steps.

    π(n) = π(0) × P^n

    Args:
        initial_dist: Initial probability for each state
        P: Transition matrix
        n: Number of steps

    Returns:
        Probability distribution after n steps
    """
    P_n = matrix_power(P, n)

    result = [0.0] * len(initial_dist)
    for j in range(len(initial_dist)):
        for i in range(len(initial_dist)):
            result[j] += initial_dist[i] * P_n[i][j]

    return result
```

---

2. Propiedades y Distribución Estacionaria {#2-propiedades}

2.1 Propiedades Importantes

```
PROPIEDADES DE CADENAS DE MARKOV:
═══════════════════════════════════

IRREDUCIBLE:
• Desde cualquier estado se puede llegar a cualquier otro
• Un solo componente comunicante

APERIÓDICA:
• El GCD de los ciclos de retorno es 1
• No hay ciclos determinísticos

ERGÓDICA = Irreducible + Aperiódica + Finita
```

```
• Tiene distribución estacionaria ÚNICA
• Converge a estacionaria desde cualquier inicio
```

2.2 Distribución Estacionaria

```
DISTRIBUCIÓN ESTACIONARIA π:
═══════════════════════════

Una distribución π tal que:
π = π × P

"Si empiezo con distribución π, después de un paso
sigo teniendo distribución π"

TEOREMA ERGÓDICO:
Para cadenas ergódicas, sin importar el estado inicial:
lim(n→∞) P^n converge a matriz con filas iguales a π

APLICACIÓN:
• PageRank: π = importancia de cada página web
• Física estadística: π = distribución de equilibrio
```

```
def find_stationary_power_iteration(
    P: List[List[float]],
    tolerance: float = 1e-10,
    max_iterations: int = 1000
) -> List[float]:
    """Find stationary distribution using power iteration.

    Iteratively multiply by P until convergence.

    π × P = π means π is left eigenvector with eigenvalue 1.
    """
    n = len(P)

    # Start with uniform distribution
    pi = [1.0 / n] * n

    for iteration in range(max_iterations):
        # Compute π × P
        new_pi = [0.0] * n
        for j in range(n):
            for i in range(n):
                new_pi[j] += pi[i] * P[i][j]

        # Check convergence
        diff = sum(abs(new_pi[i] - pi[i]) for i in range(n))
        if diff < tolerance:
            return new_pi

        pi = new_pi

    return pi  # May not have converged


def verify_stationary(pi: List[float], P: List[List[float]]) -> bool:
    """Verify that π is stationary: π × P ≈ π."""
    n = len(pi)
    result = [0.0] * n

    for j in range(n):
        for i in range(n):
```

```
            result[j] += pi[i] * P[i][j]

    diff = sum(abs(result[i] - pi[i]) for i in range(n))
    return diff < 1e-6
```

2.3 Tiempo de Mezcla (Mixing Time)

```
def estimate_mixing_time(
    chain: MarkovChain,
    epsilon: float = 0.01,
    samples: int = 1000
) -> int:
    """Estimate how many steps until distribution is close to stationary.

    Uses empirical simulation.
    """
    P = chain.get_matrix()
    pi = find_stationary_power_iteration(P)

    for n in range(1, 1000):
        P_n = matrix_power(P, n)

        # Total variation distance from stationary
        max_dist = 0.0
        for i in range(len(chain.states)):
            dist = 0.5 * sum(abs(P_n[i][j] - pi[j]) for j in range(len(pi)))
            max_dist = max(max_dist, dist)

        if max_dist < epsilon:
            return n

    return -1  # Did not converge
```

---

3. Algoritmos Basados en Markov {#3-algoritmos}

3.1 PageRank (Google)

```
def pagerank(
    graph: Dict[str, List[str]],
    damping: float = 0.85,
    iterations: int = 100
) -> Dict[str, float]:
    """PageRank algorithm - a Markov chain on the web graph.

    Random surfer model:
    - With probability d, follow a random outgoing link
    - With probability 1-d, jump to a random page

    Args:
        graph: Adjacency list (page -> list of linked pages)
        damping: Probability of following a link (typically 0.85)
        iterations: Number of power iterations

    Returns:
        PageRank score for each page
    """
    pages = list(graph.keys())
    n = len(pages)
    page_to_idx = {p: i for i, p in enumerate(pages)}
```

```
    # Initialize uniform
    rank = {page: 1.0 / n for page in pages}

    for _ in range(iterations):
        new_rank = {}

        for page in pages:
            # Teleport contribution (random jump)
            score = (1 - damping) / n

            # Link contribution
            for other_page in pages:
                if page in graph.get(other_page, []):
                    out_degree = len(graph[other_page])
                    if out_degree > 0:
                        score += damping * rank[other_page] / out_degree

            new_rank[page] = score

        rank = new_rank

    return rank


# Example
web_graph = {
    "A": ["B", "C"],
    "B": ["C"],
    "C": ["A"],
    "D": ["C"]
}
# scores = pagerank(web_graph)
```

3.2 Random Walk para Búsqueda en Grafos

```
def random_walk_similarity(
    graph: Dict[str, List[str]],
    node1: str,
    node2: str,
    walk_length: int = 100,
    num_walks: int = 1000
) -> float:
    """Estimate similarity between nodes via random walks.

    Probability of reaching node2 from node1 via random walk.
    """
    hits = 0

    for _ in range(num_walks):
        current = node1

        for _ in range(walk_length):
            neighbors = graph.get(current, [])
            if not neighbors:
                break
            current = random.choice(neighbors)

            if current == node2:
                hits += 1
                break

    return hits / num_walks
```

4. Métodos Monte Carlo {#4-monte-carlo}

4.1 Integración Monte Carlo

```
def monte_carlo_integration(
    f: callable,
    a: float,
    b: float,
    n_samples: int = 10000
) -> float:
    """Estimate ∫[a,b] f(x) dx using Monte Carlo.

    Estimate = (b-a) × (1/n) × Σ f(xᵢ)

    where xᵢ ~ Uniform(a, b)
    """
    total = 0.0

    for _ in range(n_samples):
        x = random.uniform(a, b)
        total += f(x)

    return (b - a) * total / n_samples


def estimate_pi(n_samples: int = 100000) -> float:
    """Estimate π using Monte Carlo.

    Ratio of points inside quarter circle to total points.
    Area of quarter circle = πr²/4, area of square = r²
    π = 4 × (points inside) / (total points)
    """
    inside = 0

    for _ in range(n_samples):
        x = random.random()
        y = random.random()

        if x*x + y*y <= 1:
            inside += 1

    return 4 * inside / n_samples
```

4.2 Muestreo por Importancia (Importance Sampling)

```
def importance_sampling(
    f: callable,           # Function to integrate
    p: callable,           # Proposal distribution density
    sample_p: callable,    # Function to sample from p
    n_samples: int = 10000
) -> float:
    """Importance sampling for E_q[f(x)] where q is hard to sample.

    E_q[f(x)] = E_p[f(x) × q(x)/p(x)]

    where p is easy to sample from.
    """
    total = 0.0

    for _ in range(n_samples):
```

```
        x = sample_p()
        # Weight = q(x) / p(x), here we assume q is the target
        weight = 1.0  # Simplified; actual requires q(x)/p(x) ratio
        total += f(x) * weight

    return total / n_samples
```

---

5. Markov Chain Monte Carlo (MCMC) {#5-mcmc}

5.1 Por Qué MCMC

```
┌─────────────────────────────────────────────────────────────────────┐
│                                                                     │
│   EL PROBLEMA DE MUESTREO                                           │
│   ────────────────────────                                          │
│                                                                     │
│   Queremos muestras de una distribución P(x)                        │
│   pero:                                                             │
│   • No podemos calcular P(x) directamente                           │
│   • Solo conocemos P(x) ∝ f(x) (hasta una constante)                │
│   • El espacio es de alta dimensión                                 │
│                                                                     │
│   SOLUCIÓN: MCMC                                                    │
│   ──────────────                                                    │
│   Construir una cadena de Markov cuya distribución estacionaria     │
│   sea exactamente la distribución que queremos muestrear.           │
│                                                                     │
│   Después de "burn-in", las muestras de la cadena                   │
│   son muestras aproximadas de P(x).                                 │
│                                                                     │
│   APLICACIONES:                                                     │
│   • Inferencia Bayesiana (posterior sampling)                       │
│   • Modelos generativos                                             │
│   • Física estadística                                              │
│   • Optimización estocástica                                        │
│                                                                     │
└─────────────────────────────────────────────────────────────────────┘
```

5.2 Algoritmo Metropolis-Hastings

```
def metropolis_hastings(
    target_log_prob: callable,  # Log of target distribution (up to constant)
    proposal_sample: callable,  # Sample from proposal q(x'|x)
    initial_state: List[float],
    n_samples: int = 10000,
    burn_in: int = 1000
) -> List[List[float]]:
    """Metropolis-Hastings MCMC sampler.

    1. Propose x' ~ q(x'|x)
    2. Accept with probability min(1, [P(x')/P(x)] × [q(x|x')/q(x'|x)])
    3. If symmetric proposal (q(x|x') = q(x'|x)), acceptance = min(1, P(x')/P(x))

    Args:
        target_log_prob: Log probability of target (can be unnormalized)
        proposal_sample: Function that proposes new state given current
        initial_state: Starting point
        n_samples: Number of samples to collect (after burn-in)
        burn_in: Number of initial samples to discard
```

```
    Returns:
        List of samples from target distribution
    """
    samples = []
    current = initial_state
    current_log_prob = target_log_prob(current)

    total_iterations = n_samples + burn_in
    accepted = 0

    for i in range(total_iterations):
        # Propose new state
        proposed = proposal_sample(current)
        proposed_log_prob = target_log_prob(proposed)

        # Acceptance ratio (log scale for numerical stability)
        # Assuming symmetric proposal
        log_acceptance = proposed_log_prob - current_log_prob

        # Accept or reject
        if math.log(random.random()) < log_acceptance:
            current = proposed
            current_log_prob = proposed_log_prob
            accepted += 1

        # Collect sample (after burn-in)
        if i >= burn_in:
            samples.append(current[:])  # Copy

    acceptance_rate = accepted / total_iterations
    print(f"Acceptance rate: {acceptance_rate:.2%}")

    return samples


# Example: Sample from 2D Gaussian
def gaussian_log_prob(x: List[float]) -> float:
    """Log probability of standard 2D Gaussian."""
    return -0.5 * (x[0]**2 + x[1]**2)


def gaussian_proposal(x: List[float]) -> List[float]:
    """Random walk proposal."""
    step_size = 0.5
    return [xi + random.gauss(0, step_size) for xi in x]


# samples = metropolis_hastings(gaussian_log_prob, gaussian_proposal, [0.0, 0.0])
```

5.3 Gibbs Sampling

```
def gibbs_sampling_2d(
    conditional_x: callable,  # Sample x given y
    conditional_y: callable,  # Sample y given x
    initial: Tuple[float, float],
    n_samples: int = 10000,
    burn_in: int = 1000
) -> List[Tuple[float, float]]:
    """Gibbs sampling for 2D distribution.

    Instead of proposing both coordinates at once,
    sample each coordinate from its conditional distribution.
```

```
    Always accepts! No rejection step.

    Requirements:
    - Must be able to sample from P(x|y) and P(y|x)
    """
    samples = []
    x, y = initial

    for i in range(n_samples + burn_in):
        # Sample x given current y
        x = conditional_x(y)

        # Sample y given new x
        y = conditional_y(x)

        if i >= burn_in:
            samples.append((x, y))

    return samples


# Example: Bivariate normal with correlation
def sample_x_given_y(y: float, rho: float = 0.8) -> float:
    """Sample x | y for bivariate normal with correlation rho."""
    mean = rho * y
    std = math.sqrt(1 - rho**2)
    return random.gauss(mean, std)


def sample_y_given_x(x: float, rho: float = 0.8) -> float:
    """Sample y | x for bivariate normal with correlation rho."""
    mean = rho * x
    std = math.sqrt(1 - rho**2)
    return random.gauss(mean, std)
```

---

6. Aplicaciones en ML {#6-aplicaciones}

6.1 Inferencia Bayesiana con MCMC

```
def bayesian_linear_regression_mcmc(
    X: List[List[float]],
    y: List[float],
    n_samples: int = 5000
) -> List[List[float]]:
    """Bayesian linear regression using MCMC.

    Prior: weights ~ Normal(0, 1)
    Likelihood: y ~ Normal(Xw, σ²)

    Sample from posterior P(w|X,y).
    """
    n_features = len(X[0])

    def log_posterior(weights: List[float]) -> float:
        # Log prior: N(0, 1) for each weight
        log_prior = -0.5 * sum(w**2 for w in weights)

        # Log likelihood
        sigma = 1.0  # Assume known for simplicity
        log_likelihood = 0.0
```

```
        for xi, yi in zip(X, y):
            pred = sum(w * x for w, x in zip(weights, xi))
            log_likelihood += -0.5 * ((yi - pred) / sigma) ** 2

        return log_prior + log_likelihood

    def proposal(weights: List[float]) -> List[float]:
        return [w + random.gauss(0, 0.1) for w in weights]

    initial = [0.0] * n_features

    return metropolis_hastings(log_posterior, proposal, initial, n_samples)
```

6.2 Hidden Markov Models (HMM)

```
class HiddenMarkovModel:
    """Simple Hidden Markov Model.

    Hidden states follow Markov chain.
    Observations depend only on current hidden state.

    Used for: speech recognition, sequence labeling, financial modeling.
    """

    def __init__(
        self,
        states: List[str],
        observations: List[str],
        transition_probs: Dict[str, Dict[str, float]],
        emission_probs: Dict[str, Dict[str, float]],
        initial_probs: Dict[str, float]
    ) -> None:
        self.states = states
        self.observations = observations
        self.A = transition_probs    # P(next_state | current_state)
        self.B = emission_probs      # P(observation | state)
        self.pi = initial_probs      # P(initial_state)

    def generate_sequence(self, length: int) -> Tuple[List[str], List[str]]:
        """Generate hidden states and observations."""
        hidden = []
        observed = []

        # Initial state
        state = self._sample_from_dist(self.pi)

        for _ in range(length):
            hidden.append(state)

            # Emit observation
            obs = self._sample_from_dist(self.B[state])
            observed.append(obs)

            # Transition to next state
            state = self._sample_from_dist(self.A[state])

        return hidden, observed

    def _sample_from_dist(self, dist: Dict[str, float]) -> str:
        """Sample from a discrete distribution."""
        r = random.random()
        cumulative = 0.0
```

```
        for item, prob in dist.items():
            cumulative += prob
            if r <= cumulative:
                return item
        return list(dist.keys())[-1]

    def forward_algorithm(self, observations: List[str]) -> float:
        """Compute P(observations | model) using forward algorithm.

        Dynamic programming to avoid exponential computation.
        """
        T = len(observations)

        # Alpha[t][state] = P(O₁...Oₜ, Sₜ = state)
        alpha = [{} for _ in range(T)]

        # Initialize
        for state in self.states:
            alpha[0][state] = self.pi[state] * self.B[state][observations[0]]

        # Forward pass
        for t in range(1, T):
            for state in self.states:
                prob = sum(
                    alpha[t-1][prev_state] * self.A[prev_state][state]
                    for prev_state in self.states
                )
                alpha[t][state] = prob * self.B[state][observations[t]]

        # Total probability
        return sum(alpha[T-1][state] for state in self.states)
```

⚠️ Conceptos Clave

Convergence Diagnostics

```
¿CÓMO SABER SI MCMC CONVERGIÓ?
──────────────────────────────

1. TRACE PLOTS: Visualizar muestras vs iteración
   - Debe parecer "ruido estacionario"
   - No tendencias ni patrones

2. MULTIPLE CHAINS: Correr varias cadenas desde diferentes inicios
   - Deben mezclarse (coincidir)

3. AUTOCORRELATION: Muestras consecutivas correlacionadas
   - Usar "thinning" (tomar cada k-ésima muestra)

4. EFFECTIVE SAMPLE SIZE: Muestras independientes efectivas
   - ESS << N indica alta autocorrelación
```

Burn-in y Thinning

```
def process_mcmc_samples(
    raw_samples: List[List[float]],
    burn_in_fraction: float = 0.2,
    thinning: int = 10
) -> List[List[float]]:
    """Post-process MCMC samples.
```

```
    1. Discard burn-in (initial samples before convergence)
    2. Thin (reduce autocorrelation)
    """
    n_samples = len(raw_samples)
    burn_in = int(n_samples * burn_in_fraction)

    # Remove burn-in and thin
    processed = raw_samples[burn_in::thinning]

    return processed
```

🔧 Ejercicios Prácticos

Ejercicio 21.1: Weather Markov Chain

Simular 365 días de clima con la cadena sunny/rainy.

Ejercicio 21.2: Estimate π

Usar Monte Carlo para estimar π con 1M puntos.

Ejercicio 21.3: Sample from Mixture

Usar Metropolis-Hastings para muestrear de mezcla de Gaussianas.

📚 Recursos Externos

Recurso	Tipo	Prioridad
MCMC for ML	Curso	🔴 Obligatorio
PageRank Explained	Video	🟡 Recomendado
Markov Chains (3B1B)	Video	🔴 Obligatorio

🧭 Navegación

← Anterior	Índice	Siguiente →
20_ESTADISTICA_INFERENCIAL	00_INDICE	22_ML_SUPERVISADO

Módulo 07 - ML Supervisado ⭐

Guía MS in AI Pathway

DUQUEOM · 2025

Módulo 07 - Machine Learning Supervisado

> 🎯 **Objetivo:** Dominar algoritmos de aprendizaje supervisado
> ⭐ **PATHWAY LÍNEA 1:** Introduction to Machine Learning: Supervised Learning

---

🧠 Analogía: Aprender de Ejemplos con Respuestas

```
┌─────────────────────────────────────────────────────────────┐
│                                                             │
│   APRENDIZAJE SUPERVISADO = Un Maestro con Respuestas       │
│   ──────────────────────────────────────────────────        │
│                                                             │
│   ENTRADA (X)          ETIQUETA (y)         MODELO APRENDE  │
│   ──────────────       ──────────────       ──────────────  │
│   Foto de gato    →    "gato"          →    Patrones de gatos│
│   Email spam      →    "spam"          →    Palabras sospechosas│
│   Precio casa     →    $500,000        →    Relación features/precio│
│                                                             │
│   OBJETIVO:                                                 │
│   Encontrar función f(X) ≈ y que GENERALICE a datos nuevos  │
│                                                             │
│   DOS TIPOS:                                                │
│   ┌──────────────────┐        ┌──────────────────┐          │
│   │  REGRESIÓN       │        │  CLASIFICACIÓN   │          │
│   │  y ∈ ℝ (número)  │        │  y ∈ {0,1,...}   │          │
│   │  Ej: precio      │        │  Ej: spam/no     │          │
│   └──────────────────┘        └──────────────────┘          │
│                                                             │
└─────────────────────────────────────────────────────────────┘
```

📋 Contenido

---

1. Fundamentos del Aprendizaje Supervisado {#1-fundamentos}

1.1 El Pipeline de ML

```
┌─────────────────────────────────────────────────────────────┐
│                    PIPELINE DE ML SUPERVISADO               │
│                                                             │
│   DATOS ──► PREPROCESO ──► SPLIT ──► TRAIN ──► EVALUATE ──► DEPLOY  │
│                                                             │
│   1. Recolectar datos (X, y)                                │
│   2. Limpiar, normalizar, codificar                         │
│   3. Dividir en train/validation/test (70/15/15)            │
│   4. Entrenar modelo en train set                           │
│   5. Evaluar en validation, ajustar hiperparámetros         │
│   6. Evaluación final en test set                           │
│   7. Desplegar si el rendimiento es satisfactorio           │
```

```
┌─────────────────────────────────────────────────────────────────────┐
│                                                                     │
└─────────────────────────────────────────────────────────────────────┘
```

1.2 Bias-Variance Tradeoff ⭐ FUNDAMENTAL

```
┌─────────────────────────────────────────────────────────────────────┐
│                                                                     │
│   ERROR = BIAS² + VARIANCE + RUIDO IRREDUCIBLE                      │
│                                                                     │
│   BIAS (Sesgo):                                                     │
│   ──────────────                                                    │
│   Error por suposiciones incorrectas del modelo.                    │
│   Modelo muy simple → Alto bias → UNDERFITTING                      │
│                                                                     │
│   VARIANCE (Varianza):                                              │
│   ──────────────────────                                            │
│   Sensibilidad a fluctuaciones en datos de entrenamiento.           │
│   Modelo muy complejo → Alta varianza → OVERFITTING                 │
│                                                                     │
│        Error                                                        │
│          ▲                                                          │
│          │    \                    /                               │
│          │     \     Total       /                                 │
│          │      \   Error      /                                   │
│          │       \    ▼      /                                     │
│          │        \________/                                       │
│          │   Bias  \      /  Variance                              │
│          │    ▼     \    /    ▼                                    │
│          │──────────\──/──────────► Complejidad                     │
│          │           \/                                            │
│          │         Óptimo                                          │
│                                                                     │
└─────────────────────────────────────────────────────────────────────┘
```

1.3 Implementación de Train/Test Split

```python
from typing import List, Tuple, TypeVar
import random

T = TypeVar('T')

def train_test_split(
    X: List[T],
    y: List[T],
    test_size: float = 0.2,
    random_state: int = None
) -> Tuple[List[T], List[T], List[T], List[T]]:
    """Split data into training and test sets.

    Args:
        X: Features
        y: Labels
        test_size: Fraction for test set
        random_state: Seed for reproducibility

    Returns:
        X_train, X_test, y_train, y_test
    """
    if random_state is not None:
        random.seed(random_state)

    n = len(X)
```

```
    indices = list(range(n))
    random.shuffle(indices)

    n_test = int(n * test_size)

    test_indices = set(indices[:n_test])

    X_train = [X[i] for i in range(n) if i not in test_indices]
    X_test = [X[i] for i in range(n) if i in test_indices]
    y_train = [y[i] for i in range(n) if i not in test_indices]
    y_test = [y[i] for i in range(n) if i in test_indices]

    return X_train, X_test, y_train, y_test


def k_fold_split(
    n_samples: int,
    k: int = 5
) -> List[Tuple[List[int], List[int]]]:
    """Generate K-fold cross-validation indices.

    Returns list of (train_indices, val_indices) for each fold.
    """
    indices = list(range(n_samples))
    random.shuffle(indices)

    fold_size = n_samples // k
    folds = []

    for i in range(k):
        start = i * fold_size
        end = start + fold_size if i < k - 1 else n_samples

        val_indices = indices[start:end]
        train_indices = indices[:start] + indices[end:]

        folds.append((train_indices, val_indices))

    return folds
```

---

2. Regresión Lineal y Logística {#2-regresion}

2.1 Regresión Lineal desde Cero

```
import math

class LinearRegression:
    """Linear regression using gradient descent.

    Model: y = Xw + b
    Loss: MSE = (1/n) Σ(yᵢ - ŷᵢ)²

    Gradient:
        ∂L/∂w = -(2/n) Xᵀ(y - ŷ)
        ∂L/∂b = -(2/n) Σ(y - ŷ)
    """

    def __init__(self, learning_rate: float = 0.01, n_iterations: int = 1000):
        self.lr = learning_rate
        self.n_iter = n_iterations
        self.weights: List[float] = []
```

```
        self.bias: float = 0.0
        self.loss_history: List[float] = []

    def fit(self, X: List[List[float]], y: List[float]) -> 'LinearRegression':
        """Train the model using gradient descent."""
        n_samples = len(X)
        n_features = len(X[0])

        # Initialize weights
        self.weights = [0.0] * n_features
        self.bias = 0.0

        for _ in range(self.n_iter):
            # Predictions
            y_pred = self._predict(X)

            # Calculate gradients
            dw = [0.0] * n_features
            db = 0.0

            for i in range(n_samples):
                error = y_pred[i] - y[i]
                for j in range(n_features):
                    dw[j] += (2 / n_samples) * error * X[i][j]
                db += (2 / n_samples) * error

            # Update weights
            for j in range(n_features):
                self.weights[j] -= self.lr * dw[j]
            self.bias -= self.lr * db

            # Track loss
            loss = self._mse(y, y_pred)
            self.loss_history.append(loss)

        return self

    def _predict(self, X: List[List[float]]) -> List[float]:
        """Make predictions."""
        predictions = []
        for xi in X:
            pred = self.bias + sum(w * x for w, x in zip(self.weights, xi))
            predictions.append(pred)
        return predictions

    def predict(self, X: List[List[float]]) -> List[float]:
        """Public prediction method."""
        return self._predict(X)

    def _mse(self, y_true: List[float], y_pred: List[float]) -> float:
        """Mean Squared Error."""
        return sum((yt - yp) ** 2 for yt, yp in zip(y_true, y_pred)) / len(y_true)
```

2.2 Regresión Logística desde Cero ⭐

```
class LogisticRegression:
    """Logistic regression for binary classification.

    Model: P(y=1|x) = σ(wᵀx + b) = 1 / (1 + e^(-(wᵀx + b)))

    Loss: Binary Cross-Entropy
        L = -(1/n) Σ [yᵢ log(p̂ᵢ) + (1-yᵢ) log(1-p̂ᵢ)]
```

```
    Gradient:
        ∂L/∂w = (1/n) Xᵀ(p̂ - y)
        ∂L/∂b = (1/n) Σ(p̂ - y)
    """

    def __init__(self, learning_rate: float = 0.01, n_iterations: int = 1000):
        self.lr = learning_rate
        self.n_iter = n_iterations
        self.weights: List[float] = []
        self.bias: float = 0.0

    def _sigmoid(self, z: float) -> float:
        """Sigmoid activation function."""
        if z < -500:
            return 0.0
        elif z > 500:
            return 1.0
        return 1.0 / (1.0 + math.exp(-z))

    def fit(self, X: List[List[float]], y: List[int]) -> 'LogisticRegression':
        """Train using gradient descent."""
        n_samples = len(X)
        n_features = len(X[0])

        self.weights = [0.0] * n_features
        self.bias = 0.0

        for _ in range(self.n_iter):
            # Forward pass
            linear = [
                self.bias + sum(w * x for w, x in zip(self.weights, xi))
                for xi in X
            ]
            predictions = [self._sigmoid(z) for z in linear]

            # Gradients
            dw = [0.0] * n_features
            db = 0.0

            for i in range(n_samples):
                error = predictions[i] - y[i]
                for j in range(n_features):
                    dw[j] += (1 / n_samples) * error * X[i][j]
                db += (1 / n_samples) * error

            # Update
            for j in range(n_features):
                self.weights[j] -= self.lr * dw[j]
            self.bias -= self.lr * db

        return self

    def predict_proba(self, X: List[List[float]]) -> List[float]:
        """Predict probabilities."""
        return [
            self._sigmoid(self.bias + sum(w * x for w, x in zip(self.weights, xi)))
            for xi in X
        ]

    def predict(self, X: List[List[float]], threshold: float = 0.5) -> List[int]:
        """Predict class labels."""
```

```
        probs = self.predict_proba(X)
        return [1 if p >= threshold else 0 for p in probs]
```

2.3 Regularización (L1/L2)

```
class RidgeRegression(LinearRegression):
    """Linear regression with L2 regularization.

    Loss = MSE + λ Σ wⱼ²

    L2 shrinks weights but doesn't set them to zero.
    Equivalent to MAP estimation with Gaussian prior.
    """

    def __init__(self, learning_rate: float = 0.01, n_iterations: int = 1000,
                 alpha: float = 1.0):
        super().__init__(learning_rate, n_iterations)
        self.alpha = alpha  # Regularization strength

    def fit(self, X: List[List[float]], y: List[float]) -> 'RidgeRegression':
        """Train with L2 regularization."""
        n_samples = len(X)
        n_features = len(X[0])

        self.weights = [0.0] * n_features
        self.bias = 0.0

        for _ in range(self.n_iter):
            y_pred = self._predict(X)

            # Gradients with regularization
            dw = [0.0] * n_features
            db = 0.0

            for i in range(n_samples):
                error = y_pred[i] - y[i]
                for j in range(n_features):
                    dw[j] += (2 / n_samples) * error * X[i][j]
                db += (2 / n_samples) * error

            # Add L2 penalty gradient
            for j in range(n_features):
                dw[j] += 2 * self.alpha * self.weights[j]

            # Update
            for j in range(n_features):
                self.weights[j] -= self.lr * dw[j]
            self.bias -= self.lr * db

        return self
```

---

3. Árboles de Decisión {#3-arboles}

3.1 Concepto y Criterios de Split

```
ÁRBOL DE DECISIÓN:
──────────────────

                    [Feature₁ < 5?]
                    /             \
```

```
                  Yes              No
                 /                   \
        [Feature₂ < 3?]           [Clase: B]
         /          \
       Yes          No
       /              \
   [Clase: A]     [Clase: C]

CRITERIOS DE SPLIT:
• GINI IMPURITY (Clasificación):
  Gini(S) = 1 - Σ pᵢ²
  Mide "impureza" de un nodo

• INFORMATION GAIN (Clasificación):
  IG = Entropy(parent) - Σ (nⱼ/n) × Entropy(childⱼ)
  Entropy(S) = -Σ pᵢ log₂(pᵢ)

• MSE (Regresión):
  Split que minimiza varianza en nodos hijos
```

3.2 Implementación Simplificada

```python
from typing import Dict, Any, Optional
from collections import Counter

class DecisionTreeClassifier:
    """Simple decision tree for classification.

    Uses Gini impurity for splitting.
    """

    def __init__(self, max_depth: int = 10, min_samples_split: int = 2):
        self.max_depth = max_depth
        self.min_samples = min_samples_split
        self.tree: Optional[Dict[str, Any]] = None

    def _gini(self, y: List[int]) -> float:
        """Calculate Gini impurity."""
        if not y:
            return 0.0
        counts = Counter(y)
        n = len(y)
        return 1.0 - sum((count / n) ** 2 for count in counts.values())

    def _best_split(
        self,
        X: List[List[float]],
        y: List[int]
    ) -> Tuple[Optional[int], Optional[float], float]:
        """Find best feature and threshold to split on."""
        best_gain = -1
        best_feature = None
        best_threshold = None

        n_features = len(X[0])
        parent_gini = self._gini(y)
        n = len(y)

        for feature_idx in range(n_features):
            # Get unique values for thresholds
            values = sorted(set(x[feature_idx] for x in X))
            thresholds = [(values[i] + values[i+1]) / 2
```

```
                        for i in range(len(values) - 1)]

            for threshold in thresholds:
                # Split data
                left_y = [y[i] for i in range(n) if X[i][feature_idx] <= threshold]
                right_y = [y[i] for i in range(n) if X[i][feature_idx] > threshold]

                if not left_y or not right_y:
                    continue

                # Calculate information gain
                left_gini = self._gini(left_y)
                right_gini = self._gini(right_y)
                weighted_gini = (len(left_y) * left_gini +
                                 len(right_y) * right_gini) / n
                gain = parent_gini - weighted_gini

                if gain > best_gain:
                    best_gain = gain
                    best_feature = feature_idx
                    best_threshold = threshold

        return best_feature, best_threshold, best_gain

    def _build_tree(
        self,
        X: List[List[float]],
        y: List[int],
        depth: int
    ) -> Dict[str, Any]:
        """Recursively build the tree."""
        n_samples = len(y)
        n_classes = len(set(y))

        # Stopping conditions
        if (depth >= self.max_depth or
            n_samples < self.min_samples or
            n_classes == 1):
            # Create leaf node
            return {"leaf": True, "class": Counter(y).most_common(1)[0][0]}

        # Find best split
        feature, threshold, gain = self._best_split(X, y)

        if feature is None or gain <= 0:
            return {"leaf": True, "class": Counter(y).most_common(1)[0][0]}

        # Split data
        left_indices = [i for i in range(n_samples) if X[i][feature] <= threshold]
        right_indices = [i for i in range(n_samples) if X[i][feature] > threshold]

        left_X = [X[i] for i in left_indices]
        left_y = [y[i] for i in left_indices]
        right_X = [X[i] for i in right_indices]
        right_y = [y[i] for i in right_indices]

        return {
            "leaf": False,
            "feature": feature,
            "threshold": threshold,
            "left": self._build_tree(left_X, left_y, depth + 1),
            "right": self._build_tree(right_X, right_y, depth + 1)
        }
```

```
    def fit(self, X: List[List[float]], y: List[int]) -> 'DecisionTreeClassifier':
        """Build the decision tree."""
        self.tree = self._build_tree(X, y, 0)
        return self

    def _predict_one(self, x: List[float], node: Dict[str, Any]) -> int:
        """Predict class for single sample."""
        if node["leaf"]:
            return node["class"]

        if x[node["feature"]] <= node["threshold"]:
            return self._predict_one(x, node["left"])
        else:
            return self._predict_one(x, node["right"])

    def predict(self, X: List[List[float]]) -> List[int]:
        """Predict classes for samples."""
        return [self._predict_one(x, self.tree) for x in X]
```

---

4. K-Nearest Neighbors {#4-knn}

4.1 Implementación

```
class KNearestNeighbors:
    """K-Nearest Neighbors classifier.

    Non-parametric: no training, just stores data.
    Prediction: vote of k nearest neighbors.

    Time: O(n × d) per prediction (n samples, d dimensions)
    """

    def __init__(self, k: int = 3):
        self.k = k
        self.X_train: List[List[float]] = []
        self.y_train: List[int] = []

    def fit(self, X: List[List[float]], y: List[int]) -> 'KNearestNeighbors':
        """Store training data."""
        self.X_train = X
        self.y_train = y
        return self

    def _euclidean_distance(self, x1: List[float], x2: List[float]) -> float:
        """Calculate Euclidean distance."""
        return math.sqrt(sum((a - b) ** 2 for a, b in zip(x1, x2)))

    def _predict_one(self, x: List[float]) -> int:
        """Predict class for single sample."""
        # Calculate distances to all training samples
        distances = [
            (self._euclidean_distance(x, x_train), y_train)
            for x_train, y_train in zip(self.X_train, self.y_train)
        ]

        # Sort by distance and get k nearest
        distances.sort(key=lambda d: d[0])
        k_nearest = distances[:self.k]

        # Vote
```

```
        votes = Counter(label for _, label in k_nearest)
        return votes.most_common(1)[0][0]

    def predict(self, X: List[List[float]]) -> List[int]:
        """Predict classes."""
        return [self._predict_one(x) for x in X]
```

---

5. Support Vector Machines {#5-svm}

5.1 Concepto

```
┌─────────────────────────────────────────────────────────────────┐
│                                                                 │
│   SUPPORT VECTOR MACHINES                                       │
│   ──────────────────────                                        │
│                                                                 │
│   Objetivo: Encontrar el HIPERPLANO que separa las clases       │
│             con el MÁXIMO MARGEN.                               │
│                                                                 │
│        ○  ○                                                     │
│     ○        ○                                                  │
│        ○                      ← Support Vectors (los más cercanos)
│   ─ ─ ─ ─ ─ ─ ─ ─ ─           ← Margen                          │
│   ═══════════════════         ← Hiperplano separador (wᵀx + b = 0)
│   ─ ─ ─ ─ ─ ─ ─ ─ ─           ← Margen                          │
│        ●                      ← Support Vectors                 │
│     ●        ●                                                  │
│        ●  ●                                                     │
│                                                                 │
│   MARGEN = 2 / ||w||                                            │
│   Maximizar margen = Minimizar ||w||²                           │
│                                                                 │
│   KERNEL TRICK:                                                 │
│   Para datos no linealmente separables, proyectar a dimensión superior
│   donde SÍ sean separables (sin calcular la proyección explícitamente).
│                                                                 │
└─────────────────────────────────────────────────────────────────┘
```

5.2 Implementación Simplificada (Lineal)

```
class LinearSVM:
    """Linear SVM using gradient descent (simplified).

    Minimizes: (1/2)||w||² + C × Σ max(0, 1 - yᵢ(wᵀxᵢ + b))

    Hinge loss for classification.
    """

    def __init__(self, learning_rate: float = 0.001,
                 n_iterations: int = 1000, C: float = 1.0):
        self.lr = learning_rate
        self.n_iter = n_iterations
        self.C = C  # Regularization parameter
        self.weights: List[float] = []
        self.bias: float = 0.0

    def fit(self, X: List[List[float]], y: List[int]) -> 'LinearSVM':
        """Train SVM. Labels should be -1 or 1."""
        # Convert 0/1 to -1/1 if needed
        y = [1 if label == 1 else -1 for label in y]
```

```
        n_samples = len(X)
        n_features = len(X[0])

        self.weights = [0.0] * n_features
        self.bias = 0.0

        for _ in range(self.n_iter):
            for i in range(n_samples):
                # Check if sample satisfies margin constraint
                margin = y[i] * (sum(w * x for w, x in zip(self.weights, X[i])) +
self.bias)

                if margin >= 1:
                    # Correctly classified with margin
                    # Only regularization gradient
                    for j in range(n_features):
                        self.weights[j] -= self.lr * self.weights[j]
                else:
                    # Misclassified or within margin
                    # Regularization + hinge loss gradient
                    for j in range(n_features):
                        self.weights[j] -= self.lr * (self.weights[j] -
                                                      self.C * y[i] * X[i][j])
                    self.bias -= self.lr * (-self.C * y[i])

        return self

    def predict(self, X: List[List[float]]) -> List[int]:
        """Predict class labels (0 or 1)."""
        predictions = []
        for x in X:
            score = sum(w * xi for w, xi in zip(self.weights, x)) + self.bias
            predictions.append(1 if score >= 0 else 0)
        return predictions
```

---

6. Evaluación de Modelos {#6-evaluacion}

6.1 Métricas de Clasificación

```
def confusion_matrix(y_true: List[int], y_pred: List[int]) -> Dict[str, int]:
    """Calculate confusion matrix components.

    Returns:
        TP, TN, FP, FN counts
    """
    tp = sum(1 for yt, yp in zip(y_true, y_pred) if yt == 1 and yp == 1)
    tn = sum(1 for yt, yp in zip(y_true, y_pred) if yt == 0 and yp == 0)
    fp = sum(1 for yt, yp in zip(y_true, y_pred) if yt == 0 and yp == 1)
    fn = sum(1 for yt, yp in zip(y_true, y_pred) if yt == 1 and yp == 0)

    return {"TP": tp, "TN": tn, "FP": fp, "FN": fn}


def accuracy(y_true: List[int], y_pred: List[int]) -> float:
    """Accuracy = (TP + TN) / Total."""
    cm = confusion_matrix(y_true, y_pred)
    total = cm["TP"] + cm["TN"] + cm["FP"] + cm["FN"]
    return (cm["TP"] + cm["TN"]) / total if total > 0 else 0.0
```

```
def precision(y_true: List[int], y_pred: List[int]) -> float:
    """Precision = TP / (TP + FP).

    Of all predicted positive, how many are actually positive?
    """
    cm = confusion_matrix(y_true, y_pred)
    denom = cm["TP"] + cm["FP"]
    return cm["TP"] / denom if denom > 0 else 0.0


def recall(y_true: List[int], y_pred: List[int]) -> float:
    """Recall = TP / (TP + FN).

    Of all actual positive, how many did we find?
    Also called Sensitivity or True Positive Rate.
    """
    cm = confusion_matrix(y_true, y_pred)
    denom = cm["TP"] + cm["FN"]
    return cm["TP"] / denom if denom > 0 else 0.0


def f1_score(y_true: List[int], y_pred: List[int]) -> float:
    """F1 = 2 × (Precision × Recall) / (Precision + Recall).

    Harmonic mean of precision and recall.
    """
    p = precision(y_true, y_pred)
    r = recall(y_true, y_pred)
    return 2 * p * r / (p + r) if (p + r) > 0 else 0.0
```

6.2 Métricas de Regresión

```
def mean_squared_error(y_true: List[float], y_pred: List[float]) -> float:
    """MSE = (1/n) Σ(yᵢ - ŷᵢ)²."""
    return sum((yt - yp) ** 2 for yt, yp in zip(y_true, y_pred)) / len(y_true)


def root_mean_squared_error(y_true: List[float], y_pred: List[float]) -> float:
    """RMSE = √MSE."""
    return math.sqrt(mean_squared_error(y_true, y_pred))


def mean_absolute_error(y_true: List[float], y_pred: List[float]) -> float:
    """MAE = (1/n) Σ|yᵢ - ŷᵢ|."""
    return sum(abs(yt - yp) for yt, yp in zip(y_true, y_pred)) / len(y_true)


def r_squared(y_true: List[float], y_pred: List[float]) -> float:
    """R² = 1 - SS_res / SS_tot.

    Proportion of variance explained.
    """
    mean_y = sum(y_true) / len(y_true)
    ss_tot = sum((y - mean_y) ** 2 for y in y_true)
    ss_res = sum((yt - yp) ** 2 for yt, yp in zip(y_true, y_pred))

    return 1 - ss_res / ss_tot if ss_tot > 0 else 0.0
```

6.3 Cross-Validation

```
def cross_validate(
    model_class,
```

```
    X: List[List[float]],
    y: List,
    k: int = 5,
    **model_params
) -> List[float]:
    """K-fold cross-validation.

    Returns accuracy for each fold.
    """
    folds = k_fold_split(len(X), k)
    scores = []

    for train_idx, val_idx in folds:
        X_train = [X[i] for i in train_idx]
        y_train = [y[i] for i in train_idx]
        X_val = [X[i] for i in val_idx]
        y_val = [y[i] for i in val_idx]

        model = model_class(**model_params)
        model.fit(X_train, y_train)
        y_pred = model.predict(X_val)

        score = accuracy(y_val, y_pred)
        scores.append(score)

    return scores
```

🔧 Ejercicios Prácticos

Ejercicio 22.1: Regresión Lineal

Implementar y entrenar regresión lineal en datos sintéticos.

Ejercicio 22.2: Clasificador de Spam

Usar Logistic Regression para clasificar emails.

Ejercicio 22.3: Comparar Modelos

Evaluar Decision Tree vs KNN vs Logistic Regression con cross-validation.

📚 Recursos Externos

Recurso	Tipo	Prioridad
Andrew Ng's ML Course	Curso	🔴 Obligatorio
Hands-On ML Book	Libro	🔴 Obligatorio
StatQuest: ML	Videos	🟡 Recomendado

🧭 Navegación

← Anterior	Índice	Siguiente →
21_CADENAS_MARKOV_MONTECARLO	00_INDICE	23_ML_NO_SUPERVISADO

Módulo 08 - ML No Supervisado ⭐

Guía MS in AI Pathway

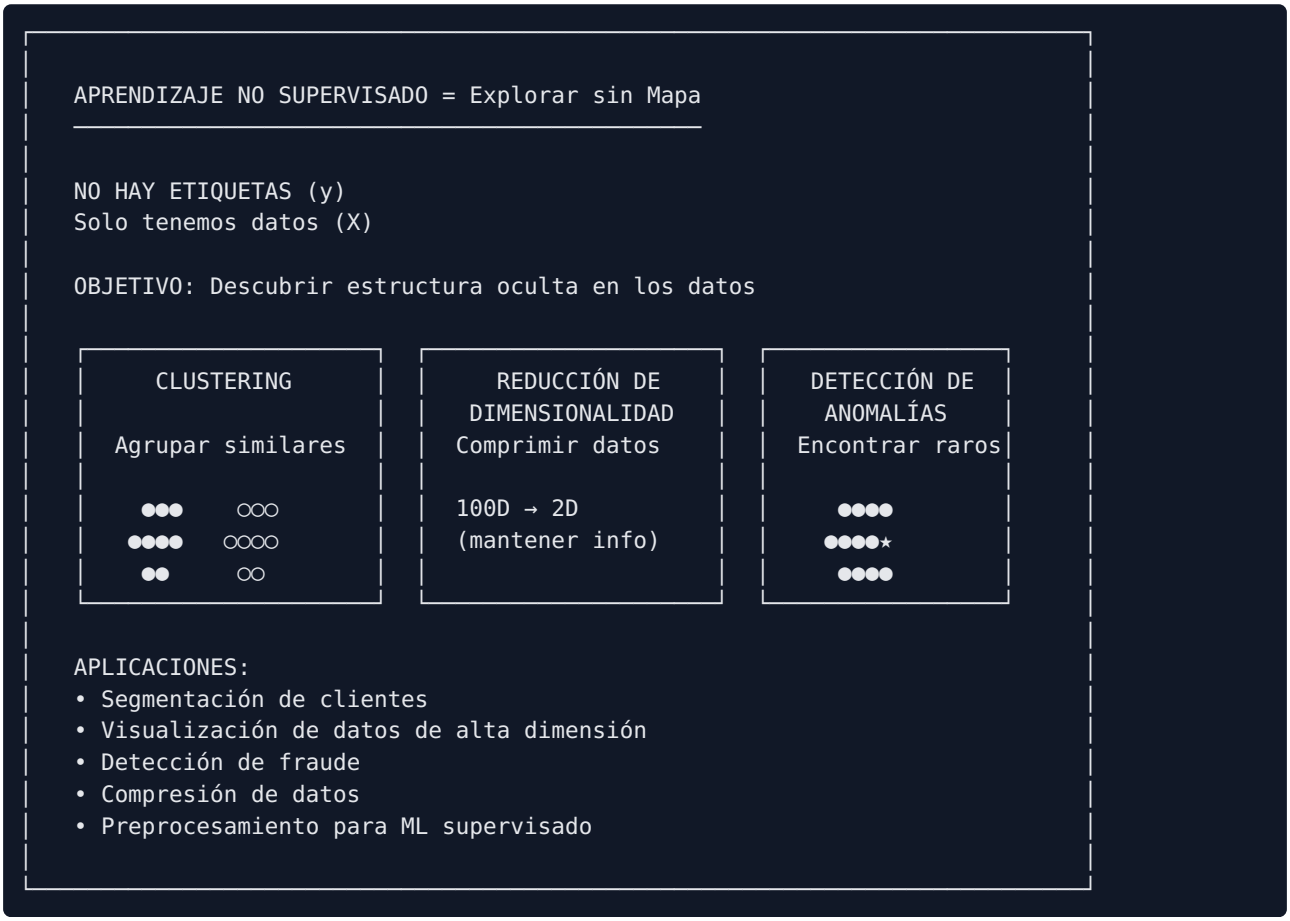
DUQUEOM · 2025

Módulo 08 - Machine Learning No Supervisado

> 🎯 **Objetivo:** Dominar clustering, reducción de dimensionalidad y detección de anomalías
> ⭐ **PATHWAY LÍNEA 1:** Unsupervised Algorithms in Machine Learning

---

🧠 Analogía: Encontrar Patrones sin Respuestas



📋 Contenido

---

1. Fundamentos de Clustering {#1-fundamentos}

1.1 ¿Qué es Clustering?

```
CLUSTERING:
──────────

Particionar n objetos en k grupos (clusters) donde:
```

```
• Objetos dentro del mismo cluster son SIMILARES
• Objetos en diferentes clusters son DIFERENTES

MEDIDAS DE SIMILITUD/DISTANCIA:
──────────────────────────────

EUCLIDIANA: d(x,y) = √Σ(xᵢ - yᵢ)²
• La más común
• Sensible a escala

MANHATTAN: d(x,y) = Σ|xᵢ - yᵢ|
• Para grids

COSENO: sim(x,y) = (x·y) / (||x|| × ||y||)
• Para texto, documentos
• Ignora magnitud, solo dirección

¿CUÁNTOS CLUSTERS?
• Elbow method
• Silhouette score
• Domain knowledge
```

1.2 Métricas de Evaluación

```python
from typing import List, Dict
import math
from collections import defaultdict

def euclidean_distance(x: List[float], y: List[float]) -> float:
    """Euclidean distance between two points."""
    return math.sqrt(sum((xi - yi) ** 2 for xi, yi in zip(x, y)))


def manhattan_distance(x: List[float], y: List[float]) -> float:
    """Manhattan (L1) distance."""
    return sum(abs(xi - yi) for xi, yi in zip(x, y))


def cosine_similarity(x: List[float], y: List[float]) -> float:
    """Cosine similarity between vectors."""
    dot = sum(xi * yi for xi, yi in zip(x, y))
    norm_x = math.sqrt(sum(xi ** 2 for xi in x))
    norm_y = math.sqrt(sum(yi ** 2 for yi in y))

    if norm_x == 0 or norm_y == 0:
        return 0.0
    return dot / (norm_x * norm_y)


def inertia(X: List[List[float]], labels: List[int],
            centroids: List[List[float]]) -> float:
    """Within-cluster sum of squares (WCSS).

    Sum of squared distances from each point to its centroid.
    Lower is better (more compact clusters).
    """
    total = 0.0
    for x, label in zip(X, labels):
        dist = euclidean_distance(x, centroids[label])
        total += dist ** 2
    return total
```

```
def silhouette_sample(
    X: List[List[float]],
    labels: List[int],
    idx: int
) -> float:
    """Silhouette coefficient for a single sample.

    s(i) = (b(i) - a(i)) / max(a(i), b(i))

    a(i) = average distance to points in same cluster
    b(i) = average distance to points in nearest other cluster

    Range: [-1, 1], higher is better
    """
    x = X[idx]
    label = labels[idx]

    # Calculate a(i): mean distance to same cluster
    same_cluster = [
        euclidean_distance(x, X[j])
        for j in range(len(X))
        if labels[j] == label and j != idx
    ]
    a = sum(same_cluster) / len(same_cluster) if same_cluster else 0

    # Calculate b(i): min mean distance to other clusters
    other_clusters = set(labels) - {label}
    b = float('inf')

    for other_label in other_clusters:
        other_points = [
            euclidean_distance(x, X[j])
            for j in range(len(X))
            if labels[j] == other_label
        ]
        if other_points:
            mean_dist = sum(other_points) / len(other_points)
            b = min(b, mean_dist)

    if b == float('inf'):
        b = 0

    if max(a, b) == 0:
        return 0.0
    return (b - a) / max(a, b)


def silhouette_score(X: List[List[float]], labels: List[int]) -> float:
    """Average silhouette coefficient for all samples."""
    scores = [silhouette_sample(X, labels, i) for i in range(len(X))]
    return sum(scores) / len(scores)
```

---

2. K-Means Clustering {#2-kmeans}

2.1 Algoritmo

```
K-MEANS ALGORITHM:
──────────────────

1. INICIALIZAR: Elegir k centroides aleatorios
```

```
2. ASIGNAR: Cada punto al centroide más cercano
3. ACTUALIZAR: Mover cada centroide al promedio de sus puntos
4. REPETIR 2-3 hasta convergencia

CONVERGENCIA:
• Las asignaciones no cambian, O
• Centroides se mueven menos que ε

COMPLEJIDAD: O(n × k × d × i)
• n = puntos
• k = clusters
• d = dimensiones
• i = iteraciones
```

2.2 Implementación

```
import random

class KMeans:
    """K-Means clustering algorithm.

    Partitions n samples into k clusters by minimizing
    within-cluster sum of squares (inertia).
    """

    def __init__(self, n_clusters: int = 3, max_iterations: int = 100,
                 tolerance: float = 1e-4, random_state: int = None):
        self.k = n_clusters
        self.max_iter = max_iterations
        self.tol = tolerance
        self.random_state = random_state
        self.centroids: List[List[float]] = []
        self.labels: List[int] = []
        self.inertia_: float = 0.0

    def _init_centroids(self, X: List[List[float]]) -> List[List[float]]:
        """Initialize centroids randomly from data points."""
        if self.random_state is not None:
            random.seed(self.random_state)

        indices = random.sample(range(len(X)), self.k)
        return [X[i][:] for i in indices]  # Copy points

    def _assign_clusters(self, X: List[List[float]]) -> List[int]:
        """Assign each point to nearest centroid."""
        labels = []
        for x in X:
            distances = [euclidean_distance(x, c) for c in self.centroids]
            labels.append(distances.index(min(distances)))
        return labels

    def _update_centroids(
        self,
        X: List[List[float]],
        labels: List[int]
    ) -> List[List[float]]:
        """Update centroids to mean of assigned points."""
        n_features = len(X[0])
        new_centroids = []

        for k in range(self.k):
            cluster_points = [X[i] for i in range(len(X)) if labels[i] == k]
```

```
            if cluster_points:
                centroid = [
                    sum(p[d] for p in cluster_points) / len(cluster_points)
                    for d in range(n_features)
                ]
            else:
                # Empty cluster: keep old centroid
                centroid = self.centroids[k]

            new_centroids.append(centroid)

        return new_centroids

    def _centroid_shift(
        self,
        old: List[List[float]],
        new: List[List[float]]
    ) -> float:
        """Calculate total movement of centroids."""
        return sum(euclidean_distance(o, n) for o, n in zip(old, new))

    def fit(self, X: List[List[float]]) -> 'KMeans':
        """Fit K-Means to data."""
        self.centroids = self._init_centroids(X)

        for _ in range(self.max_iter):
            # Assign points to clusters
            self.labels = self._assign_clusters(X)

            # Update centroids
            new_centroids = self._update_centroids(X, self.labels)

            # Check convergence
            shift = self._centroid_shift(self.centroids, new_centroids)
            self.centroids = new_centroids

            if shift < self.tol:
                break

        # Calculate final inertia
        self.inertia_ = inertia(X, self.labels, self.centroids)

        return self

    def predict(self, X: List[List[float]]) -> List[int]:
        """Predict cluster for new points."""
        return self._assign_clusters(X)

    def fit_predict(self, X: List[List[float]]) -> List[int]:
        """Fit and return cluster labels."""
        self.fit(X)
        return self.labels


def elbow_method(X: List[List[float]], max_k: int = 10) -> List[float]:
    """Calculate inertia for different k values.

    Plot inertia vs k to find "elbow" point.
    """
    inertias = []

    for k in range(1, max_k + 1):
```

```
        kmeans = KMeans(n_clusters=k, random_state=42)
        kmeans.fit(X)
        inertias.append(kmeans.inertia_)

    return inertias
```

2.3 K-Means++: Mejor Inicialización

```
def kmeans_plus_plus_init(X: List[List[float]], k: int) -> List[List[float]]:
    """K-Means++ initialization.

    Spreads initial centroids by selecting them with probability
    proportional to their distance from existing centroids.

    Improves convergence and final clustering quality.
    """
    n = len(X)
    centroids = []

    # First centroid: random
    centroids.append(X[random.randint(0, n - 1)][:])

    for _ in range(1, k):
        # Calculate distance to nearest centroid for each point
        distances = []
        for x in X:
            min_dist = min(euclidean_distance(x, c) for c in centroids)
            distances.append(min_dist ** 2)  # Square for probability

        # Sample with probability proportional to distance²
        total = sum(distances)
        probs = [d / total for d in distances]

        r = random.random()
        cumulative = 0
        for i, p in enumerate(probs):
            cumulative += p
            if r <= cumulative:
                centroids.append(X[i][:])
                break

    return centroids
```

---

3. Clustering Jerárquico {#3-jerarquico}

3.1 Concepto

```
CLUSTERING JERÁRQUICO:
──────────────────────

AGLOMERATIVO (Bottom-up):
1. Empezar: cada punto es su propio cluster
2. Fusionar: los dos clusters más cercanos
3. Repetir hasta tener k clusters (o 1)

DIVISIVO (Top-down):
1. Empezar: todos los puntos en un cluster
2. Dividir: el cluster más grande
3. Repetir hasta tener k clusters
```

```
LINKAGE (Distancia entre clusters):
• SINGLE: min distancia entre puntos
• COMPLETE: max distancia entre puntos
• AVERAGE: promedio de distancias
• WARD: minimiza varianza al fusionar

DENDROGRAMA:
Visualización del proceso de fusión/división
```

3.2 Implementación Aglomerativa

```
class AgglomerativeClustering:
    """Agglomerative (bottom-up) hierarchical clustering.

    Uses single linkage by default.
    """

    def __init__(self, n_clusters: int = 2, linkage: str = 'single'):
        self.n_clusters = n_clusters
        self.linkage = linkage
        self.labels: List[int] = []
        self.merge_history: List[tuple] = []

    def _cluster_distance(
        self,
        cluster1: List[int],
        cluster2: List[int],
        distances: List[List[float]]
    ) -> float:
        """Calculate distance between two clusters."""
        if self.linkage == 'single':
            # Minimum distance between any two points
            return min(
                distances[i][j]
                for i in cluster1
                for j in cluster2
            )
        elif self.linkage == 'complete':
            # Maximum distance
            return max(
                distances[i][j]
                for i in cluster1
                for j in cluster2
            )
        elif self.linkage == 'average':
            # Average distance
            total = sum(
                distances[i][j]
                for i in cluster1
                for j in cluster2
            )
            return total / (len(cluster1) * len(cluster2))
        else:
            raise ValueError(f"Unknown linkage: {self.linkage}")

    def fit(self, X: List[List[float]]) -> 'AgglomerativeClustering':
        """Build hierarchical clustering."""
        n = len(X)

        # Precompute pairwise distances
        distances = [
            [euclidean_distance(X[i], X[j]) for j in range(n)]
```

```
            for i in range(n)
        ]

        # Initialize: each point is its own cluster
        clusters = [[i] for i in range(n)]

        while len(clusters) > self.n_clusters:
            # Find closest pair of clusters
            min_dist = float('inf')
            merge_i, merge_j = 0, 1

            for i in range(len(clusters)):
                for j in range(i + 1, len(clusters)):
                    dist = self._cluster_distance(
                        clusters[i], clusters[j], distances
                    )
                    if dist < min_dist:
                        min_dist = dist
                        merge_i, merge_j = i, j

            # Record merge
            self.merge_history.append((merge_i, merge_j, min_dist))

            # Merge clusters
            clusters[merge_i] = clusters[merge_i] + clusters[merge_j]
            clusters.pop(merge_j)

        # Assign labels
        self.labels = [0] * n
        for cluster_idx, cluster in enumerate(clusters):
            for point_idx in cluster:
                self.labels[point_idx] = cluster_idx

        return self

    def fit_predict(self, X: List[List[float]]) -> List[int]:
        """Fit and return labels."""
        self.fit(X)
        return self.labels
```

---

4. DBSCAN {#4-dbscan}

4.1 Concepto

```
DBSCAN (Density-Based Spatial Clustering):
──────────────────────────────────────────

NO requiere especificar número de clusters.
Encuentra clusters de forma arbitraria.
Detecta outliers automáticamente.

PARÁMETROS:
• ε (eps): Radio del vecindario
• min_samples: Mínimo puntos para ser core point

TIPOS DE PUNTOS:
• CORE: Tiene ≥ min_samples vecinos en radio ε
• BORDER: No es core, pero es vecino de core
• NOISE: No es core ni border (outlier)

ALGORITMO:
```

```
1. Para cada punto sin visitar:
   a. Si es core: crear nuevo cluster, expandir
   b. Si no: marcar como ruido (puede cambiar a border)
```

4.2 Implementación

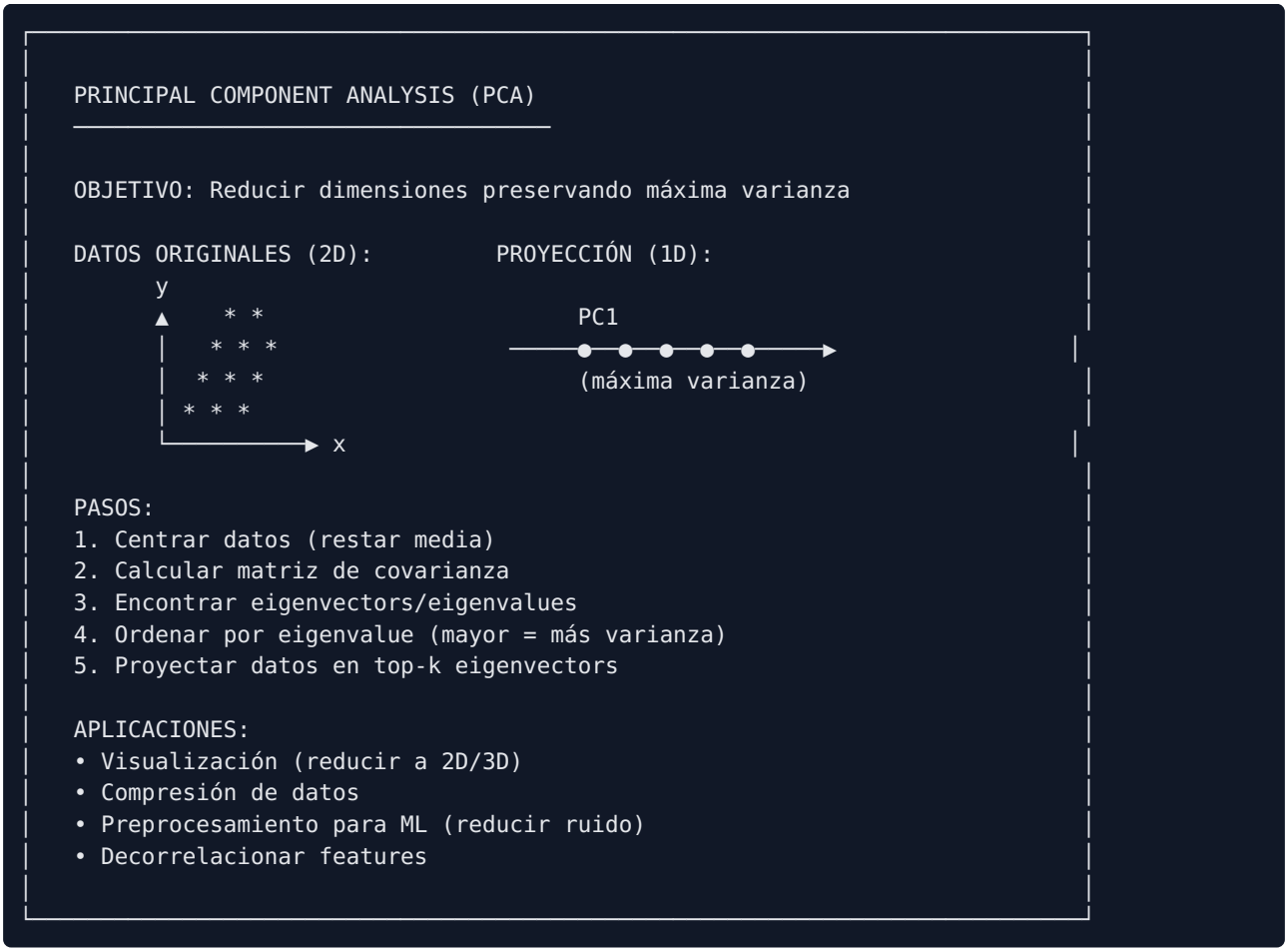
```
class DBSCAN:
    """Density-Based Spatial Clustering of Applications with Noise.

    Finds clusters based on density, automatically detecting outliers.
    """

    def __init__(self, eps: float = 0.5, min_samples: int = 5):
        self.eps = eps
        self.min_samples = min_samples
        self.labels: List[int] = []

    def _get_neighbors(
        self,
        X: List[List[float]],
        idx: int
    ) -> List[int]:
        """Find all points within eps distance."""
        neighbors = []
        for i in range(len(X)):
            if euclidean_distance(X[idx], X[i]) <= self.eps:
                neighbors.append(i)
        return neighbors

    def fit(self, X: List[List[float]]) -> 'DBSCAN':
        """Fit DBSCAN clustering."""
        n = len(X)
        self.labels = [-1] * n  # -1 = unassigned/noise

        cluster_id = 0

        for i in range(n):
            if self.labels[i] != -1:
                continue  # Already processed

            neighbors = self._get_neighbors(X, i)

            if len(neighbors) < self.min_samples:
                # Noise point (might become border later)
                continue

            # Start new cluster
            self.labels[i] = cluster_id

            # Expand cluster
            seed_set = neighbors[:]
            j = 0

            while j < len(seed_set):
                q = seed_set[j]

                if self.labels[q] == -1:
                    # Was noise, now border
                    self.labels[q] = cluster_id

                if self.labels[q] != -1 and self.labels[q] != cluster_id:
                    # Already in another cluster
```

```
                    j += 1
                    continue

                if self.labels[q] == -1 or self.labels[q] == cluster_id:
                    self.labels[q] = cluster_id

                    q_neighbors = self._get_neighbors(X, q)

                    if len(q_neighbors) >= self.min_samples:
                        # q is also core point
                        for neighbor in q_neighbors:
                            if neighbor not in seed_set:
                                seed_set.append(neighbor)

                j += 1

            cluster_id += 1

        return self

    def fit_predict(self, X: List[List[float]]) -> List[int]:
        """Fit and return labels (-1 for noise)."""
        self.fit(X)
        return self.labels
```

---

5. PCA - Reducción de Dimensionalidad {#5-pca}

5.1 Concepto



5.2 Implementación Simplificada

```
class PCA:
    """Principal Component Analysis for dimensionality reduction.

    Simplified implementation using power iteration for eigenvectors.
    """

    def __init__(self, n_components: int = 2):
        self.n_components = n_components
        self.components: List[List[float]] = []  # Principal components
        self.mean: List[float] = []
        self.explained_variance: List[float] = []

    def _center_data(self, X: List[List[float]]) -> List[List[float]]:
        """Center data by subtracting mean."""
        n, d = len(X), len(X[0])

        # Calculate mean
        self.mean = [sum(X[i][j] for i in range(n)) / n for j in range(d)]

        # Subtract mean
        centered = [
            [X[i][j] - self.mean[j] for j in range(d)]
            for i in range(n)
        ]

        return centered

    def _covariance_matrix(self, X: List[List[float]]) -> List[List[float]]:
        """Compute covariance matrix."""
        n, d = len(X), len(X[0])

        cov = [[0.0] * d for _ in range(d)]

        for i in range(d):
            for j in range(d):
                cov[i][j] = sum(X[k][i] * X[k][j] for k in range(n)) / (n - 1)

        return cov

    def _power_iteration(
        self,
        matrix: List[List[float]],
        n_iterations: int = 100
    ) -> tuple[List[float], float]:
        """Find dominant eigenvector using power iteration."""
        d = len(matrix)

        # Random initial vector
        v = [random.random() for _ in range(d)]
        norm = math.sqrt(sum(x ** 2 for x in v))
        v = [x / norm for x in v]

        for _ in range(n_iterations):
            # Matrix-vector multiplication
            v_new = [
                sum(matrix[i][j] * v[j] for j in range(d))
                for i in range(d)
            ]

            # Normalize
            norm = math.sqrt(sum(x ** 2 for x in v_new))
```

```
            if norm == 0:
                break
            v_new = [x / norm for x in v_new]

            v = v_new

        # Rayleigh quotient for eigenvalue
        Av = [sum(matrix[i][j] * v[j] for j in range(d)) for i in range(d)]
        eigenvalue = sum(v[i] * Av[i] for i in range(d))

        return v, eigenvalue

    def _deflate(
        self,
        matrix: List[List[float]],
        eigenvector: List[float],
        eigenvalue: float
    ) -> List[List[float]]:
        """Remove contribution of found eigenvector from matrix."""
        d = len(matrix)
        deflated = [row[:] for row in matrix]

        for i in range(d):
            for j in range(d):
                deflated[i][j] -= eigenvalue * eigenvector[i] * eigenvector[j]

        return deflated

    def fit(self, X: List[List[float]]) -> 'PCA':
        """Fit PCA to data."""
        # Center data
        X_centered = self._center_data(X)

        # Compute covariance matrix
        cov = self._covariance_matrix(X_centered)

        # Find principal components using power iteration + deflation
        self.components = []
        self.explained_variance = []

        for _ in range(self.n_components):
            eigenvector, eigenvalue = self._power_iteration(cov)
            self.components.append(eigenvector)
            self.explained_variance.append(eigenvalue)
            cov = self._deflate(cov, eigenvector, eigenvalue)

        return self

    def transform(self, X: List[List[float]]) -> List[List[float]]:
        """Project data onto principal components."""
        # Center data using fitted mean
        d = len(X[0])
        X_centered = [
            [X[i][j] - self.mean[j] for j in range(d)]
            for i in range(len(X))
        ]

        # Project onto components
        transformed = []
        for x in X_centered:
            projection = [
                sum(x[j] * self.components[k][j] for j in range(d))
                for k in range(self.n_components)
```

```
            ]
            transformed.append(projection)

        return transformed

    def fit_transform(self, X: List[List[float]]) -> List[List[float]]:
        """Fit and transform in one step."""
        self.fit(X)
        return self.transform(X)

    def explained_variance_ratio(self) -> List[float]:
        """Proportion of variance explained by each component."""
        total = sum(self.explained_variance)
        return [v / total for v in self.explained_variance]
```

---

6. Detección de Anomalías {#6-anomalias}

6.1 Métodos

```
DETECCIÓN DE ANOMALÍAS:
──────────────────────

BASADO EN ESTADÍSTICAS:
• Z-score: puntos con |z| > 3 son outliers
• IQR: puntos fuera de [Q1 - 1.5×IQR, Q3 + 1.5×IQR]

BASADO EN DENSIDAD:
• Local Outlier Factor (LOF)
• DBSCAN (noise points = outliers)

BASADO EN DISTANCIA:
• Puntos lejos de sus vecinos
• Isolation Forest

BASADO EN CLUSTERING:
• Puntos lejos de todos los centroides
• Clusters muy pequeños
```

6.2 Implementación de LOF

```
class LocalOutlierFactor:
    """Local Outlier Factor for anomaly detection.

    Compares local density of a point to its neighbors.
    High LOF = lower density than neighbors = potential outlier.
    """

    def __init__(self, n_neighbors: int = 20, threshold: float = 1.5):
        self.k = n_neighbors
        self.threshold = threshold
        self.lof_scores: List[float] = []

    def _k_neighbors(
        self,
        X: List[List[float]],
        idx: int
    ) -> List[tuple]:
        """Find k nearest neighbors and their distances."""
        distances = [
            (i, euclidean_distance(X[idx], X[i]))
```

```
            for i in range(len(X)) if i != idx
        ]
        distances.sort(key=lambda x: x[1])
        return distances[:self.k]

    def _reachability_distance(
        self,
        X: List[List[float]],
        idx: int,
        neighbor_idx: int,
        k_distances: Dict[int, float]
    ) -> float:
        """Reachability distance from idx to neighbor.

        max(k-distance(neighbor), actual_distance)
        """
        actual_dist = euclidean_distance(X[idx], X[neighbor_idx])
        return max(k_distances[neighbor_idx], actual_dist)

    def _local_reachability_density(
        self,
        X: List[List[float]],
        idx: int,
        neighbors: Dict[int, List[tuple]],
        k_distances: Dict[int, float]
    ) -> float:
        """LRD = inverse of average reachability distance to neighbors."""
        neighbor_list = neighbors[idx]

        if not neighbor_list:
            return 0.0

        total_reach_dist = sum(
            self._reachability_distance(X, idx, n_idx, k_distances)
            for n_idx, _ in neighbor_list
        )

        if total_reach_dist == 0:
            return float('inf')

        return len(neighbor_list) / total_reach_dist

    def fit_predict(self, X: List[List[float]]) -> List[int]:
        """Calculate LOF and return outlier labels.

        Returns 1 for inliers, -1 for outliers.
        """
        n = len(X)

        # Find k neighbors for all points
        neighbors = {}
        k_distances = {}

        for i in range(n):
            knn = self._k_neighbors(X, i)
            neighbors[i] = knn
            k_distances[i] = knn[-1][1] if knn else 0  # Distance to k-th neighbor

        # Calculate LRD for all points
        lrd = {}
        for i in range(n):
            lrd[i] = self._local_reachability_density(
                X, i, neighbors, k_distances
```

```
            )

        # Calculate LOF
        self.lof_scores = []

        for i in range(n):
            neighbor_list = neighbors[i]

            if not neighbor_list or lrd[i] == 0:
                self.lof_scores.append(1.0)
                continue

            # LOF = average ratio of neighbor's LRD to own LRD
            lof = sum(
                lrd[n_idx] / lrd[i] if lrd[i] != float('inf') else 0
                for n_idx, _ in neighbor_list
            ) / len(neighbor_list)

            self.lof_scores.append(lof)

        # Classify as outlier if LOF > threshold
        labels = [
            -1 if score > self.threshold else 1
            for score in self.lof_scores
        ]

        return labels
```

⚠️ Cuándo Usar Cada Algoritmo

```
GUÍA DE SELECCIÓN:
──────────────────

K-MEANS:
• Clusters esféricos de tamaño similar
• Conoces número de clusters
• Datos grandes (escalable)

HIERARCHICAL:
• Quieres dendrograma
• No conoces k
• Datos pequeños/medianos

DBSCAN:
• Clusters de forma arbitraria
• Hay outliers
• No conoces k

PCA:
• Reducir dimensionalidad
• Visualización
• Decorrelacionar features
```

🔧 Ejercicios Prácticos

Ejercicio 23.1: K-Means en datos sintéticos

Crear blobs y encontrar clusters óptimos con elbow method.

Ejercicio 23.2: Comparar algoritmos

K-Means vs DBSCAN en datos con forma de luna.

Ejercicio 23.3: PCA para visualización

Reducir MNIST a 2D y visualizar dígitos.

📚 Recursos Externos

Recurso	Tipo	Prioridad
Scikit-learn Clustering	Docs	🔴 Obligatorio
StatQuest: PCA	Video	🔴 Obligatorio
Visualizing DBSCAN	Interactivo	🟡 Recomendado

🧭 Navegación

← Anterior	Índice	Siguiente →
22_ML_SUPERVISADO	00_INDICE	24_INTRO_DEEP_LEARNING

Módulo 09 - Deep Learning ⭐

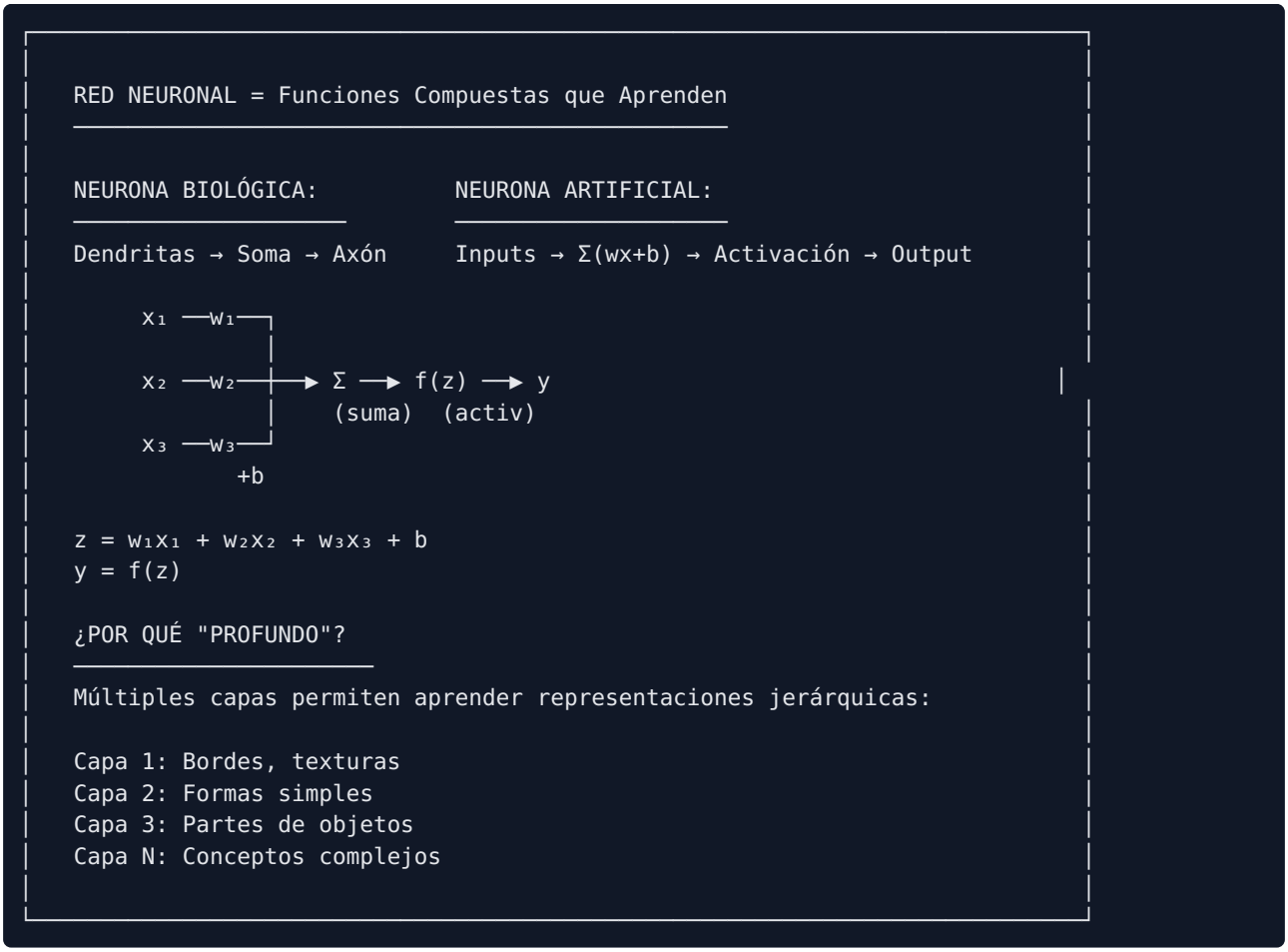
Guía MS in AI Pathway

DUQUEOM · 2025

Módulo 09 - Introducción al Deep Learning

> 🎯 **Objetivo:** Dominar fundamentos de redes neuronales y backpropagation
> ⭐ **PATHWAY LÍNEA 1:** Introduction to Deep Learning

🧠 Analogía: El Cerebro Artificial



📋 Contenido

---

1. Perceptrón y Neurona {#1-perceptron}

1.1 Perceptrón Simple

```
from typing import List, Tuple
import math
import random

class Perceptron:
```

```
    """Single layer perceptron (binary classifier).

    The simplest neural network: one neuron.
    Can only learn linearly separable patterns.

    Model: y = sign(w·x + b)
    """

    def __init__(self, n_features: int, learning_rate: float = 0.01):
        self.lr = learning_rate
        self.weights = [random.uniform(-1, 1) for _ in range(n_features)]
        self.bias = random.uniform(-1, 1)

    def predict_one(self, x: List[float]) -> int:
        """Predict for single sample."""
        z = sum(w * xi for w, xi in zip(self.weights, x)) + self.bias
        return 1 if z >= 0 else 0

    def predict(self, X: List[List[float]]) -> List[int]:
        """Predict for multiple samples."""
        return [self.predict_one(x) for x in X]

    def fit(self, X: List[List[float]], y: List[int],
            epochs: int = 100) -> 'Perceptron':
        """Train perceptron using perceptron learning rule.

        Update rule: w = w + lr × (y - ŷ) × x
        Only updates when prediction is wrong.
        """
        for _ in range(epochs):
            errors = 0
            for xi, yi in zip(X, y):
                y_pred = self.predict_one(xi)
                error = yi - y_pred

                if error != 0:
                    errors += 1
                    for j in range(len(self.weights)):
                        self.weights[j] += self.lr * error * xi[j]
                    self.bias += self.lr * error

            if errors == 0:
                break  # Converged

        return self
```

1.2 Neurona con Activación Continua

```
class Neuron:
    """Single neuron with continuous activation.

    More expressive than perceptron.
    Can use different activation functions.
    """

    def __init__(
        self,
        n_inputs: int,
        activation: str = 'sigmoid'
    ):
        self.weights = [random.gauss(0, 0.1) for _ in range(n_inputs)]
        self.bias = 0.0
```

```
        self.activation = activation

        # For backprop
        self.last_input: List[float] = []
        self.last_z: float = 0.0
        self.last_output: float = 0.0

    def _activate(self, z: float) -> float:
        """Apply activation function."""
        if self.activation == 'sigmoid':
            if z < -500:
                return 0.0
            elif z > 500:
                return 1.0
            return 1.0 / (1.0 + math.exp(-z))
        elif self.activation == 'relu':
            return max(0, z)
        elif self.activation == 'tanh':
            return math.tanh(z)
        elif self.activation == 'linear':
            return z
        else:
            raise ValueError(f"Unknown activation: {self.activation}")

    def _activation_derivative(self, z: float) -> float:
        """Derivative of activation function."""
        if self.activation == 'sigmoid':
            s = self._activate(z)
            return s * (1 - s)
        elif self.activation == 'relu':
            return 1.0 if z > 0 else 0.0
        elif self.activation == 'tanh':
            return 1 - math.tanh(z) ** 2
        elif self.activation == 'linear':
            return 1.0
        else:
            raise ValueError(f"Unknown activation: {self.activation}")

    def forward(self, x: List[float]) -> float:
        """Forward pass."""
        self.last_input = x
        self.last_z = sum(w * xi for w, xi in zip(self.weights, x)) + self.bias
        self.last_output = self._activate(self.last_z)
        return self.last_output
```

---

2. Funciones de Activación {#2-activaciones}

2.1 Comparación

```
┌─────────────────────────────────────────────────────────────────────────┐
│                    FUNCIONES DE ACTIVACIÓN                              │
├─────────────────────────────────────────────────────────────────────────┤
│                                                                         │
│   SIGMOID: σ(z) = 1 / (1 + e⁻ᶻ)                                        │
│   ──────────────────────────────                                        │
│   Rango: (0, 1)                                                         │
│   Uso: Output para probabilidad binaria                                 │
│   Problema: Vanishing gradient para |z| grande                          │
│                ___________                                              │
│               /                                                         │
│   ──────────/──────────                                                 │
```

```
┌──────────────────────────────────────────────────────────────────────────┐
│                                                                          │
│   TANH: tanh(z) = (eᶻ - e⁻ᶻ) / (eᶻ + e⁻ᶻ)                               │
│   ──────────────────────────────────────                                 │
│   Rango: (-1, 1)                                                         │
│   Uso: Capas ocultas (centrado en 0)                                     │
│          ___                                                             │
│         /                                                                │
│   _____/                                                                 │
│                                                                          │
│   ReLU: f(z) = max(0, z)                                                 │
│   ──────────────────────                                                 │
│   Rango: [0, ∞)                                                          │
│   Uso: ESTÁNDAR para capas ocultas                                       │
│   Ventaja: No vanishing gradient, rápido                                 │
│   Problema: "Dying ReLU" (neuronas muertas)                              │
│            /                                                             │
│   ________/                                                              │
│                                                                          │
│   Leaky ReLU: f(z) = max(αz, z), α ≈ 0.01                                │
│   ──────────────────────────────────────                                 │
│   Soluciona dying ReLU                                                   │
│            /                                                             │
│   _       /                                                              │
│    \____/                                                                │
│                                                                          │
│   Softmax: σ(z)ᵢ = eᶻⁱ / Σeᶻʲ                                           │
│   ──────────────────────────                                             │
│   Rango: (0, 1), suma = 1                                                │
│   Uso: Output para clasificación multiclase                              │
│                                                                          │
└──────────────────────────────────────────────────────────────────────────┘
```

2.2 Implementación

```python
def sigmoid(z: float) -> float:
    """Sigmoid activation."""
    if z < -500:
        return 0.0
    elif z > 500:
        return 1.0
    return 1.0 / (1.0 + math.exp(-z))


def sigmoid_derivative(z: float) -> float:
    """Derivative of sigmoid."""
    s = sigmoid(z)
    return s * (1 - s)


def relu(z: float) -> float:
    """ReLU activation."""
    return max(0, z)


def relu_derivative(z: float) -> float:
    """Derivative of ReLU."""
    return 1.0 if z > 0 else 0.0


def leaky_relu(z: float, alpha: float = 0.01) -> float:
    """Leaky ReLU activation."""
    return z if z > 0 else alpha * z
```

```
def softmax(z: List[float]) -> List[float]:
    """Softmax for vector (numerically stable)."""
    max_z = max(z)
    exp_z = [math.exp(zi - max_z) for zi in z]
    sum_exp = sum(exp_z)
    return [e / sum_exp for e in exp_z]
```

---

3. Redes Multicapa (MLP) {#3-mlp}

3.1 Arquitectura

```
MULTILAYER PERCEPTRON (MLP):
─────────────────────────────

INPUT       HIDDEN 1     HIDDEN 2     OUTPUT
LAYER       LAYER        LAYER        LAYER

  x₁ ──┬──────────┬──────────┬──────► ŷ₁
       │          │          │
  x₂ ──┼──────────┼──────────┼──────► ŷ₂
       │          │          │
  x₃ ──┴──────────┴──────────┴──────► ŷ₃

FORWARD PROPAGATION:
h₁ = f(W₁x + b₁)      # Primera capa oculta
h₂ = f(W₂h₁ + b₂)     # Segunda capa oculta
ŷ = g(W₃h₂ + b₃)      # Output (g puede ser softmax)

PARÁMETROS TOTALES:
Para arquitectura [input, h1, h2, output] = [784, 128, 64, 10]:
W₁: 784×128 + 128 = 100,480
W₂: 128×64 + 64 = 8,256
W₃: 64×10 + 10 = 650
Total: ~109,000 parámetros
```

3.2 Implementación

```
class Layer:
    """A single layer in a neural network."""

    def __init__(
        self,
        n_inputs: int,
        n_neurons: int,
        activation: str = 'relu'
    ):
        # Xavier initialization
        limit = math.sqrt(6 / (n_inputs + n_neurons))
        self.weights = [
            [random.uniform(-limit, limit) for _ in range(n_inputs)]
            for _ in range(n_neurons)
        ]
        self.biases = [0.0] * n_neurons
        self.activation = activation

        # Cache for backprop
        self.inputs: List[float] = []
        self.z: List[float] = []  # Pre-activation
```

```
        self.outputs: List[float] = []

        # Gradients
        self.weight_gradients: List[List[float]] = []
        self.bias_gradients: List[float] = []

    def _activate(self, z: float) -> float:
        """Apply activation function."""
        if self.activation == 'sigmoid':
            return sigmoid(z)
        elif self.activation == 'relu':
            return relu(z)
        elif self.activation == 'tanh':
            return math.tanh(z)
        elif self.activation == 'linear':
            return z
        else:
            raise ValueError(f"Unknown activation: {self.activation}")

    def _activation_derivative(self, z: float) -> float:
        """Derivative of activation function."""
        if self.activation == 'sigmoid':
            return sigmoid_derivative(z)
        elif self.activation == 'relu':
            return relu_derivative(z)
        elif self.activation == 'tanh':
            return 1 - math.tanh(z) ** 2
        elif self.activation == 'linear':
            return 1.0
        else:
            raise ValueError(f"Unknown activation: {self.activation}")

    def forward(self, inputs: List[float]) -> List[float]:
        """Forward pass through layer."""
        self.inputs = inputs
        self.z = []
        self.outputs = []

        for neuron_idx in range(len(self.weights)):
            # Linear combination
            z = sum(
                w * x for w, x in zip(self.weights[neuron_idx], inputs)
            ) + self.biases[neuron_idx]
            self.z.append(z)

            # Activation
            self.outputs.append(self._activate(z))

        return self.outputs

    def backward(self, output_gradients: List[float]) -> List[float]:
        """Backward pass: compute gradients."""
        n_neurons = len(self.weights)
        n_inputs = len(self.weights[0])

        # Gradient of activation
        activation_gradients = [
            output_gradients[i] * self._activation_derivative(self.z[i])
            for i in range(n_neurons)
        ]

        # Weight gradients
        self.weight_gradients = [
```

```
            [activation_gradients[i] * self.inputs[j] for j in range(n_inputs)]
            for i in range(n_neurons)
        ]

        # Bias gradients
        self.bias_gradients = activation_gradients[:]

        # Input gradients (for previous layer)
        input_gradients = [0.0] * n_inputs
        for j in range(n_inputs):
            for i in range(n_neurons):
                input_gradients[j] += activation_gradients[i] * self.weights[i][j]

        return input_gradients

    def update(self, learning_rate: float) -> None:
        """Update weights using computed gradients."""
        for i in range(len(self.weights)):
            for j in range(len(self.weights[i])):
                self.weights[i][j] -= learning_rate * self.weight_gradients[i][j]
            self.biases[i] -= learning_rate * self.bias_gradients[i]


class NeuralNetwork:
    """Multilayer Perceptron neural network."""

    def __init__(self, layer_sizes: List[int], activations: List[str] = None):
        """
        Args:
            layer_sizes: [input_size, hidden1, hidden2, ..., output_size]
            activations: activation for each layer (default: relu + linear)
        """
        if activations is None:
            activations = ['relu'] * (len(layer_sizes) - 2) + ['linear']

        self.layers = []
        for i in range(len(layer_sizes) - 1):
            layer = Layer(
                layer_sizes[i],
                layer_sizes[i + 1],
                activations[i]
            )
            self.layers.append(layer)

    def forward(self, x: List[float]) -> List[float]:
        """Forward pass through all layers."""
        output = x
        for layer in self.layers:
            output = layer.forward(output)
        return output

    def backward(self, loss_gradient: List[float]) -> None:
        """Backward pass through all layers."""
        gradient = loss_gradient
        for layer in reversed(self.layers):
            gradient = layer.backward(gradient)

    def update(self, learning_rate: float) -> None:
        """Update all layers."""
        for layer in self.layers:
            layer.update(learning_rate)

    def predict(self, X: List[List[float]]) -> List[List[float]]:
```

```
        """Predict for batch."""
        return [self.forward(x) for x in X]
```

---

4. Backpropagation {#4-backpropagation}

4.1 La Regla de la Cadena

```
┌─────────────────────────────────────────────────────────────────────┐
│                                                                     │
│   BACKPROPAGATION = Regla de la Cadena Aplicada                     │
│   ═══════════════════════════════════════════                       │
│                                                                     │
│   Objetivo: ∂L/∂wᵢⱼ (cómo cambiar cada peso para reducir el loss)   │
│                                                                     │
│   Forward:  x → [W₁] → h₁ → [W₂] → h₂ → [W₃] → ŷ → L              │
│                                                                     │
│   Backward: x ← [∂] ← h₁ ← [∂] ← h₂ ← [∂] ← ŷ ← ∂L/∂ŷ            │
│                                                                     │
│   REGLA DE LA CADENA:                                               │
│   ──────────────────                                                │
│   ∂L/∂W₂ = ∂L/∂ŷ × ∂ŷ/∂h₂ × ∂h₂/∂W₂                                 │
│                                                                     │
│   Para cada capa:                                                   │
│   1. Recibir gradiente de la capa siguiente (∂L/∂output)            │
│   2. Multiplicar por derivada de la activación (∂output/∂z)         │
│   3. Calcular gradientes de pesos: ∂L/∂W = (grad) × input            │
│   4. Pasar gradiente a capa anterior: ∂L/∂input = Wᵀ × (grad)        │
│                                                                     │
└─────────────────────────────────────────────────────────────────────┘
```

4.2 Funciones de Pérdida

```
def mse_loss(y_true: List[float], y_pred: List[float]) -> float:
    """Mean Squared Error loss."""
    return sum((yt - yp) ** 2 for yt, yp in zip(y_true, y_pred)) / len(y_true)


def mse_gradient(y_true: List[float], y_pred: List[float]) -> List[float]:
    """Gradient of MSE loss with respect to predictions."""
    n = len(y_true)
    return [2 * (yp - yt) / n for yt, yp in zip(y_true, y_pred)]


def binary_cross_entropy(y_true: List[float], y_pred: List[float]) -> float:
    """Binary cross-entropy loss."""
    eps = 1e-15
    loss = 0.0
    for yt, yp in zip(y_true, y_pred):
        yp = max(min(yp, 1 - eps), eps)  # Clip to avoid log(0)
        loss -= yt * math.log(yp) + (1 - yt) * math.log(1 - yp)
    return loss / len(y_true)


def bce_gradient(y_true: List[float], y_pred: List[float]) -> List[float]:
    """Gradient of binary cross-entropy."""
    eps = 1e-15
    return [
        ((yp - yt) / (yp * (1 - yp) + eps)) / len(y_true)
        for yt, yp in zip(y_true, y_pred)
    ]
```

```
def categorical_cross_entropy(y_true: List[int], y_pred: List[List[float]]) -> float:
    """Cross-entropy for multi-class classification.

    y_true: class indices
    y_pred: softmax probabilities
    """
    eps = 1e-15
    loss = 0.0
    for i, (true_class, pred_probs) in enumerate(zip(y_true, y_pred)):
        pred = max(pred_probs[true_class], eps)
        loss -= math.log(pred)
    return loss / len(y_true)
```

4.3 Training Loop Completo

```
def train_network(
    network: NeuralNetwork,
    X_train: List[List[float]],
    y_train: List[List[float]],
    epochs: int = 100,
    learning_rate: float = 0.01,
    batch_size: int = 32,
    verbose: bool = True
) -> List[float]:
    """Train neural network with mini-batch gradient descent.

    Returns list of losses per epoch.
    """
    n_samples = len(X_train)
    losses = []

    for epoch in range(epochs):
        # Shuffle data
        indices = list(range(n_samples))
        random.shuffle(indices)

        epoch_loss = 0.0
        n_batches = 0

        for start in range(0, n_samples, batch_size):
            end = min(start + batch_size, n_samples)
            batch_indices = indices[start:end]

            batch_loss = 0.0

            for idx in batch_indices:
                x = X_train[idx]
                y = y_train[idx]

                # Forward
                y_pred = network.forward(x)

                # Loss
                loss = mse_loss(y, y_pred)
                batch_loss += loss

                # Backward
                gradient = mse_gradient(y, y_pred)
                network.backward(gradient)
```

```
                # Update
                network.update(learning_rate)

            epoch_loss += batch_loss
            n_batches += 1

        avg_loss = epoch_loss / n_samples
        losses.append(avg_loss)

        if verbose and (epoch + 1) % 10 == 0:
            print(f"Epoch {epoch + 1}/{epochs}, Loss: {avg_loss:.6f}")

    return losses
```

---

5. Optimización y Regularización {#5-optimizacion}

5.1 Optimizadores

```
class SGD:
    """Stochastic Gradient Descent with momentum."""

    def __init__(self, learning_rate: float = 0.01, momentum: float = 0.0):
        self.lr = learning_rate
        self.momentum = momentum
        self.velocities: Dict = {}

    def update(self, param_id: str, param: List[float],
               gradient: List[float]) -> List[float]:
        """Update parameters."""
        if param_id not in self.velocities:
            self.velocities[param_id] = [0.0] * len(param)

        v = self.velocities[param_id]

        for i in range(len(param)):
            v[i] = self.momentum * v[i] - self.lr * gradient[i]
            param[i] += v[i]

        return param


class Adam:
    """Adam optimizer (simplified)."""

    def __init__(
        self,
        learning_rate: float = 0.001,
        beta1: float = 0.9,
        beta2: float = 0.999,
        epsilon: float = 1e-8
    ):
        self.lr = learning_rate
        self.beta1 = beta1
        self.beta2 = beta2
        self.epsilon = epsilon
        self.m: Dict = {}  # First moment
        self.v: Dict = {}  # Second moment
        self.t: int = 0    # Time step

    def update(self, param_id: str, param: List[float],
               gradient: List[float]) -> List[float]:
```

```
        """Update parameters using Adam."""
        self.t += 1

        if param_id not in self.m:
            self.m[param_id] = [0.0] * len(param)
            self.v[param_id] = [0.0] * len(param)

        m = self.m[param_id]
        v = self.v[param_id]

        for i in range(len(param)):
            # Update biased first moment
            m[i] = self.beta1 * m[i] + (1 - self.beta1) * gradient[i]

            # Update biased second moment
            v[i] = self.beta2 * v[i] + (1 - self.beta2) * gradient[i] ** 2

            # Bias correction
            m_hat = m[i] / (1 - self.beta1 ** self.t)
            v_hat = v[i] / (1 - self.beta2 ** self.t)

            # Update parameter
            param[i] -= self.lr * m_hat / (math.sqrt(v_hat) + self.epsilon)

        return param
```

5.2 Regularización

```
def l2_regularization(weights: List[List[float]], lambda_: float) -> float:
    """L2 regularization term: λ × Σ w²."""
    total = 0.0
    for layer_weights in weights:
        for row in layer_weights:
            total += sum(w ** 2 for w in row)
    return lambda_ * total


def l2_gradient(weight: float, lambda_: float) -> float:
    """Gradient of L2 regularization: 2λw."""
    return 2 * lambda_ * weight


def dropout(layer_output: List[float], keep_prob: float = 0.8,
            training: bool = True) -> List[float]:
    """Dropout regularization.

    Randomly zeros out neurons during training.
    Scales outputs during inference.
    """
    if not training:
        return layer_output

    result = []
    for val in layer_output:
        if random.random() < keep_prob:
            result.append(val / keep_prob)  # Inverted dropout
        else:
            result.append(0.0)

    return result
```

5.3 Batch Normalization (Concepto)

```
BATCH NORMALIZATION:
────────────────────

Normaliza las activaciones de cada capa:

1. Calcular μ y σ del batch
2. Normalizar: x̂ = (x - μ) / σ
3. Escalar y desplazar: y = γx̂ + β (parámetros aprendidos)

BENEFICIOS:
• Permite learning rates más altas
• Reduce dependencia de inicialización
• Actúa como regularizador
• Acelera el entrenamiento

NOTA: Comportamiento diferente en train vs inference
• Train: estadísticas del batch
• Inference: estadísticas acumuladas (running mean/var)
```

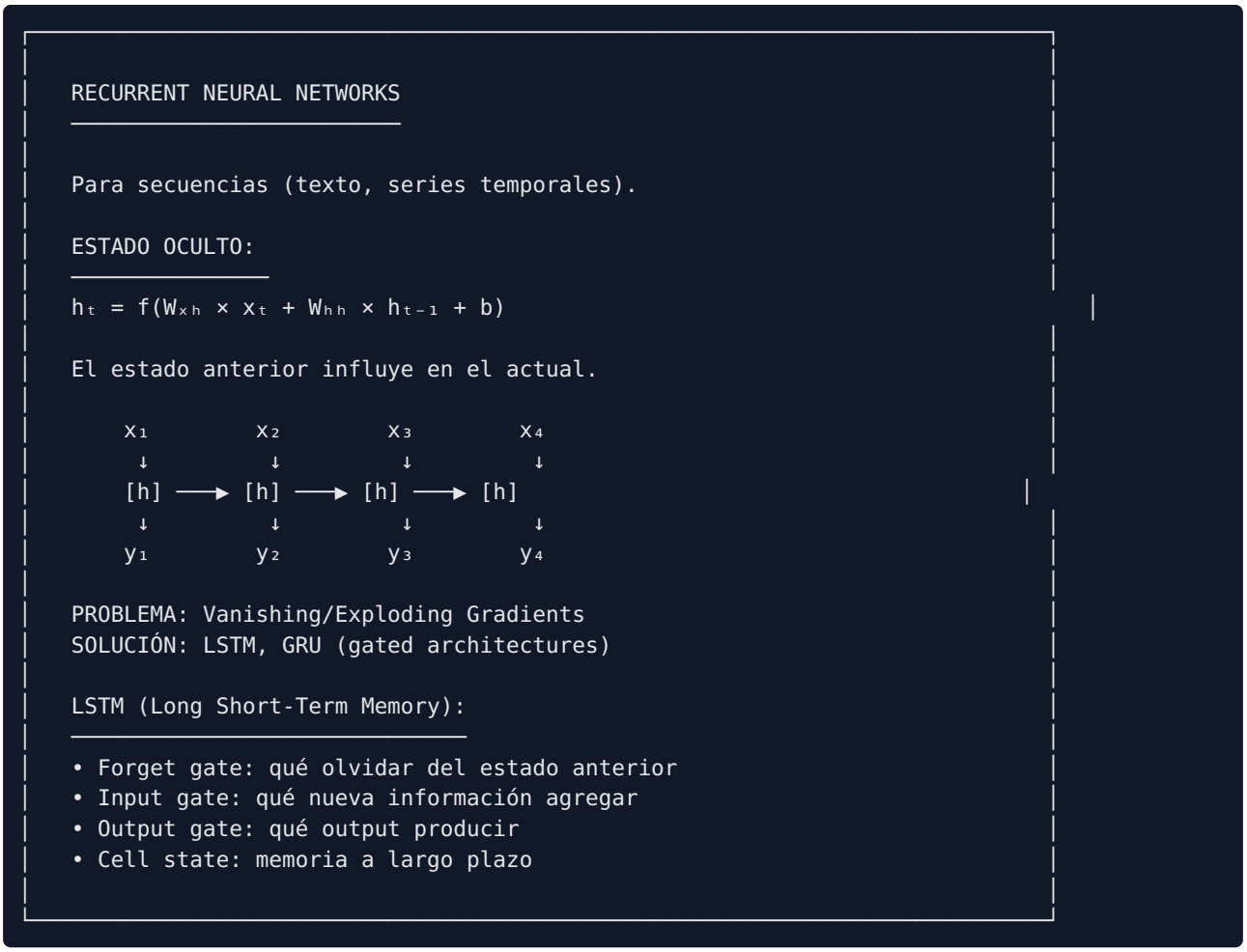
---

6. Arquitecturas Especiales (CNN, RNN) {#6-arquitecturas}

6.1 Convolutional Neural Networks (CNN)

```
┌─────────────────────────────────────────────────────────────────┐
│                                                                 │
│   CONVOLUTIONAL NEURAL NETWORKS                                 │
│   ─────────────────────────────                                 │
│                                                                 │
│   Para datos con estructura espacial (imágenes).                │
│                                                                 │
│   CONVOLUCIÓN:                                                  │
│   ────────────                                                  │
│   Filtro 3×3 deslizándose sobre la imagen:                      │
│                                                                 │
│   Input Image        Filter        Feature Map                  │
│   ┌─────────┐      ┌───┐         ┌───────┐                      │
│   │ 1 2 3 4 │   *  │a b│    =    │ . . . │                      │
│   │ 5 6 7 8 │      │c d│         │ . . . │                      │
│   │ 9 . . . │      └───┘         └───────┘                      │
│   └─────────┘                                                   │
│                                                                 │
│   output[i,j] = Σ input[i+k, j+l] × filter[k,l]                 │
│                                                                 │
│   POOLING:                                                      │
│   ────────                                                      │
│   Reduce dimensión espacial:                                    │
│   • Max Pooling: toma el máximo de cada región                  │
│   • Average Pooling: promedio de cada región                    │
│                                                                 │
│   ARQUITECTURA TÍPICA:                                          │
│   ────────────────────                                          │
│   [Conv → ReLU → Pool] × N → Flatten → [Dense] × M → Output     │
│                                                                 │
└─────────────────────────────────────────────────────────────────┘
```

6.2 Recurrent Neural Networks (RNN)



6.3 Convolution Simplificada

```
def convolve_2d(
    image: List[List[float]],
    kernel: List[List[float]]
) -> List[List[float]]:
    """2D convolution (no padding, stride=1).

    Simplified implementation for understanding.
    """
    h, w = len(image), len(image[0])
    kh, kw = len(kernel), len(kernel[0])

    output_h = h - kh + 1
    output_w = w - kw + 1

    output = [[0.0] * output_w for _ in range(output_h)]

    for i in range(output_h):
        for j in range(output_w):
            total = 0.0
            for ki in range(kh):
                for kj in range(kw):
                    total += image[i + ki][j + kj] * kernel[ki][kj]
            output[i][j] = total

    return output
```

```
def max_pool_2d(
    feature_map: List[List[float]],
    pool_size: int = 2
) -> List[List[float]]:
    """Max pooling with given pool size."""
    h, w = len(feature_map), len(feature_map[0])
    output_h = h // pool_size
    output_w = w // pool_size

    output = [[0.0] * output_w for _ in range(output_h)]

    for i in range(output_h):
        for j in range(output_w):
            max_val = float('-inf')
            for pi in range(pool_size):
                for pj in range(pool_size):
                    val = feature_map[i * pool_size + pi][j * pool_size + pj]
                    max_val = max(max_val, val)
            output[i][j] = max_val

    return output
```

⚠️ Mejores Prácticas

```
DEEP LEARNING BEST PRACTICES:
─────────────────────────────

DATOS:
• Más datos > modelo más complejo
• Augmentación para aumentar datos
• Normalización de inputs

ARQUITECTURA:
• Empezar simple, agregar complejidad si es necesario
• ReLU para capas ocultas
• Batch normalization después de capas densas

ENTRENAMIENTO:
• Adam optimizer por defecto
• Learning rate scheduling
• Early stopping
• Validación para detectar overfitting

DEBUGGING:
• Verificar que loss disminuye en train pequeño
• Graficar loss curves
• Monitorear gradientes (no vanishing/exploding)
```

🔧 Ejercicios Prácticos

Ejercicio 24.1: Perceptrón para AND/OR

Entrenar perceptrón en funciones lógicas simples.

Ejercicio 24.2: MLP para XOR

Red de 2 capas para resolver XOR (no linealmente separable).

Ejercicio 24.3: MNIST desde Cero

Clasificar dígitos con MLP implementado manualmente.

📚 Recursos Externos

Recurso	Tipo	Prioridad
Deep Learning Specialization	Curso	🔴 Obligatorio
3Blue1Brown: Neural Networks	Videos	🔴 Obligatorio
Neural Networks from Scratch	Libro	🟡 Recomendado

🧭 Navegación

← Anterior	Índice	Siguiente →
23_ML_NO_SUPERVISADO	00_INDICE	12_PROYECTO_INTEGRADOR

Módulo 10 - Proyecto Final

Guía MS in AI Pathway

DUQUEOM · 2025

Módulo 10 - Proyecto Final: ML Pipeline Completo

> 🎯 **Objetivo:** Construir un sistema de ML end-to-end que integre clasificación, clustering, análisis probabilístico y una red neuronal
> **Fase:** Integración | **Demuestra dominio de los 6 cursos del Pathway**

🧠 ¿Qué Estamos Construyendo?

El Proyecto Demuestra Dominio de las 2 Líneas del Pathway

```
┌─────────────────────────────────────────────────────────────┐
│                                                             │
│   PROYECTO: SISTEMA DE CLASIFICACIÓN DE TEXTO CON ML COMPLETO
│   ──────────────────────────────────────────────────────────
│                                                             │
│   LÍNEA 1: MACHINE LEARNING (3 créditos)                    │
│   ├── Clasificador Naive Bayes (ML Supervisado)             │
│   ├── Clustering K-Means de documentos (ML No Supervisado)  │
│   └── Red Neuronal MLP para clasificación (Deep Learning)   │
│                                                             │
│   LÍNEA 2: PROBABILIDAD Y ESTADÍSTICA (3 créditos)          │
│   ├── Análisis Bayesiano (Fundamentos de Probabilidad)      │
│   ├── Generador de texto con cadenas de Markov (MCMC)       │
│   └── Evaluación estadística con intervalos de confianza (Estimación)
│                                                             │
│   RESULTADO:                                                │
│   Un pipeline que clasifica documentos usando 3 enfoques diferentes,
│   compara su rendimiento estadísticamente, y genera texto sintético.
│                                                             │
└─────────────────────────────────────────────────────────────┘
```

Explicación Técnica Progresiva

Nivel 1 - Concepto:
Un sistema que clasifica textos automáticamente usando diferentes técnicas de ML.

Nivel 2 - Componentes:
- **Preprocesamiento:** Tokenización, TF-IDF vectorization
- **Naive Bayes:** Clasificador probabilístico (usa Teorema de Bayes)
- **K-Means:** Agrupa documentos similares (no supervisado)
- **MLP:** Red neuronal multicapa (deep learning)
- **Markov Chain:** Genera texto sintético
- **Evaluación estadística:** Intervalos de confianza, cross-validation

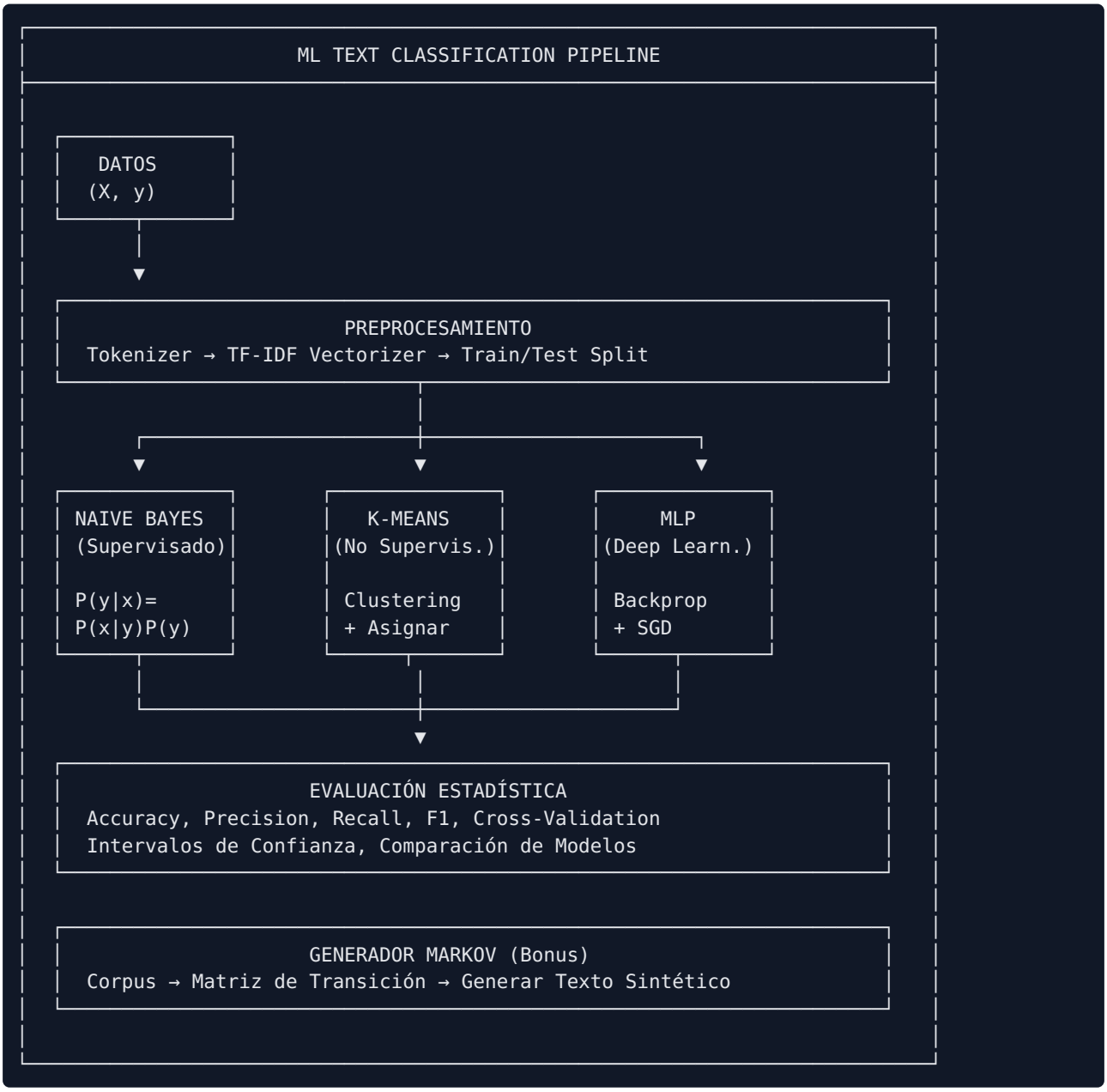
Nivel 3 - Flujo Completo:

```
ENTRENAMIENTO:
datos → preprocesar → vectorizar → entrenar modelos → evaluar

PREDICCIÓN:
nuevo texto → vectorizar → predecir con cada modelo → comparar

GENERACIÓN:
corpus → construir cadena de Markov → generar texto nuevo
```

```
ANÁLISIS:
resultados → intervalos de confianza → comparación estadística
```

Nivel 4 - Conexión con Pathway:
Componente	Curso del Pathway
Naive Bayes	Supervised Learning + Probability Foundations
K-Means	Unsupervised Algorithms
MLP	Introduction to Deep Learning
Markov Chain	Discrete-Time Markov Chains
Evaluación	Statistical Estimation

🏗️ Arquitectura Detallada



📁 Estructura de Archivos del Proyecto

```
ml-text-classifier/
├── src/
```

```
│   ├── __init__.py
│   │
│   ├── # PREPROCESAMIENTO (Módulos 04-06, 10-11)
│   ├── preprocessing.py         # Tokenizer, TF-IDF Vectorizer
│   ├── data_utils.py            # Train/test split, data loading
│   │
│   ├── # PROBABILIDAD (Módulos 19-21)
│   ├── probability.py           # Distribuciones, Bayes
│   ├── statistics.py            # MLE, intervalos de confianza
│   ├── markov.py                # Cadenas de Markov, generador de texto
│   │
│   ├── # MACHINE LEARNING (Módulos 22-23)
│   ├── naive_bayes.py           # Clasificador Naive Bayes
│   ├── kmeans.py                # Clustering K-Means
│   ├── evaluation.py            # Métricas, cross-validation
│   │
│   ├── # DEEP LEARNING (Módulo 24)
│   ├── neural_network.py        # MLP desde cero
│   ├── activations.py           # Sigmoid, ReLU, Softmax
│   ├── optimizers.py            # SGD, Adam
│   │
│   └── # INTEGRACIÓN
│   └── pipeline.py              # Pipeline completo que usa todo
│
├── tests/
│   ├── test_preprocessing.py
│   ├── test_probability.py
│   ├── test_naive_bayes.py
│   ├── test_kmeans.py
│   ├── test_neural_network.py
│   ├── test_markov.py
│   └── test_pipeline.py         # Tests de integración
│
├── data/
│   └── sample_dataset/          # Dataset de clasificación de texto
│       ├── train.csv
│       └── test.csv
│
├── notebooks/
│   └── demo.ipynb               # Demo interactivo del pipeline
│
├── docs/
│   ├── MODEL_COMPARISON.md      # Comparación estadística de modelos
│   └── PATHWAY_ALIGNMENT.md     # Cómo el proyecto cubre el Pathway
│
├── README.md                    # Documentación principal (inglés)
├── pyproject.toml
└── requirements-dev.txt
```

💻 Implementación Guiada: ML Pipeline

Paso 1: Pipeline Principal

```python
# src/pipeline.py
"""ML Pipeline integrating all components for the Pathway project."""

from typing import Dict, List, Tuple
from .preprocessing import TFIDFVectorizer, train_test_split
from .naive_bayes import NaiveBayesClassifier
from .kmeans import KMeans
from .neural_network import NeuralNetwork
```

```
from .evaluation import accuracy, precision, recall, f1_score, cross_validate
from .statistics import confidence_interval
from .markov import MarkovTextGenerator


class MLPipeline:
    """Complete ML pipeline demonstrating Pathway competencies.

    This class integrates:
    - Naive Bayes (Supervised Learning + Probability)
    - K-Means (Unsupervised Learning)
    - Neural Network (Deep Learning)
    - Statistical evaluation (Statistical Estimation)
    - Markov text generation (Markov Chains)

    Example:
        >>> pipeline = MLPipeline()
        >>> pipeline.load_data(texts, labels)
        >>> pipeline.train_all_models()
        >>> results = pipeline.compare_models()
        >>> print(results)
        {'naive_bayes': 0.85, 'kmeans': 0.72, 'neural_net': 0.88}
    """

    def __init__(self, n_classes: int = 2) -> None:
        """Initialize pipeline with empty models."""
        self.vectorizer = TFIDFVectorizer()
        self.naive_bayes = NaiveBayesClassifier()
        self.kmeans = KMeans(n_clusters=n_classes)
        self.neural_net: NeuralNetwork = None  # Initialized after vectorization
        self.markov_generator = MarkovTextGenerator()

        self.n_classes = n_classes
        self.X_train: List[List[float]] = []
        self.X_test: List[List[float]] = []
        self.y_train: List[int] = []
        self.y_test: List[int] = []

    def load_data(
        self,
        texts: List[str],
        labels: List[int],
        test_size: float = 0.2
    ) -> None:
        """Load and preprocess data.

        Args:
            texts: List of text documents.
            labels: List of class labels.
            test_size: Fraction for test set.
        """
        # Vectorize texts
        X = self.vectorizer.fit_transform(texts)

        # Split data
        self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(
            X, labels, test_size=test_size
        )

        # Initialize neural network with correct input size
        n_features = len(X[0]) if X else 0
        self.neural_net = NeuralNetwork(
            layer_sizes=[n_features, 64, 32, self.n_classes],
```

```
            activations=['relu', 'relu', 'softmax']
        )

        # Train Markov generator on all texts
        self.markov_generator.fit(texts)

    def train_all_models(self, verbose: bool = True) -> Dict[str, float]:
        """Train all three models and return training metrics.

        Returns:
            Dictionary with training accuracy for each model.
        """
        results = {}

        # 1. Train Naive Bayes (Módulo 19 + 22)
        if verbose:
            print("Training Naive Bayes...")
        self.naive_bayes.fit(self.X_train, self.y_train)
        nb_pred = self.naive_bayes.predict(self.X_train)
        results['naive_bayes_train'] = accuracy(self.y_train, nb_pred)

        # 2. Train K-Means (Módulo 23)
        if verbose:
            print("Training K-Means...")
        self.kmeans.fit(self.X_train)
        # Assign cluster labels to classes (majority voting)
        km_pred = self._kmeans_predict_with_labels(self.X_train, self.y_train)
        results['kmeans_train'] = accuracy(self.y_train, km_pred)

        # 3. Train Neural Network (Módulo 24)
        if verbose:
            print("Training Neural Network...")
        self.neural_net.fit(self.X_train, self.y_train, epochs=100)
        nn_pred = self.neural_net.predict(self.X_train)
        results['neural_net_train'] = accuracy(self.y_train, nn_pred)

        return results

    def evaluate_all_models(self) -> Dict[str, Dict[str, float]]:
        """Evaluate all models on test set.

        Returns metrics and confidence intervals (Módulo 20).
        """
        results = {}

        for name, model in [
            ('naive_bayes', self.naive_bayes),
            ('neural_net', self.neural_net)
        ]:
            y_pred = model.predict(self.X_test)

            acc = accuracy(self.y_test, y_pred)
            prec = precision(self.y_test, y_pred)
            rec = recall(self.y_test, y_pred)
            f1 = f1_score(self.y_test, y_pred)

            # Confidence interval for accuracy (Módulo 20)
            ci_low, ci_high = confidence_interval(
                successes=int(acc * len(self.y_test)),
                trials=len(self.y_test),
                confidence=0.95
            )
```

```
            results[name] = {
                'accuracy': acc,
                'precision': prec,
                'recall': rec,
                'f1': f1,
                'accuracy_ci_95': (ci_low, ci_high)
            }

        return results

    def cross_validate_models(self, k: int = 5) -> Dict[str, List[float]]:
        """K-fold cross-validation for all models (Módulo 20)."""
        X_all = self.X_train + self.X_test
        y_all = self.y_train + self.y_test

        return {
            'naive_bayes': cross_validate(NaiveBayesClassifier, X_all, y_all, k),
            'neural_net': cross_validate(NeuralNetwork, X_all, y_all, k)
        }

    def generate_text(self, seed: str = None, length: int = 50) -> str:
        """Generate synthetic text using Markov chain (Módulo 21)."""
        return self.markov_generator.generate(seed=seed, length=length)

    def _kmeans_predict_with_labels(
        self,
        X: List[List[float]],
        y: List[int]
    ) -> List[int]:
        """Assign class labels to K-Means clusters via majority voting."""
        cluster_labels = self.kmeans.predict(X)

        # Map cluster -> most common class
        from collections import Counter
        cluster_to_class = {}
        for k in range(self.n_classes):
            cluster_points = [y[i] for i in range(len(y)) if cluster_labels[i] == k]
            if cluster_points:
                cluster_to_class[k] = Counter(cluster_points).most_common(1)[0][0]
            else:
                cluster_to_class[k] = 0

        return [cluster_to_class[c] for c in cluster_labels]
```

Paso 2: Generador de Texto Markov (Módulo 21)

```
# src/markov.py
"""Markov Chain text generator - demonstrates Discrete-Time Markov Chains."""

from typing import Dict, List, Optional
import random
from collections import defaultdict


class MarkovTextGenerator:
    """Text generator using Markov Chains.

    Demonstrates:
    - Discrete-time Markov Chains (Pathway course)
    - Transition probability matrices
    - Random sampling from distributions
```

```
    Example:
        >>> generator = MarkovTextGenerator(order=2)
        >>> generator.fit(["The quick brown fox", "The quick dog"])
        >>> generator.generate(seed="The quick", length=10)
        "The quick brown fox jumps..."
    """

    def __init__(self, order: int = 2) -> None:
        """Initialize with n-gram order."""
        self.order = order
        self.transitions: Dict[tuple, Dict[str, int]] = defaultdict(lambda:
defaultdict(int))
        self.start_states: List[tuple] = []

    def fit(self, texts: List[str]) -> 'MarkovTextGenerator':
        """Build transition matrix from corpus.

        For each n-gram, count how often each word follows it.
        """
        for text in texts:
            words = text.split()
            if len(words) < self.order + 1:
                continue

            # Store starting states
            self.start_states.append(tuple(words[:self.order]))

            # Build transitions
            for i in range(len(words) - self.order):
                state = tuple(words[i:i + self.order])
                next_word = words[i + self.order]
                self.transitions[state][next_word] += 1

        return self

    def _sample_next(self, state: tuple) -> Optional[str]:
        """Sample next word given current state.

        Uses counts as unnormalized probabilities.
        """
        if state not in self.transitions:
            return None

        choices = self.transitions[state]
        total = sum(choices.values())

        r = random.random() * total
        cumulative = 0

        for word, count in choices.items():
            cumulative += count
            if r <= cumulative:
                return word

        return list(choices.keys())[-1]

    def generate(self, seed: str = None, length: int = 50) -> str:
        """Generate text using the Markov chain.

        Args:
            seed: Starting words (must match order)
            length: Number of words to generate
```

```
        Returns:
            Generated text string
        """
        if seed:
            words = seed.split()
            if len(words) < self.order:
                # Pad with random start
                start = random.choice(self.start_states) if self.start_states else ()
                words = list(start)[:self.order - len(words)] + words
            state = tuple(words[-self.order:])
        else:
            if not self.start_states:
                return ""
            state = random.choice(self.start_states)
            words = list(state)

        for _ in range(length):
            next_word = self._sample_next(state)
            if next_word is None:
                break
            words.append(next_word)
            state = tuple(words[-self.order:])

        return " ".join(words)
```

📊 Análisis de Complejidad Completo

Template COMPLEXITY_ANALYSIS.md

```
# Complexity Analysis - ML Text Classification Pipeline

## Overview

This document analyzes the time and space complexity of all models
in the ML Pipeline for the MS in AI Pathway project.

## Notation

- N = number of documents
- T = average tokens per document
- V = vocabulary size (unique terms)
- Q = query length (tokens)
- R = number of results

## Component Analysis

### 1. Tokenizer.tokenize(text)

**Time:** O(T)
- Split text: O(T)
- Lowercase: O(T)
- Filter stop words: O(T) with set lookup

**Space:** O(T) for output list

### 2. InvertedIndex.add_document(doc_id, tokens)

**Time:** O(T)
- For each token: O(1) dict access + O(1) set add
- Total: O(T)
```

```
**Space:** O(V) for index + O(N) doc_ids per term

### 3. InvertedIndex.search_or(terms)

**Time:** O(Q × avg_docs_per_term)
- For each query term: O(1) lookup
- Union of sets: O(total matching docs)

### 4. TFIDFVectorizer.fit_transform(corpus)

**Time:** O(N × T + V)
- Build vocabulary: O(N × T)
- Compute IDF: O(V)
- Transform each doc: O(N × V)

**Space:** O(N × V) for document vectors

### 5. cosine_similarity(v1, v2)

**Time:** O(V)
- Dot product: O(V)
- Magnitudes: O(V) each
- Division: O(1)

### 6. quicksort(results)

**Time:** O(R log R) average, O(R²) worst case
**Space:** O(log R) for recursion stack

### 7. SearchEngine.build_index()

**Time:** O(N × T + N × V)
- Tokenize all docs: O(N × T)
- Build inverted index: O(N × T)
- Build TF-IDF vectors: O(N × T + N × V)

**Space:** O(V + N × V)
- Inverted index: O(V)
- Document vectors: O(N × V)

### 8. SearchEngine.search(query)

**Time:** O(Q + R × V + R log R)
- Tokenize query: O(Q)
- Find candidates: O(Q)
- Transform query: O(V)
- Calculate similarities: O(R × V)
- Sort results: O(R log R)

**Space:** O(V + R)
- Query vector: O(V)
- Results list: O(R)

## Summary Table

| Operation | Time | Space |
|-----------|------|-------|
| add_document | O(1) | O(T) |
| build_index | O(N×T + N×V) | O(V + N×V) |
| search | O(Q + R×V + R log R) | O(V + R) |

## Bottlenecks and Optimizations
```

```
1. **TF-IDF vectors are dense** → Could use sparse representation
2. **Similarity calculated for all candidates** → Could use inverted index scores
3. **QuickSort worst case** → Using random pivot mitigates this
```

⚠️ Errores Comunes y Soluciones

Error 1: Olvidar llamar build_index()

```
# ❌ Error: RuntimeError
engine = SearchEngine()
engine.add_document(1, "Title", "Content")
results = engine.search("query")  # ¡No se indexó!

# ✅ Correcto
engine = SearchEngine()
engine.add_document(1, "Title", "Content")
engine.build_index()  # ¡Importante!
results = engine.search("query")
```

Error 2: No manejar queries vacías

```
# ❌ Puede causar errores
def search(self, query):
    tokens = self.tokenizer.tokenize(query)
    # Si query="", tokens=[] y query_vector tiene problemas

# ✅ Manejar caso vacío
def search(self, query):
    tokens = self.tokenizer.tokenize(query)
    if not tokens:
        return []  # Retornar lista vacía
```

Error 3: Modificar documento después de indexar

```
# ❌ El índice queda desactualizado
engine.add_document(1, "Title", "Python tutorial")
engine.build_index()
engine.corpus.get(1).content = "Java tutorial"  # ¡Índice no actualizado!

# ✅ Reconstruir índice después de modificaciones
engine.add_document(2, "Title2", "New content")
engine.build_index()  # Reconstruir
```

Error 4: No normalizar texto consistentemente

```
# ❌ "Python" vs "python" son diferentes
index.search("Python")  # Encuentra
index.search("python")  # No encuentra

# ✅ Normalizar siempre en tokenizer
def tokenize(self, text):
    return text.lower().split()  # Siempre minúsculas
```

💡 Recomendaciones Profesionales

1. Testing

```
# Mínimo: tests unitarios para cada componente
pytest tests/ -v --cov=src --cov-report=term-missing
# Objetivo: >80% coverage
```

2. Type Hints

```
# Todas las funciones deben tener type hints
def search(self, query: str, top_k: int = 10) -> list[SearchResult]:
```

3. Docstrings

```
# Google style docstrings para todas las funciones públicas
def function(param: Type) -> ReturnType:
    """One-line description.

    Longer description if needed.

    Args:
        param: Description of parameter.

    Returns:
        Description of return value.

    Raises:
        ErrorType: When this error occurs.

    Example:
        >>> function(value)
        expected_result
    """
```

4. Configuración de Herramientas

```
# pyproject.toml
[tool.mypy]
strict = true
python_version = "3.11"

[tool.ruff]
line-length = 88
select = ["E", "F", "W", "I", "N", "UP", "B"]

[tool.pytest.ini_options]
testpaths = ["tests"]
addopts = "-v --cov=src"
```

📋 Checklist de Entrega (100 puntos)

Línea 1: Machine Learning (40 pts)

- [ ] `NaiveBayesClassifier` desde cero (10 pts) - Módulo 19 + 22
- [ ] `KMeans` clustering desde cero (10 pts) - Módulo 23
- [ ] `NeuralNetwork` MLP con backprop (15 pts) - Módulo 24
- [ ] Preprocesamiento TF-IDF (5 pts) - Módulo 11

Línea 2: Probabilidad y Estadística (30 pts)

- [ ] Análisis Bayesiano en Naive Bayes (10 pts) - Módulo 19
- [ ] `MarkovTextGenerator` funcional (10 pts) - Módulo 21
- [ ] Evaluación con intervalos de confianza (5 pts) - Módulo 20
- [ ] Cross-validation implementado (5 pts) - Módulo 20

Testing y Documentación (20 pts)

- [ ] Tests unitarios para cada modelo (10 pts)
- [ ] README.md profesional en inglés (5 pts)
- [ ] `MODEL_COMPARISON.md` con análisis estadístico (5 pts)

Funcionalidad (10 pts)

- [ ] Pipeline entrena y predice correctamente (5 pts)
- [ ] Demo interactivo funcional (5 pts)

🏗️ Arquitectura Final

```
MLPipeline (Proyecto Integrador)
    │
    ├── Preprocesamiento
    │   ├── Tokenizer (text → tokens)
    │   └── TFIDFVectorizer (tokens → vectors)
    │
    ├── LÍNEA 1: Machine Learning
    │   ├── NaiveBayesClassifier (Supervisado + Bayes)
    │   ├── KMeans (No Supervisado)
    │   └── NeuralNetwork (Deep Learning)
    │
    ├── LÍNEA 2: Probabilidad/Estadística
    │   ├── MarkovTextGenerator (Cadenas de Markov)
    │   ├── confidence_interval() (Estimación)
    │   └── cross_validate() (Evaluación)
    │
    └── Evaluación
        ├── accuracy, precision, recall, f1
        └── Comparación estadística de modelos
```

📊 Análisis de Complejidad Requerido

Documenta en `MODEL_COMPARISON.md`:

Modelo	Train	Predict	Space
`NaiveBayes.fit()`	O(N × V)	O(V × C)	O(V × C)
`KMeans.fit()`	O(N × K × V × I)	O(K × V)	O(K × V)
`NeuralNetwork.fit()`	O(E × N × L)	O(N × L)	O(L)
`MarkovGenerator.fit()`	O(T)	O(L)	$O(V^2)$

Donde: N=samples, V=features, C=classes, K=clusters, I=iterations, E=epochs, L=layers, T=tokens

---

📝 Template README.md

```
# ML Text Classification Pipeline

A complete ML pipeline built from scratch in pure Python for the MS in AI Pathway.

## Pathway Alignment

This project demonstrates competency in both Pathway lines:

### Machine Learning (3 credits)
- ✅ Naive Bayes Classifier (Supervised Learning)
- ✅ K-Means Clustering (Unsupervised Learning)
- ✅ Neural Network with Backpropagation (Deep Learning)

### Probability & Statistics (3 credits)
- ✅ Bayesian Analysis in Naive Bayes (Probability Foundations)
- ✅ Markov Chain Text Generator (Discrete-Time Markov Chains)
- ✅ Confidence Intervals & Cross-Validation (Statistical Estimation)

## Features
- All models implemented from scratch (no sklearn, no pytorch)
- TF-IDF text vectorization
- Statistical model comparison with confidence intervals
- Markov chain text generation

## Installation
\`\`\`bash
git clone <repo>
cd ml-text-classifier
python -m venv venv
source venv/bin/activate
\`\`\`

## Usage
\`\`\`python
from src.pipeline import MLPipeline

pipeline = MLPipeline(n_classes=2)
pipeline.load_data(texts, labels)
pipeline.train_all_models()
results = pipeline.evaluate_all_models()
print(results)
# {'naive_bayes': {'accuracy': 0.85, 'accuracy_ci_95': (0.78, 0.92)}, ...}

# Generate synthetic text
generated = pipeline.generate_text(seed="The model", length=20)
\`\`\`

## Model Comparison
See [MODEL_COMPARISON.md](#mod_MODEL_COMPARISON)

## Testing
\`\`\`bash
python -m pytest tests/ -v --cov=src
\`\`\`
```

✅ Criterios de Aprobación

Puntuación	Nivel	Significado
90-100	🏆 Listo para Pathway	Dominas ambas líneas
75-89	✅ Buen nivel	Reforzar gaps menores
60-74	⚠️ Necesita trabajo	Revisar módulos 19-24
<60	❌ Insuficiente	Volver a estudiar fundamentos

🎯 Verificación de Competencias del Pathway

Curso del Pathway	¿Cubierto?	Evidencia en el Proyecto
ML: Supervised Learning	✅	NaiveBayesClassifier, evaluación
ML: Unsupervised Algorithms	✅	KMeans clustering
ML: Deep Learning	✅	NeuralNetwork con backprop
Prob: Foundations	✅	Bayes en clasificador, distribuciones
Prob: Markov Chains	✅	MarkovTextGenerator
Prob: Statistical Estimation	✅	Intervalos de confianza, cross-val

---

🔗 Navegación

← Anterior	Índice
24_INTRO_DEEP_LEARNING	00_INDICE

Ejercicios Prácticos

Guía MS in AI Pathway

DUQUEOM · 2025

📝 Ejercicios Prácticos

Ejercicios organizados por módulo con dificultad progresiva.

Índice de Ejercicios

Módulo	Tema	Dificultad	# Ejercicios
01	Python Profesional	🟢 Básico	4
02	OOP	🟢 Básico	5
03	Lógica y Big O	🟡 Intermedio	3
04	Arrays y Strings	🟢 Básico	3
05	Hash Maps	🟡 Intermedio	3
06	Índice Invertido	🟡 Intermedio	3
07	Recursión	🟡 Intermedio	3
08	Sorting	🔴 Avanzado	3
09	Binary Search	🟡 Intermedio	3
10	Álgebra Lineal	🟡 Intermedio	3
11	TF-IDF	🔴 Avanzado	3
13	Linked Lists, Stacks, Queues	🟡 Intermedio	4
14	Trees y BST	🔴 Avanzado	5
15	Graphs, BFS, DFS	🔴 Avanzado	5
16	Dynamic Programming	🔴 Avanzado	5
17	Greedy Algorithms	🟡 Intermedio	4
18	Heaps	🔴 Avanzado	4
19	**Probabilidad** ⭐ PATHWAY	🔴 Avanzado	5
20	**Estadística Inferencial** ⭐ PATHWAY	🔴 Avanzado	5
21	**Markov y Monte Carlo** ⭐ PATHWAY	🔴 Avanzado	5
22	**ML Supervisado** ⭐ PATHWAY	🔴 Avanzado	5
23	**ML No Supervisado** ⭐ PATHWAY	🔴 Avanzado	5
24	**Deep Learning** ⭐ PATHWAY	🔴 Avanzado	5

Módulo 01: Python Profesional

Ejercicio 1.1: Type Hints Básicos

Objetivo: Agregar type hints a funciones existentes.

```
# Agregar type hints a estas funciones:

def clean_text(text):
    return text.lower().strip()

def count_words(text):
```

```
    return len(text.split())

def get_unique_words(words):
    return list(set(words))
```

Ejercicio 1.2: Función Pura

Objetivo: Convertir función impura a pura.

```
# Convertir a función pura (sin modificar estado externo):
results = []

def add_to_results(item):
    results.append(item)
    return len(results)
```

Ejercicio 1.3: Docstrings

Objetivo: Escribir docstrings estilo Google.

```
# Agregar docstring completo con Args, Returns, Example:
def tokenize(text, min_length=2):
    words = text.lower().split()
    return [w for w in words if len(w) >= min_length]
```

Ejercicio 1.4: Configurar Linters

Objetivo: Crear `pyproject.toml` con mypy y ruff configurados.

Módulo 02: OOP

Ejercicio 2.1: Clase Document Básica

Objetivo: Crear clase Document con `__init__`, atributos tipados.

```
# Crear clase Document con:
# - doc_id: int
# - content: str
# - tokens: list[str] (vacía inicialmente)
# - Método tokenize() que llena tokens
```

Ejercicio 2.2: Métodos Mágicos

Objetivo: Implementar `__repr__`, `__str__`, `__eq__`, `__len__`.

Ejercicio 2.3: Properties

Objetivo: Agregar validación con properties para `doc_id` (>= 0) y `content` (no vacío).

Ejercicio 2.4: Clase Corpus

Objetivo: Crear Corpus que contenga Documents con métodos add, get, remove.

Ejercicio 2.5: SOLID

Objetivo: Refactorizar una clase "Dios" que hace todo en clases separadas.

Módulo 03: Lógica y Big O

Ejercicio 3.1: Stop Words como Set

Objetivo: Implementar filtrado de stop words usando set para O(1) lookup.

```
# Dado:
stop_words_list = ["the", "a", "an", "is", "are"]
tokens = ["the", "quick", "brown", "fox", "is", "fast"]

# Implementar filter_stopwords() que sea O(n) no O(n×m)
```

Ejercicio 3.2: Operaciones de Conjuntos

Objetivo: Implementar búsqueda AND y OR usando set operations.

Ejercicio 3.3: Analizar Complejidad

Objetivo: Determinar Big O de 5 fragmentos de código dados.

```
# ¿Cuál es la complejidad de cada uno?

# A
for i in range(n):
    print(i)

# B
for i in range(n):
    for j in range(n):
        print(i, j)

# C
for i in range(n):
    for j in range(i):
        print(i, j)

# D
i = n
while i > 0:
    print(i)
    i = i // 2

# E
def recursive(n):
    if n <= 1:
        return
    recursive(n - 1)
    recursive(n - 1)
```

Módulo 04: Arrays y Strings

Ejercicio 4.1: Manipulación de Listas

Objetivo: Implementar rotate_left(list, k) sin usar slicing.

Ejercicio 4.2: Tokenizador

Objetivo: Implementar tokenizador completo con:
- Eliminar puntuación

- Convertir a minúsculas
- Filtrar por longitud mínima

Ejercicio 4.3: Análisis de Complejidad

Objetivo: Comparar dos implementaciones de reverse y explicar cuál es mejor.

Módulo 05: Hash Maps

Ejercicio 5.1: Contador de Frecuencias

Objetivo: Implementar word_frequencies(tokens) → dict[str, int].

Ejercicio 5.2: Benchmark List vs Set

Objetivo: Escribir script que mide tiempo de búsqueda en list vs set.

Ejercicio 5.3: Term-Document Map

Objetivo: Construir diccionario term → set[doc_id].

Módulo 06: Índice Invertido

Ejercicio 6.1: Índice Básico

Objetivo: Implementar InvertedIndex con add_document() y search().

Ejercicio 6.2: Búsqueda AND/OR

Objetivo: Agregar search_and() y search_or() al índice.

Ejercicio 6.3: Índice con Frecuencias

Objetivo: Modificar índice para guardar frecuencia de cada término por documento.

Módulo 07: Recursión

Ejercicio 7.1: Factorial y Fibonacci

Objetivo: Implementar ambos recursivamente con casos base correctos.

Ejercicio 7.2: Suma y Máximo

Objetivo: Implementar sum_list() y find_max() recursivamente.

Ejercicio 7.3: Merge de Listas

Objetivo: Implementar merge(list1, list2) que fusiona dos listas ordenadas.

Módulo 08: Sorting

Ejercicio 8.1: QuickSort

Objetivo: Implementar quicksort() con partición Lomuto.

Ejercicio 8.2: MergeSort

Objetivo: Implementar mergesort() con función merge() auxiliar.

Ejercicio 8.3: Ordenar por Score

Objetivo: Ordenar lista de (doc_id, score) por score descendente usando tu quicksort.

Módulo 09: Binary Search

Ejercicio 9.1: Binary Search Básica

Objetivo: Implementar binary_search() iterativo sin errores off-by-one.

Ejercicio 9.2: Primera y Última Ocurrencia

Objetivo: Implementar find_first() y find_last() para elementos repetidos.

Ejercicio 9.3: Búsqueda de Umbral

Objetivo: Encontrar todos los documentos con score >= threshold en lista ordenada.

Módulo 10: Álgebra Lineal

Ejercicio 10.1: Operaciones Vectoriales

Objetivo: Implementar add_vectors(), subtract_vectors(), scalar_multiply().

Ejercicio 10.2: Producto Punto y Norma

Objetivo: Implementar dot_product() y magnitude().

Ejercicio 10.3: Similitud de Coseno

Objetivo: Implementar cosine_similarity() usando las funciones anteriores.

Módulo 11: TF-IDF

Ejercicio 11.1: Term Frequency

Objetivo: Implementar compute_tf(term, document).

Ejercicio 11.2: Inverse Document Frequency

Objetivo: Implementar compute_idf(term, corpus).

Ejercicio 11.3: Sistema de Ranking

Objetivo: Implementar rank_documents() que ordena por similitud de coseno.

Módulo 13: Linked Lists, Stacks, Queues

Ejercicio 13.1: Implementar Stack

Objetivo: Crear clase Stack con push, pop, peek, is_empty.

Ejercicio 13.2: Paréntesis Balanceados

Objetivo: Verificar si string tiene paréntesis `()[]{}` balanceados usando Stack.

Ejercicio 13.3: Implementar Queue

Objetivo: Crear clase Queue con enqueue, dequeue usando deque.

Ejercicio 13.4: Reverse Linked List

Objetivo: Invertir una linked list iterativamente.

Módulo 14: Trees y BST

Ejercicio 14.1: Implementar BST

Objetivo: Crear clase BST con insert y search.

Ejercicio 14.2: Tree Traversals

Objetivo: Implementar inorder, preorder, postorder (recursivo e iterativo).

Ejercicio 14.3: Validar BST

Objetivo: Verificar si un árbol cumple la propiedad BST.

Ejercicio 14.4: Altura del Árbol

Objetivo: Calcular altura de un árbol binario.

Ejercicio 14.5: Level Order Traversal

Objetivo: Recorrer árbol por niveles usando Queue.

Módulo 15: Graphs, BFS, DFS

Ejercicio 15.1: Implementar Graph

Objetivo: Crear clase Graph con adjacency list.

Ejercicio 15.2: BFS

Objetivo: Implementar Breadth-First Search.

Ejercicio 15.3: DFS

Objetivo: Implementar Depth-First Search (recursivo e iterativo).

Ejercicio 15.4: Shortest Path (Unweighted)

Objetivo: Encontrar camino más corto usando BFS.

Ejercicio 15.5: Detectar Ciclo

Objetivo: Detectar si un grafo tiene ciclo usando DFS.

Módulo 16: Dynamic Programming

Ejercicio 16.1: Fibonacci con DP

Objetivo: Implementar con memoization y tabulation.

Ejercicio 16.2: Climbing Stairs

Objetivo: Contar formas de subir n escaleras (1 o 2 pasos).

Ejercicio 16.3: Coin Change

Objetivo: Mínimas monedas para un amount.

Ejercicio 16.4: Longest Common Subsequence

Objetivo: Encontrar LCS de dos strings.

Ejercicio 16.5: 0/1 Knapsack

Objetivo: Maximizar valor con capacidad limitada.

Módulo 17: Greedy Algorithms

Ejercicio 17.1: Activity Selection

Objetivo: Seleccionar máximas actividades no superpuestas.

Ejercicio 17.2: Fractional Knapsack

Objetivo: Maximizar valor tomando fracciones de items.

Ejercicio 17.3: Jump Game

Objetivo: Determinar si puedes llegar al final del array.

Ejercicio 17.4: Minimum Meeting Rooms

Objetivo: Mínimas salas para todas las reuniones.

Módulo 18: Heaps

Ejercicio 18.1: Implementar MinHeap

Objetivo: Crear clase MinHeap con push, pop, peek.

Ejercicio 18.2: K Largest Elements

Objetivo: Encontrar los k elementos más grandes.

Ejercicio 18.3: Top K Frequent

Objetivo: Encontrar los k elementos más frecuentes.

Ejercicio 18.4: Merge K Sorted Lists

Objetivo: Fusionar k listas ordenadas.

Módulo 19: Fundamentos de Probabilidad ⭐ PATHWAY

Ejercicio 19.1: Teorema de Bayes

Objetivo: Implementar función que calcule probabilidad posterior usando Bayes.

```
# Dado:
# - P(enfermedad) = 0.001 (prior)
# - P(test_positivo | enfermedad) = 0.99 (sensitivity)
# - P(test_positivo | no_enfermedad) = 0.05 (false positive rate)
#
# Calcular: P(enfermedad | test_positivo)
def bayes_posterior(prior, likelihood, false_positive_rate):
    # Tu implementación
    pass
```

Ejercicio 19.2: Distribución Normal

Objetivo: Implementar PDF de distribución normal sin scipy.

Ejercicio 19.3: Esperanza y Varianza

Objetivo: Calcular E[X] y Var(X) de una distribución discreta.

Ejercicio 19.4: Naive Bayes Simple

Objetivo: Implementar clasificador Naive Bayes para spam detection.

Ejercicio 19.5: Sampling de Distribución

Objetivo: Implementar muestreo de distribución categórica.

Módulo 20: Estadística Inferencial ⭐ PATHWAY

Ejercicio 20.1: Maximum Likelihood Estimation

Objetivo: Estimar parámetro de distribución Bernoulli usando MLE.

```
# Dado un conjunto de observaciones [0, 1, 1, 1, 0, 1, 0, 1]
# Encontrar el parámetro p que maximiza la likelihood
def mle_bernoulli(observations):
    # Tu implementación
    pass
```

Ejercicio 20.2: Intervalo de Confianza

Objetivo: Calcular intervalo de confianza al 95% para media muestral.

Ejercicio 20.3: Z-Test

Objetivo: Implementar test de hipótesis Z-test.

Ejercicio 20.4: Bootstrap

Objetivo: Implementar bootstrap para estimar varianza de estimador.

Ejercicio 20.5: Cross-Validation

Objetivo: Implementar k-fold cross-validation desde cero.

Módulo 21: Cadenas de Markov y Monte Carlo ⭐ PATHWAY

Ejercicio 21.1: Matriz de Transición

Objetivo: Construir matriz de transición de cadena de Markov.

```
# Dada una secuencia de estados: ["A", "B", "A", "A", "B", "C", "A"]
# Construir matriz de transición P[i][j] = P(next=j | current=i)
def build_transition_matrix(sequence):
    # Tu implementación
    pass
```

Ejercicio 21.2: Distribución Estacionaria

Objetivo: Calcular distribución estacionaria π tal que π = πP.

Ejercicio 21.3: PageRank Simple

Objetivo: Implementar algoritmo PageRank usando power iteration.

Ejercicio 21.4: Monte Carlo π

Objetivo: Estimar π usando Monte Carlo (puntos en círculo/cuadrado).

Ejercicio 21.5: Metropolis-Hastings

Objetivo: Implementar sampler Metropolis-Hastings para distribución normal.

Módulo 22: ML Supervisado ⭐ PATHWAY

Ejercicio 22.1: Regresión Lineal

Objetivo: Implementar regresión lineal con gradient descent.

```
# Implementar clase LinearRegression con fit() y predict()
# Sin usar sklearn, solo Python puro
class LinearRegression:
    def fit(self, X, y, lr=0.01, epochs=1000):
        # Tu implementación
        pass

    def predict(self, X):
        pass
```

Ejercicio 22.2: Regresión Logística

Objetivo: Implementar clasificador logístico con sigmoid y cross-entropy.

Ejercicio 22.3: Árbol de Decisión

Objetivo: Implementar árbol de decisión con information gain.

Ejercicio 22.4: K-Nearest Neighbors

Objetivo: Implementar KNN con distancia euclidiana.

Ejercicio 22.5: Métricas de Evaluación

Objetivo: Implementar accuracy, precision, recall, F1 desde cero.

Módulo 23: ML No Supervisado ⭐ PATHWAY

Ejercicio 23.1: K-Means

Objetivo: Implementar K-Means clustering completo.

```
# Implementar clase KMeans con fit() y predict()
class KMeans:
    def __init__(self, n_clusters=3, max_iters=100):
        pass

    def fit(self, X):
        # 1. Inicializar centroides
        # 2. Asignar puntos al centroide más cercano
        # 3. Actualizar centroides
        # 4. Repetir hasta convergencia
        pass
```

Ejercicio 23.2: Elbow Method

Objetivo: Implementar elbow method para selección de k.

Ejercicio 23.3: Silhouette Score

Objetivo: Implementar cálculo de silhouette score.

Ejercicio 23.4: PCA desde Cero

Objetivo: Implementar PCA calculando eigenvectors de covarianza.

Ejercicio 23.5: Detección de Anomalías

Objetivo: Implementar detector de anomalías basado en distancia.

Módulo 24: Deep Learning ⭐ PATHWAY

Ejercicio 24.1: Perceptrón

Objetivo: Implementar perceptrón simple con regla de aprendizaje.

```
# Implementar perceptrón que aprenda función AND
class Perceptron:
    def __init__(self, n_inputs):
        pass

    def predict(self, x):
        pass
```

```
    def train(self, X, y, epochs=100):
        pass
```

Ejercicio 24.2: Funciones de Activación

Objetivo: Implementar sigmoid, ReLU, tanh, softmax con sus derivadas.

Ejercicio 24.3: MLP Forward Pass

Objetivo: Implementar forward pass de MLP de 2 capas.

Ejercicio 24.4: Backpropagation

Objetivo: Implementar backprop para MLP que resuelva XOR.

Ejercicio 24.5: Mini-batch SGD

Objetivo: Implementar entrenamiento con mini-batches y learning rate decay.

📚 Soluciones

Ver EJERCICIOS_SOLUCIONES.md para soluciones detalladas.

💡 Consejos

1. **Intenta primero:** No mires las soluciones hasta intentar al menos 30 minutos.
2. **Escribe tests:** Antes de implementar, escribe casos de prueba.
3. **Analiza complejidad:** Para cada solución, determina su Big O.
4. **Compara:** Después de resolver, compara con la solución oficial.
5. **Sin sklearn:** Implementa TODO desde cero, sin librerías de ML.
6. **Conexión con Pathway:** Cada ejercicio prepara para un concepto del Pathway.

Glosario Técnico

Guía MS in AI Pathway

DUQUEOM · 2025

📖 Glosario Técnico

Definiciones A-Z de términos usados en la guía.

A

Adjacency List

Definición: Representación de grafo donde cada vértice tiene lista de vecinos.
Espacio: O(V + E)
Uso: Grafos sparse (pocos edges).

Adjacency Matrix

Definición: Matriz donde M[i][j] = 1 si hay edge de i a j.
Espacio: $O(V^2)$
Uso: Grafos dense, verificar edge en O(1).

Algoritmo

Definición: Secuencia finita de pasos para resolver un problema.
Analogía: Una receta de cocina: ingredientes (input) → pasos → plato (output).

Amortizado

Definición: Complejidad promedio sobre muchas operaciones.
Ejemplo: `list.append()` es O(1) amortizado aunque ocasionalmente sea O(n).

Array

Definición: Estructura de datos con elementos en posiciones contiguas de memoria.
En Python: Las `list` son arrays dinámicos.

B

Big O Notation

Definición: Notación para describir el crecimiento del tiempo/espacio con el tamaño de entrada.
Común: O(1) < O(log n) < O(n) < O(n log n) < $O(n^2)$ < O(2^n)

BFS (Breadth-First Search)

Definición: Algoritmo de recorrido de grafos que explora por niveles.
Estructura: Usa Queue (FIFO).
Uso: Shortest path en grafos no ponderados.
Complejidad: O(V + E)

Binary Search

Definición: Algoritmo que encuentra un elemento en lista ordenada dividiendo el espacio a la mitad.
Complejidad: O(log n)
Requisito: Lista debe estar ordenada.

Binary Search Tree (BST)

Definición: Árbol binario donde left < root < right para cada nodo.
Operaciones: O(log n) promedio, O(n) peor caso.
Uso: Búsqueda, inserción y eliminación eficientes.

Bottom-Up (DP)

Definición: Enfoque de DP que resuelve subproblemas desde los más pequeños.
Sinónimo: Tabulation.
Ventaja: No usa call stack, más eficiente en memoria.

C

Caso Base

Definición: Condición que termina la recursión sin más llamadas recursivas.
Ejemplo: En factorial, `if n <= 1: return 1`.

Clase

Definición: Plantilla para crear objetos con atributos y métodos.
Analogía: El plano de una casa; los objetos son las casas construidas.

Colisión (Hash)

Definición: Cuando dos claves diferentes producen el mismo hash.
Resolución: Python usa "open addressing" para encontrar otro slot.

Complejidad Temporal

Definición: Cuánto tiempo toma un algoritmo en función del tamaño de entrada.

Cycle (Grafo)

Definición: Camino que comienza y termina en el mismo vértice.
Detección: DFS puede detectar ciclos en O(V + E).

Cosine Similarity

Definición: Medida de similitud entre vectores basada en el ángulo entre ellos.
Fórmula: cos(θ) = (A·B) / (||A|| × ||B||)
Rango: 0 (perpendiculares) a 1 (paralelos) para vectores TF-IDF.

D

DFS (Depth-First Search)

Definición: Algoritmo de recorrido que explora lo más profundo posible antes de retroceder.
Estructura: Usa Stack o recursión.
Uso: Detectar ciclos, encontrar caminos, topological sort.
Complejidad: O(V + E)

Divide & Conquer

Definición: Estrategia de dividir problema en subproblemas, resolverlos y combinar.
Ejemplos: MergeSort, QuickSort, Binary Search.

Document Frequency (DF)

Definición: Número de documentos que contienen un término.
Uso: Para calcular IDF.

Docstring

Definición: String de documentación al inicio de función/clase/módulo.
Formato: Google style, NumPy style, o reStructuredText.

Dynamic Programming (DP)

Definición: Técnica de optimización que guarda resultados de subproblemas.
Requisitos: Optimal substructure + overlapping subproblems.
Enfoques: Top-down (memoization) y Bottom-up (tabulation).

F

FIFO (First In, First Out)

Definición: Orden donde el primero en entrar es el primero en salir.
Estructura: Queue.
Analogía: Fila del supermercado.

G

Graph (Grafo)

Definición: Estructura de nodos (vértices) conectados por aristas (edges).
Tipos: Dirigido/no dirigido, ponderado/no ponderado.
Representación: Adjacency list o matrix.

Greedy Algorithm

Definición: Estrategia que toma la mejor opción local en cada paso.
Requisito: Greedy choice property para garantizar óptimo.
Ejemplos: Activity selection, Huffman coding.

H

Heap

Definición: Árbol binario completo con propiedad de heap (parent <= children para min-heap).
Operaciones: Insert O(log n), extract-min O(log n), peek O(1).
Uso: Priority queues, heapsort, top-K problems.

Hash Function

Definición: Función que convierte cualquier dato en un número (hash).
Propiedades: Determinista, rápida, distribución uniforme.

Hash Map / Hash Table

Definición: Estructura que mapea claves a valores usando hashing.
En Python: `dict`.
Complejidad: O(1) promedio para get/set/delete.

I

IDF (Inverse Document Frequency)

Definición: Medida de qué tan raro es un término en el corpus.
Fórmula: IDF(t) = log(N / df(t)) donde N = total docs, df = doc frequency.
Intuición: Palabras raras tienen IDF alto.

Índice Invertido

Definición: Estructura que mapea términos a documentos que los contienen.
Estructura: `{término: [lista de doc_ids]}`
Uso: Corazón de los motores de búsqueda.

Inmutabilidad

Definición: Propiedad de objetos que no pueden modificarse después de crearse.
En Python: str, tuple, frozenset son inmutables.

In-Place

Definición: Algoritmo que modifica la estructura original sin crear copia.
Ejemplo: QuickSort in-place usa O(log n) espacio extra.

I

Inorder Traversal

Definición: Recorrido de árbol: Left, Root, Right.
Propiedad: En BST, da elementos en orden ascendente.

L

Leaf Node

Definición: Nodo de árbol sin hijos.
Identificación: node.left == None and node.right == None

LIFO (Last In, First Out)

Definición: Orden donde el último en entrar es el primero en salir.
Estructura: Stack.
Analogía: Pila de platos.

Linked List

Definición: Estructura de nodos donde cada nodo apunta al siguiente.
Tipos: Singly (un puntero), Doubly (dos punteros).
Ventaja: O(1) insert/delete al inicio.

Linter

Definición: Herramienta que analiza código para detectar errores y problemas de estilo.
Ejemplos: ruff, flake8, pylint.

Logarítmico

Definición: Complejidad O(log n) - crece muy lentamente.
Ejemplo: Binary search en 1 billón de elementos = ~30 pasos.

M

Matriz

Definición: Array bidimensional de números.
En Python puro: Lista de listas: `[[1,2], [3,4]]`.

Memoization

Definición: Técnica de cachear resultados de funciones para evitar recálculo.
Uso: Optimizar recursión (ej: Fibonacci).

MergeSort

Definición: Algoritmo de ordenamiento divide & conquer.
Complejidad: O(n log n) siempre.
Propiedad: Estable.

N

Norma (Vector)

Definición: Longitud/magnitud de un vector.
Fórmula: $||v|| = \sqrt{(v_1^2 + v_2^2 + ... + v_n^2)}$

O

Optimal Substructure

Definición: Propiedad donde solución óptima contiene soluciones óptimas de subproblemas.
Requisito: Necesario para aplicar DP o Greedy.

Overlapping Subproblems

Definición: Cuando los mismos subproblemas se resuelven múltiples veces.
Requisito: Necesario para que DP sea beneficioso.

Off-by-One Error

Definición: Error donde un índice está desplazado por 1.
Común en: Loops, binary search, slicing.

OOP (Object-Oriented Programming)

Definición: Paradigma que organiza código en objetos con datos y comportamiento.
Pilares: Encapsulamiento, herencia, polimorfismo.

P

Postorder Traversal

Definición: Recorrido de árbol: Left, Right, Root.
Uso: Eliminar árbol (hijos antes que padre), evaluar expresiones.

Preorder Traversal

Definición: Recorrido de árbol: Root, Left, Right.
Uso: Copiar/serializar árbol.

Priority Queue

Definición: Cola donde elementos salen según prioridad, no orden de llegada.
Implementación: Típicamente con Heap.
Operaciones: Insert O(log n), extract O(log n).

Partition

Definición: En QuickSort, reorganizar array para que elementos < pivot estén antes.
Resultado: Pivot queda en su posición final.

PEP8

Definición: Guía de estilo oficial de Python.
Puntos clave: 4 espacios, 79-88 chars línea, snake_case.

Producto Punto (Dot Product)

Definición: Suma de productos de componentes correspondientes.
Fórmula: $a \cdot b = a_1b_1 + a_2b_2 + ... + a_nb_n$

Property

Definición: Mecanismo para controlar acceso a atributos con getters/setters.
Uso: Validación, cálculo dinámico, encapsulamiento.

Q

Queue

Definición: Estructura de datos FIFO (First In, First Out).
Operaciones: enqueue O(1), dequeue O(1).
Uso: BFS, scheduling, buffers.

QuickSort

Definición: Algoritmo de ordenamiento basado en partición.
Complejidad: O(n log n) promedio, O(n²) peor caso.
Ventaja: In-place, cache-friendly.

R

Recursión

Definición: Técnica donde una función se llama a sí misma.
Componentes: Caso base + caso recursivo.

S

Stack

Definición: Estructura de datos LIFO (Last In, First Out).
Operaciones: push O(1), pop O(1), peek O(1).
Uso: Call stack, DFS, undo, parsing.

Set

Definición: Colección de elementos únicos sin orden.
Operaciones O(1): add, remove, contains.

SOLID

Definición: 5 principios de diseño orientado a objetos.
- **S**ingle Responsibility
- **O**pen/Closed
- **L**iskov Substitution
- **I**nterface Segregation
- **D**ependency Inversion

Stable Sort

Definición: Ordenamiento que mantiene orden relativo de elementos iguales.
Ejemplo: MergeSort es estable, QuickSort no.

T

Tabulation

Definición: Enfoque de DP que llena tabla iterativamente desde casos base.
Sinónimo: Bottom-up DP.
Ventaja: No usa call stack.

Top-Down (DP)

Definición: Enfoque de DP recursivo con memoization.
Ventaja: Solo calcula subproblemas necesarios.

Tree (Árbol)

Definición: Estructura jerárquica de nodos sin ciclos.
Términos: Root, parent, child, leaf, height, depth.
Tipos: Binary tree, BST, AVL, etc.

Tree Traversal

Definición: Visitar todos los nodos de un árbol.
DFS: Inorder, Preorder, Postorder.
BFS: Level-order.

Term Frequency (TF)

Definición: Frecuencia de un término en un documento.
Fórmula: TF(t,d) = count(t,d) / total_terms(d)

TF-IDF

Definición: Producto de Term Frequency × Inverse Document Frequency.
Uso: Medir importancia de término en documento dentro de corpus.

Tokenización

Definición: Proceso de dividir texto en unidades (tokens).
Ejemplo: "Hello, World!" → ["hello", "world"]

Type Hint

Definición: Anotación que indica el tipo esperado de variable/parámetro/retorno.
Ejemplo: `def greet(name: str) -> str:`

V

Vector

Definición: Lista ordenada de números que representa punto/dirección en espacio.
En Python puro: `list[float]`
Uso en IR: Representar documentos en espacio de términos.

Vertex (Vértice)

Definición: Nodo en un grafo.
Plural: Vertices.
Notación: V = número de vértices.

Vocabulario

Definición: Conjunto de todos los términos únicos en un corpus.
Tamaño: Determina dimensión de vectores TF-IDF.

ML y Probabilidad (Pathway)

Bayes {#bayes}

Definición: Marco probabilístico para actualizar creencias (probabilidades) usando nueva evidencia.
Teorema: $P(A \mid B) = P(B \mid A) \cdot P(A) / P(B)$.
Uso: Clasificadores probabilísticos como Naive Bayes.

Backpropagation {#backpropagation}

Definición: Algoritmo para calcular gradientes en redes neuronales aplicando la regla de la cadena hacia atrás.
Uso: Entrenar redes neuronales con Gradient Descent o variantes.

Distribución Normal {#distribucion-normal}

Definición: Distribución continua en forma de campana, simétrica alrededor de la media.
Parámetros: Media μ y desviación estándar σ.
Uso: Modelar ruido, teorema central del límite, intervalos de confianza.

Gradient Descent {#gradient-descent}

Definición: Algoritmo iterativo para minimizar una función moviéndose en dirección del gradiente negativo.
Uso: Entrenar modelos de regresión y redes neuronales.

Logistic Regression {#logistic-regression}

Definición: Modelo de clasificación binaria que aplica la función sigmoide a una combinación lineal de features.
Salida: Estima $P(y = 1 \mid x)$ y decide con un umbral.

KMeans {#kmeans}

Definición: Algoritmo de clustering que asigna puntos al centroide más cercano y actualiza centroides iterativamente.
Salida: Partición de los datos en k grupos.

PCA {#pca}

Definición: Principal Component Analysis; técnica de reducción de dimensionalidad basada en autovectores de la matriz de covarianza.
Uso: Proyectar datos a pocas dimensiones preservando la mayor varianza posible.

Neural Network {#neural-network}

Definición: Modelo compuesto por capas de neuronas (funciones) conectadas que transforman vectores de entrada en salida.
Uso: Aprender representaciones no lineales complejas.

MLE {#mle}

Definición: Maximum Likelihood Estimation; método para estimar parámetros maximizando la verosimilitud de los datos observados.

MAP {#map}

Definición: Maximum A Posteriori; similar a MLE pero incorporando un prior sobre los parámetros.

Markov Chain {#markov-chain}

Definición: Proceso estocástico donde la próxima transición depende solo del estado actual.
Uso: Modelar cadenas de estados, PageRank, modelos de clima.

MCMC {#mcmc}

Definición: Markov Chain Monte Carlo; familia de métodos que usan cadenas de Markov para muestrear de distribuciones complejas.
Ejemplos: Metropolis-Hastings, Gibbs Sampling.

Siglas Comunes

Sigla	Significado
BST	Binary Search Tree
BFS	Breadth-First Search
DFS	Depth-First Search
DP	Dynamic Programming
FIFO	First In, First Out
LIFO	Last In, First Out
OOP	Object-Oriented Programming
TF	Term Frequency
IDF	Inverse Document Frequency
MLE	Maximum Likelihood Estimation
MAP	Maximum A Posteriori
MCMC	Markov Chain Monte Carlo
PCA	Principal Component Analysis

Simulacro de Entrevista

Guía MS in AI Pathway

DUQUEOM · 2025

🎯 Simulacro de Entrevista - MS in AI Pathway

120+ preguntas con respuestas detalladas para las **2 líneas del Pathway**

📋 Estructura del Simulacro

Sección	Categoría	Preguntas	Tiempo
1. Python y OOP	[PRERREQUISITO]	10	15 min
2. Estructuras de Datos	[PRERREQUISITO]	15	25 min
3. Trees y Graphs	[PRERREQUISITO]	15	30 min
4. Algoritmos y DP	[PRERREQUISITO]	20	40 min
5. Matemáticas y Big O	[PRERREQUISITO]	20	30 min
6. Probabilidad y Estadística	⭐ [PATHWAY LÍNEA 2]	20	30 min
7. Machine Learning	⭐ [PATHWAY LÍNEA 1]	20	35 min

Total: 120+ preguntas, ~205 minutos

✅ Checklist Mínimo Pathway

Si tienes poco tiempo, **prioriza las secciones 6 y 7**:
- [ ] Sección 6: 20 preguntas de Probabilidad/Estadística
- [ ] Sección 7: 20 preguntas de Machine Learning

Sección 1: Python y OOP [PRERREQUISITO]

P1: ¿Qué son los type hints y por qué usarlos?

R: Anotaciones que indican tipos esperados. Beneficios: documentación viva, detección de errores con mypy, mejor autocompletado.

```
def greet(name: str) -> str:
    return f"Hello, {name}"
```

P2: ¿Cuál es la diferencia entre `list` y `tuple`?

R:
- `list`: mutable, se puede modificar
- `tuple`: inmutable, no se puede cambiar después de crear
- `tuple` es hashable (puede ser clave de dict), `list` no

P3: ¿Qué significa que Python sea "pass by object reference"?

R: Se pasa referencia al objeto. Si el objeto es mutable, cambios dentro de la función afectan al original. Si es inmutable, se crea nuevo objeto.

P4: ¿Para qué sirve `__init__`?

R: Inicializar atributos de instancia cuando se crea un objeto. Es el constructor de la clase.

P5: ¿Cuál es la diferencia entre `__str__` y `__repr__`?

R:
- `__str__`: para usuarios, legible
- `__repr__`: para desarrolladores, sin ambigüedad, idealmente eval-able

P6: ¿Qué es un property en Python?

R: Mecanismo para controlar acceso a atributos con getter/setter, manteniendo sintaxis de atributo.

P7: ¿Qué significa "composición sobre herencia"?

R: Preferir contener objetos de otra clase (has-a) sobre heredar (is-a). Más flexible y menos acoplado.

P8: ¿Qué es una función pura?

R: Función que siempre retorna mismo output para mismo input y no tiene efectos secundarios.

P9: ¿Para qué sirve `@dataclass`?

R: Genera automáticamente `__init__`, `__repr__`, `__eq__` para clases que principalmente almacenan datos.

P10: ¿Cómo harías una clase inmutable?

R: Usar `@dataclass(frozen=True)` o definir `__setattr__` para prevenir modificaciones.

Sección 2: Estructuras de Datos [PRERREQUISITO]

P11: ¿Cuál es la complejidad de buscar en una lista vs en un set?

R: Lista: O(n), Set: O(1) promedio. Set usa hashing.

P12: ¿Cómo funciona internamente un diccionario?

R: Hash table. La clave se hashea para determinar posición en array interno. Colisiones se resuelven con probing.

P13: ¿Por qué `dict` es O(1) para acceso?

R: Hash de la clave da posición directa. No necesita buscar secuencialmente.

P14: ¿Qué es una colisión en hash table?

R: Cuando dos claves diferentes producen el mismo hash. Se resuelve buscando siguiente slot disponible.

P15: ¿Qué puede ser clave de diccionario?

R: Solo objetos hashables (inmutables): str, int, float, tuple, frozenset. No: list, set, dict.

P16: ¿Cuál es la diferencia entre `set` y `frozenset`?

R: `set` es mutable, `frozenset` inmutable. frozenset puede ser clave de dict o elemento de otro set.

P17: ¿Qué es un índice invertido?

R: Estructura que mapea términos a documentos que los contienen. `{"word": [doc1, doc2, ...]}`. Base de motores de búsqueda.

P18: ¿Por qué usarías un set para stop words?

R: Búsqueda O(1). Si son 50 stop words y 1000 tokens, con lista sería O(50×1000)=O(50000), con set O(1000).

P19: ¿Cuál es la complejidad de `list.append()` vs `list.insert(0, x)`?

R:
- append: O(1) amortizado
- insert(0): O(n) porque mueve todos los elementos

P20: ¿Qué estructura usarías para un contador de frecuencias?

R: `dict` o `collections.Counter`. Mapea elemento a conteo, acceso O(1).

P21: ¿Cómo implementarías búsqueda AND con sets?

R: Intersección: `set1 & set2`. Retorna elementos en ambos.

P22: ¿Cómo implementarías búsqueda OR con sets?

R: Unión: `set1 | set2`. Retorna elementos en cualquiera.

P23: ¿Qué es Document Frequency?

R: Número de documentos que contienen un término. Usado para calcular IDF.

P24: ¿Cuándo usarías `defaultdict`?

R: Cuando quieres valores por defecto automáticos. Ej: `defaultdict(list)` crea listas vacías para claves nuevas.

P25: ¿Qué es un posting list?

R: Lista de documentos que contienen un término, almacenada en índice invertido.

Sección 3: Trees y Graphs [PRERREQUISITO]

P26: ¿Qué es un Binary Tree?

R: Árbol donde cada nodo tiene máximo 2 hijos (left y right).

P27: ¿Cuál es la diferencia entre Binary Tree y BST?

R:
- Binary Tree: cualquier árbol con máx 2 hijos
- BST: Binary tree donde left < root < right

P28: ¿Cuáles son los tres traversals DFS de un árbol?

R:
- Inorder: Left, Root, Right (en BST da orden ascendente)
- Preorder: Root, Left, Right
- Postorder: Left, Right, Root

P29: ¿Cómo implementarías level-order traversal?

R: Usar Queue (BFS). Agregar root, luego procesar nivel por nivel.

P30: ¿Cuál es la complejidad de search en BST?

R: O(log n) promedio, O(n) peor caso (árbol desbalanceado/lineal).

P31: ¿Qué es un grafo dirigido vs no dirigido?

R:
- Dirigido: edges tienen dirección (A→B no implica B→A)
- No dirigido: conexión bidireccional (A↔B)

P32: ¿Cuáles son las dos formas de representar un grafo?

R:
- Adjacency List: dict de listas, O(V+E) espacio
- Adjacency Matrix: matriz V×V, O(V²) espacio

P33: ¿Cuál es la diferencia entre BFS y DFS?

R:
- BFS: explora por niveles, usa Queue, encuentra shortest path
- DFS: explora en profundidad, usa Stack/recursión

P34: ¿Cuándo usar BFS vs DFS?

R:
- BFS: shortest path (no ponderado), nivel por nivel
- DFS: detectar ciclos, caminos, backtracking

P35: ¿Cómo detectar un ciclo en un grafo?

R: DFS marcando nodos como "en progreso" y "visitado". Si encuentras nodo "en progreso", hay ciclo.

P36: ¿Qué es un DAG?

R: Directed Acyclic Graph. Grafo dirigido sin ciclos. Permite topological sort.

P37: ¿Cuál es la complejidad de BFS/DFS?

R: O(V + E) donde V = vértices, E = edges.

P38: ¿Qué estructura usa BFS y cuál DFS?

R:
- BFS: Queue (FIFO)
- DFS: Stack (LIFO) o recursión

P39: ¿Por qué BFS garantiza shortest path en grafos no ponderados?

R: Porque explora todos los nodos a distancia k antes de los de distancia k+1.

P40: ¿Cómo encontrarías camino más corto en grafo ponderado?

R: Dijkstra's algorithm (no cubierto en detalle, pero saber que existe).

Sección 4: Algoritmos y DP [PRERREQUISITO]

P41: Explica cómo funciona QuickSort.

R:
1. Elegir pivote
2. Particionar: menores a izquierda, mayores a derecha
3. Recursivamente ordenar cada partición
Complejidad: O(n log n) promedio, $O(n^2)$ peor caso.

P42: ¿Por qué QuickSort puede ser $O(n^2)$?

R: Si el pivote siempre es el mínimo o máximo. Ej: lista ya ordenada con pivote fijo al final. Cada partición solo reduce en 1.

P28: ¿Cómo evitar el peor caso de QuickSort?

R: Random pivot selection. Aleatoriza la elección del pivote.

P29: Explica MergeSort.

R:
1. Dividir lista en dos mitades
2. Ordenar cada mitad recursivamente
3. Fusionar las mitades ordenadas
Complejidad: O(n log n) siempre.

P30: ¿Cuál es la diferencia entre QuickSort y MergeSort?

R:
- QuickSort: in-place, O(log n) espacio, no estable
- MergeSort: O(n) espacio, estable, siempre O(n log n)

P31: ¿Qué significa que un sort sea "estable"?

R: Elementos iguales mantienen su orden relativo original.

P32: Explica Binary Search.

R: En lista ordenada, comparar con elemento medio. Si menor, buscar en mitad izquierda; si mayor, en derecha. Complejidad: O(log n).

P33: ¿Cuál es el error off-by-one más común en binary search?

R: Usar `while left < right` en lugar de `left <= right`, o no ajustar correctamente mid+1/mid-1.

P34: ¿Qué es recursión?

R: Función que se llama a sí misma. Requiere caso base (termina) y caso recursivo (se llama con input menor).

P35: ¿Qué es el call stack?

R: Pila que guarda estado de cada llamada a función. Cada llamada recursiva agrega un frame.

P36: ¿Qué es memoization?

R: Cachear resultados de funciones para evitar recálculo. Útil en recursión con subproblemas repetidos.

P37: ¿Por qué Fibonacci naive es O(2^n)?

R: Cada llamada hace dos llamadas. Árbol de llamadas crece exponencialmente. fib(n) se recalcula muchas veces.

P38: ¿Cómo optimizar Fibonacci a O(n)?

R: Memoization: guardar resultados en dict/cache. Cada valor se calcula solo una vez.

P39: ¿Qué es Divide & Conquer?

R: Patrón que divide problema en subproblemas, resuelve cada uno, y combina soluciones. Ej: MergeSort, QuickSort.

P43: ¿Cómo fusionarías dos listas ordenadas?

R: Two pointers: comparar elementos actuales de ambas, agregar el menor al resultado, avanzar ese puntero. O(n+m).

P44: ¿Qué es Dynamic Programming?

R: Técnica que guarda resultados de subproblemas para evitar recálculo. Requiere optimal substructure + overlapping subproblems.

P45: ¿Cuáles son los dos enfoques de DP?

R:
- Top-down: Recursivo con memoization
- Bottom-up: Iterativo con tabulation

P46: ¿Qué es la recurrencia de Coin Change?

R: dp[amount] = min(dp[amount - coin] + 1) para todas las monedas válidas.

P47: ¿Cuándo usar Greedy vs DP?

R:
- Greedy: Si la mejor opción local lleva al óptimo global
- DP: Si necesitas explorar todas las opciones

P48: ¿Qué es "greedy choice property"?

R: Propiedad donde elegir el óptimo local en cada paso lleva al óptimo global.

P49: ¿Cómo funciona Activity Selection greedy?

R: Ordenar por tiempo de fin, siempre elegir la que termina primero y no se superpone.

P50: ¿Qué es un Heap?

R: Árbol binario completo con propiedad heap (parent <= children para min-heap).

P51: ¿Cuáles son las complejidades de operaciones en Heap?

R: Insert: O(log n), Extract-min: O(log n), Peek: O(1), Heapify: O(n).

P52: ¿Cómo encontrar los K elementos más grandes?

R: Usar min-heap de tamaño k. Para cada elemento, si es mayor que el mínimo del heap, reemplazar.

P53: ¿Por qué usar min-heap para K largest?

R: Min-heap mantiene el k-ésimo más grande en la raíz. Elementos más grandes que la raíz entran al heap.

P54: ¿Qué es Priority Queue?

R: Cola donde elementos salen por prioridad, no por orden de llegada. Se implementa con Heap.

P55: ¿Cómo hacer max-heap en Python?

R: heapq es min-heap. Para max-heap, negar los valores al insertar y al extraer.

Sección 5: Matemáticas y Big O [PRERREQUISITO]

P56: ¿Qué significa O(n)?

R: El tiempo crece linealmente con el tamaño de entrada. Duplicar n duplica el tiempo.

P57: Ordena de menor a mayor: $O(n^2)$, O(1), $O(n \log n)$, $O(\log n)$, O(n)

R: $O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2)$

P58: ¿Cuántas comparaciones hace binary search en 1 millón de elementos?

R: $\log_2(1{,}000{,}000) \approx 20$ comparaciones.

P59: ¿Qué es el producto punto?

R: Suma de productos de componentes correspondientes: $a \cdot b = a_1b_1 + a_2b_2 + \ldots$ Resultado es escalar.

P45: ¿Qué es la norma de un vector?

R: Su longitud/magnitud. $||v|| = \sqrt{(v_1^2 + v_2^2 + \ldots)}$. Distancia del origen al punto.

P46: ¿Qué mide la similitud de coseno?

R: El coseno del ángulo entre vectores. 1 = misma dirección, 0 = perpendiculares. Mide similitud ignorando magnitud.

P47: ¿Qué es TF (Term Frequency)?

R: Frecuencia de un término en un documento, normalizada por longitud. TF = count/ total_terms.

P48: ¿Qué es IDF (Inverse Document Frequency)?

R: Mide qué tan raro es un término. IDF = log(N/df). Términos raros tienen IDF alto.

P49: ¿Por qué usamos TF-IDF en lugar de solo TF?

R: TF solo mide frecuencia local. IDF penaliza palabras comunes ("the", "is"). TF-IDF balancea ambos.

P50: ¿Cuál es la complejidad de calcular similitud de coseno?

R: O(V) donde V es la dimensión del vector (tamaño del vocabulario). Hay que recorrer todos los componentes.

Sección 6: Probabilidad y Estadística ⭐ [PATHWAY LÍNEA 2]

P60: ¿Qué es el Teorema de Bayes y para qué se usa en ML?

R: $P(A|B) = P(B|A) \times P(A) / P(B)$. Permite actualizar creencias (prior) dado nueva evidencia (likelihood). Base de clasificadores Naive Bayes y modelos probabilísticos.

P61: ¿Cuál es la diferencia entre probabilidad y likelihood?

R:
- Probabilidad: $P(data|params)$ - probabilidad de datos dados parámetros fijos
- Likelihood: $L(params|data)$ - qué tan probables son los parámetros dados los datos

P62: ¿Qué es MLE (Maximum Likelihood Estimation)?

R: Encontrar los parámetros θ que maximizan la probabilidad de observar los datos: $\hat{\theta} = \text{argmax } P(data|\theta)$. Es cómo se entrenan la mayoría de modelos de ML.

P63: ¿Qué es MAP y cómo se relaciona con regularización?

R: Maximum A Posteriori incorpora un prior: $\hat{\theta} = \text{argmax } P(\theta|data) \propto P(data|\theta) \times P(\theta)$. Prior gaussiano → L2 regularization. Prior laplaciano → L1 regularization.

P64: ¿Qué es la distribución normal y por qué es importante?

R: Distribución "campana de Gauss". Importante por el Teorema del Límite Central: la suma de muchas variables independientes tiende a normal. Muchos errores en ML se asumen normales.

P65: ¿Qué es esperanza y varianza?

R:
- $E[X] = \Sigma\, x \times P(x)$ = "valor promedio esperado"
- $Var(X) = E[(X - \mu)^2]$ = "spread" alrededor de la media

P66: ¿Qué es una cadena de Markov?

R: Proceso estocástico donde el futuro solo depende del estado actual, no del pasado: $P(X_{n+1}|X_n, X_{n-1}, ...) = P(X_{n+1}|X_n)$. Usado en PageRank, modelos de lenguaje.

P67: ¿Qué es la distribución estacionaria de una cadena de Markov?

R: Distribución π tal que $\pi = \pi P$. Después de muchos pasos, la cadena converge a esta distribución sin importar el estado inicial.

P68: ¿Qué es MCMC y para qué se usa?

R: Markov Chain Monte Carlo. Técnica para muestrear de distribuciones complejas construyendo una cadena de Markov cuya distribución estacionaria es la distribución objetivo.

P69: Explica el algoritmo Metropolis-Hastings.

R:
1. Proponer nuevo estado x' desde distribución q(x'|x)
2. Aceptar con probabilidad min(1, P(x')/P(x))
3. Si acepta, mover a x'; si no, quedarse en x
4. Repetir

P70: ¿Qué es un intervalo de confianza?

R: Rango [a,b] tal que si repitiéramos el experimento muchas veces, el parámetro real estaría dentro del intervalo en (1-α)% de las veces (ej: 95%).

P71: ¿Cuál es la diferencia entre error Tipo I y Tipo II?

R:
- Tipo I (α): Rechazar H_0 cuando es verdadera (falso positivo)
- Tipo II (β): No rechazar H_0 cuando es falsa (falso negativo)

P72: ¿Qué es covarianza y correlación?

R:
- $Cov(X,Y) = E[(X-\mu_x)(Y-\mu_Y)]$ - Relación lineal, no normalizada
- $Correlation = Cov(X,Y)/(\sigma_x\sigma_Y)$ - Normalizada a [-1, 1]

P73: ¿Qué es la distribución Bernoulli y Binomial?

R:
- Bernoulli: Un solo experimento con prob p de éxito
- Binomial: k éxitos en n experimentos Bernoulli independientes

P74: ¿Por qué usamos log-likelihood en lugar de likelihood?

R: Producto de probabilidades pequeñas → underflow. Logaritmo convierte productos en sumas, numéricamente más estable.

P75: ¿Qué es el Teorema del Límite Central?

R: La distribución de la media muestral tiende a una normal cuando $n \to \infty$, sin importar la distribución original. Justifica asumir normalidad en muchos contextos.

P76: ¿Qué es independencia condicional?

R: $P(A,B|C) = P(A|C) \times P(B|C)$. A y B son independientes dado C. Base de Naive Bayes: features son independientes dado la clase.

P77: ¿Qué es estimador insesgado?

R: Estimador cuyo valor esperado es igual al parámetro real: $E[\hat{\theta}] = \theta$. Ejemplo: media muestral es insesgada para la media poblacional.

P78: ¿Qué es bootstrap?

R: Técnica de remuestreo: crear muchas muestras tomando con reemplazo de los datos originales. Usado para estimar varianza de estimadores.

P79: ¿Qué es el p-value?

R: Probabilidad de observar resultados tan extremos como los observados, asumiendo que H_0 es verdadera. Si $p < \alpha$, rechazamos H_0.

Sección 7: Machine Learning ⭐ [PATHWAY LÍNEA 1]

P80: ¿Cuál es la diferencia entre aprendizaje supervisado y no supervisado?

R:
- Supervisado: Datos etiquetados (X, y). Objetivo: predecir y dado X.
- No supervisado: Solo datos X. Objetivo: encontrar estructura (clusters, dimensiones).

P81: ¿Qué es el bias-variance tradeoff?

R: Error = $Bias^2$ + Variance + Ruido irreducible.
- Bias alto → underfitting (modelo muy simple)
- Variance alta → overfitting (modelo muy complejo)

Objetivo: encontrar el balance óptimo.

P82: ¿Qué es overfitting y cómo detectarlo?

R: Modelo aprende ruido del train set y no generaliza. Se detecta cuando train accuracy >> test accuracy. Soluciones: más datos, regularización, menos complejidad.

P83: ¿Qué es cross-validation y para qué sirve?

R: Dividir datos en k folds, entrenar en k-1 y validar en 1, rotar. Da estimación más robusta del rendimiento que un solo train/test split.

P84: Explica gradient descent.

R: Algoritmo de optimización: $w = w - lr \times \partial L/\partial w$. Sigue la dirección de máxima pendiente descendente para minimizar la función de pérdida.

P85: ¿Cuál es la diferencia entre batch, mini-batch y SGD?

R:
- Batch: Usa todos los datos para cada update
- Mini-batch: Usa subconjunto (ej: 32 samples)

- SGD: Usa 1 sample por update
Mini-batch es el más común: balance entre estabilidad y velocidad.

P86: ¿Qué es regularización L1 y L2?

R:
- L1 (Lasso): Suma de |w|. Produce sparsity (pesos = 0).
- L2 (Ridge): Suma de w^2. Shrinks pesos pero no a cero.
Ambas previenen overfitting al penalizar pesos grandes.

P87: Explica regresión logística.

R: Clasificador lineal: $P(y=1|x) = \sigma(w^Tx + b)$. Usa sigmoid para mapear a [0,1]. Se entrena minimizando binary cross-entropy con gradient descent.

P88: ¿Cómo funciona un árbol de decisión?

R: Divide recursivamente los datos según el feature que maximiza ganancia de información (o minimiza Gini). Hojas contienen predicciones. Fácil de interpretar, propenso a overfitting.

P89: ¿Qué es Random Forest?

R: Ensemble de árboles de decisión. Cada árbol entrena en bootstrap sample con subset de features aleatorio. Predicción final = promedio/voto mayoritario. Reduce variance.

P90: Explica K-Nearest Neighbors.

R: Predice según el voto de los k vecinos más cercanos. No-paramétrico (no entrena). Complejidad $O(n \times d)$ por predicción. Sensible a escala de features.

P91: ¿Qué es SVM y cuál es la idea del kernel trick?

R: SVM encuentra hiperplano con máximo margen entre clases. Kernel trick: proyectar a dimensión superior donde datos son linealmente separables, sin calcular la proyección explícita.

P92: ¿Qué métricas usarías para clasificación desbalanceada?

R: Accuracy engaña. Mejor usar:
- Precision: TP/(TP+FP) - de los predichos +, cuántos son +
- Recall: TP/(TP+FN) - de los reales +, cuántos encontramos
- F1: armonic mean de precision y recall
- AUC-ROC

P93: Explica K-Means clustering.

R:
1. Inicializar k centroides aleatorios
2. Asignar cada punto al centroide más cercano
3. Actualizar centroides al promedio de sus puntos
4. Repetir hasta convergencia
Requiere especificar k. Sensible a inicialización.

P94: ¿Cómo elegir el número de clusters en K-Means?

R:
- Elbow method: graficar inertia vs k, buscar "codo"

- Silhouette score: mide cohesión vs separación
- Domain knowledge

P95: ¿Qué es PCA y para qué sirve?

R: Principal Component Analysis. Reduce dimensionalidad proyectando a direcciones de máxima varianza (eigenvectors de la matriz de covarianza). Usado para visualización, compresión, preprocesamiento.

P96: ¿Qué es una red neuronal?

R: Composición de funciones: $y = f(W_3 \times f(W_2 \times f(W_1x + b_1) + b_2) + b_3)$. Cada capa es transformación lineal + activación no lineal. Aprende features automáticamente.

P97: Explica backpropagation.

R: Algoritmo para calcular gradientes en redes neuronales usando la regla de la cadena. Forward pass calcula output, backward pass propaga gradientes desde el loss hacia atrás.

P98: ¿Qué son funciones de activación y cuáles conoces?

R: Funciones no lineales entre capas.
- Sigmoid: (0,1), problemas de vanishing gradient
- ReLU: max(0,x), estándar para capas ocultas
- Softmax: para output de clasificación multiclase

P99: ¿Qué es una CNN y para qué se usa?

R: Convolutional Neural Network. Capas de convolución extraen features espaciales. Usadas para imágenes. Ventaja: comparten parámetros, detectan patrones independiente de posición.

P100: ¿Qué es una RNN y cuál es el problema del vanishing gradient?

R: Recurrent Neural Network. Estado oculto depende del anterior, captura secuencias. Vanishing gradient: gradientes se vuelven muy pequeños en secuencias largas. Solución: LSTM, GRU.

🎯 Autoevaluación

Respuestas Correctas	Nivel
100-120	🏆 Listo para Pathway - Ambas líneas
80-99	✅ Buen nivel, reforzar gaps
60-79	⚠️ Necesita más estudio
<60	❌ Revisar módulos

💡 Tips para la Entrevista Real

1. **Explica tu pensamiento:** Verbaliza mientras resuelves
2. **Empieza simple:** Primero solución bruta, luego optimiza
3. **Pregunta si dudas:** Clarifica requisitos
4. **Analiza Big O:** Siempre menciona complejidad
5. **Practica en inglés:** Todo el Pathway es en inglés
6. **Conecta conceptos:** ML usa probabilidad, DL usa álgebra lineal

7. **Implementa desde cero:** Demuestra que entiendes, no solo usas sklearn

Recursos Recomendados

Guía MS in AI Pathway

DUQUEOM · 2025

📚 Recursos de Aprendizaje - MS in AI Pathway

> Cursos y recursos organizados por prioridad para aprobar el Pathway.

🎯 CURSOS DEL PATHWAY (OBLIGATORIOS)

Estos son los **6 cursos exactos** que debes aprobar para el Pathway:

Línea 1: Machine Learning (3 créditos)

Curso	Enlace	Preparación en Esta Guía
Introduction to Machine Learning: Supervised Learning	Coursera	Módulo 22
Unsupervised Algorithms in Machine Learning	Coursera	Módulo 23
Introduction to Deep Learning	Coursera	Módulo 24

Línea 2: Probabilidad y Estadística (3 créditos)

Curso	Enlace	Preparación en Esta Guía
Probability Theory: Foundation for Data Science	Coursera	Módulo 19
Discrete-Time Markov Chains and Monte Carlo Methods	Coursera	Módulo 21
Statistical Inference for Estimation in Data Science	Coursera	Módulo 20

> ⚠️ **IMPORTANTE:** Puedes auditar estos cursos GRATIS en Coursera para ver el contenido antes de inscribirte oficialmente.

🔥 CURSOS DE PREPARACIÓN (MUY RECOMENDADOS)

Machine Learning Foundations

Curso	Plataforma	Por Qué
Machine Learning Specialization	Coursera (Andrew Ng)	El mejor curso de ML, usa Python
Deep Learning Specialization	Coursera (Andrew Ng)	Profundiza en redes neuronales

Probabilidad y Estadística

Curso	Plataforma	Por Qué
Probability & Statistics for ML	Coursera (DeepLearning.AI)	Enfocado en ML
Bayesian Statistics	Coursera (Duke)	Profundiza en Bayes

Matemáticas Fundamentales

Curso	Plataforma	Por Qué
Mathematics for ML: Linear Algebra	Coursera (Imperial)	Base para todo ML
Mathematics for ML: Multivariate Calculus	Coursera (Imperial)	Gradientes y optimización

---

📚 PRERREQUISITOS (Complementarios)

Algoritmos y DSA (Base, no foco)

Curso	Plataforma	Prioridad
Algorithms Specialization	Coursera (Stanford)	Media
Data Structures & Algorithms	Coursera (UCSD)	Media

Python

Curso	Plataforma	Nivel
Python for Everybody	Coursera	Básico
Real Python Tutorials	Web	Todos

📖 Libros

Machine Learning (Prioridad Alta)

Libro	Autor	Por Qué
Hands-On Machine Learning	Aurélien Géron	Práctico, con código, muy completo
Pattern Recognition and ML	Christopher Bishop	Teórico, excelente para fundamentos
The Hundred-Page ML Book	Andriy Burkov	Resumen conciso de todo ML

Deep Learning

Libro	Autor	Por Qué
Deep Learning	Goodfellow et al.	La biblia de DL, gratis online
Neural Networks and Deep Learning	Michael Nielsen	Gratis online, muy didáctico

Probabilidad y Estadística

Libro	Autor	Por Qué
Think Stats	Allen Downey	Gratis online, práctico con Python
Think Bayes	Allen Downey	Gratis online, Bayesiano
All of Statistics	Larry Wasserman	Referencia completa

Matemáticas para ML

Libro	Autor	Por Qué
Mathematics for ML	Deisenroth et al.	Gratis online, fundamental
Linear Algebra Done Right	Axler	Álgebra lineal rigurosa

Algoritmos (Complementario)

Libro	Autor	Por Qué
Grokking Algorithms	Aditya Bhargava	Visual, accesible, para empezar
Introduction to Algorithms (CLRS)	Cormen et al.	Referencia completa

---

🎥 Videos

Canales de YouTube (ML/DL/Probabilidad)

Canal	Tema	Por Qué
3Blue1Brown	Matemáticas + DL	Neural networks playlist es EXCELENTE
StatQuest	Estadística + ML	Explicaciones claras de conceptos complejos
Sentdex	ML práctico	Implementaciones desde cero
Two Minute Papers	Investigación IA	Para mantenerte motivado

Playlists Esenciales

Playlist	Tema	Link
Neural Networks	Deep Learning	3B1B Neural Networks
Linear Algebra	Matemáticas	3B1B Linear Algebra
Probability	Probabilidad	StatQuest Probability
Machine Learning	ML	StatQuest ML

🛠️ Herramientas

Desarrollo

Herramienta	Propósito
VS Code	Editor de código
Python	3.11+ recomendado
Git	Control de versiones

Python Tooling

Herramienta	Propósito	Comando
mypy	Type checking	`pip install mypy`
ruff	Linting rápido	`pip install ruff`
pytest	Testing	`pip install pytest`
pytest-cov	Coverage	`pip install pytest-cov`

Visualización de Algoritmos

Herramienta	URL
Visualgo	visualgo.net
Python Tutor	pythontutor.com
Algorithm Visualizer	algorithm-visualizer.org

💪 Práctica de Algoritmos

Plataformas

Plataforma	Nivel	Enfoque
LeetCode	Todos	Entrevistas técnicas
HackerRank	Principiante-Intermedio	Aprendizaje estructurado
Codewars	Todos	Katas cortos
Project Euler	Matemático	Problemas matemáticos

Problemas Recomendados por Tema

Arrays y Strings

- Two Sum (LeetCode #1)
- Valid Anagram (LeetCode #242)
- Reverse String (LeetCode #344)

Hash Maps

- Group Anagrams (LeetCode #49)
- Word Pattern (LeetCode #290)
- Top K Frequent Elements (LeetCode #347)

Sorting

- Sort an Array (LeetCode #912)
- Merge Intervals (LeetCode #56)
- Kth Largest Element (LeetCode #215)

Binary Search

- Binary Search (LeetCode #704)
- Search Insert Position (LeetCode #35)
- First Bad Version (LeetCode #278)

🎯 Pathway de CU Boulder - CURSOS EXACTOS

Las 2 Líneas del Pathway (6 créditos total)

LÍNEA 1: Machine Learning (3 créditos)
Curso	Créditos	Módulo Preparación
Introduction to Machine Learning: Supervised Learning	1	22
Unsupervised Algorithms in Machine Learning	1	23
Introduction to Deep Learning	1	24

LÍNEA 2: Probability & Statistics (3 créditos)
Curso	Créditos	Módulo Preparación
Probability Theory: Foundation for Data Science	1	19

| Discrete-Time Markov Chains and Monte Carlo Methods | 1 | 21 |
| Statistical Inference for Estimation in Data Science | 1 | 20 |

Preparación Específica

1. **Estudiar esta guía** - Módulos 19-24 cubren TODO el contenido
2. **Auditar los cursos** en Coursera (gratis) para ver el formato
3. **Practicar implementaciones** desde cero (sin sklearn)
4. **Dominar** Bayes, MLE, backpropagation, K-Means
5. **Completar el proyecto integrador** (Módulo 12)

📅 Ruta de Aprendizaje Sugerida (8 meses)

Mes 1-2: Fundamentos

- Guía: Módulos 01-06 (Python, OOP, Estructuras básicas)
- Curso: Mathematics for ML: Linear Algebra
- Videos: 3B1B Linear Algebra

Mes 3-4: Prerrequisitos Avanzados

- Guía: Módulos 07-11, 13-18 (Algoritmos, Álgebra Lineal)
- Libro: Mathematics for ML (gratis)
- Práctica: Implementar sorting, trees, graphs

Mes 5: Probabilidad y Estadística ⭐

- Guía: Módulos 19-21
- Curso: Probability & Statistics for ML (Coursera)
- Videos: StatQuest Probability playlist
- Libro: Think Stats, Think Bayes

Mes 6-7: Machine Learning ⭐

- Guía: Módulos 22-24
- Curso: Machine Learning Specialization (Andrew Ng)
- Videos: 3B1B Neural Networks
- Libro: Hands-On Machine Learning

Mes 8: Proyecto e Integración

- Guía: Módulo 12 (Proyecto Integrador)
- Auditar cursos del Pathway
- Practicar explicar en inglés
- Simulacros de entrevista

🔗 Links Directos

- MS in AI - CU Boulder
- Pathway Admissions
- Mathematics for ML Book (Free)
- Deep Learning Book (Free)
- Think Stats (Free)
- Think Bayes (Free)

Anexo DSA - Arrays y Strings

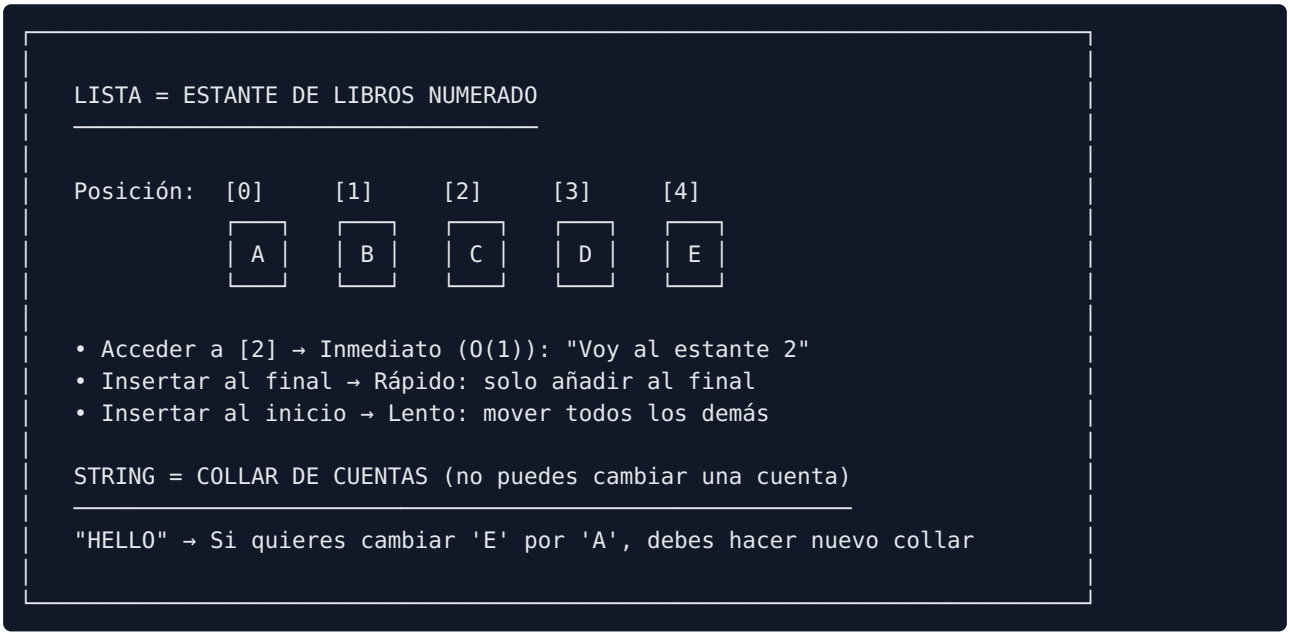
Guía MS in AI Pathway

DUQUEOM · 2025

Anexo DSA - Arrays, Strings y Memoria

> ⚠️ **MÓDULO OPCIONAL:** Este módulo NO es requerido para el Pathway. Es útil para entrevistas técnicas.
> 🎯 **Objetivo:** Dominar la manipulación de listas y strings en Python.

🧠 Analogía: El Estante de Libros

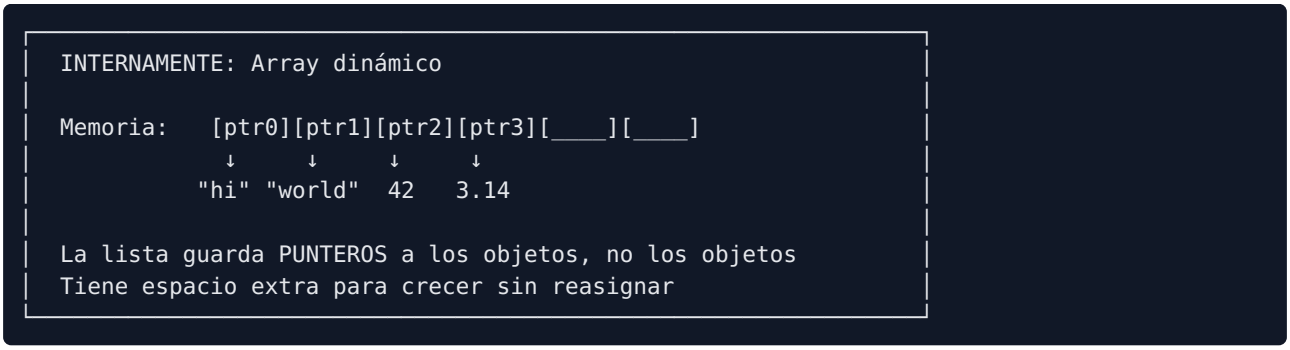


📋 Contenido

---

1. Listas en Python: Bajo Nivel {#1-listas}

1.1 Cómo Funciona una Lista



1.2 Creación y Acceso

```
# Crear listas
words: list[str] = ["hello", "world", "python"]
numbers: list[int] = [1, 2, 3, 4, 5]
mixed: list = [1, "two", 3.0, None]  # Evitar en código tipado

# Acceso por índice: O(1)
first = words[0]      # "hello"
last = words[-1]      # "python" (desde el final)

# Longitud: O(1) (Python guarda el tamaño)
length = len(words)   # 3

# Modificación: O(1)
words[0] = "hi"       # ["hi", "world", "python"]
```

1.3 Agregar y Eliminar

```
words = ["a", "b", "c"]

# Agregar al final: O(1) amortizado
words.append("d")     # ["a", "b", "c", "d"]

# Agregar al inicio: O(n) - ¡LENTO!
words.insert(0, "z")  # ["z", "a", "b", "c", "d"]
# Todos los elementos deben moverse

# Extender con otra lista: O(k) donde k = len(otra_lista)
words.extend(["e", "f"])  # ["z", "a", "b", "c", "d", "e", "f"]

# Eliminar del final: O(1)
last = words.pop()    # Retorna "f", words = ["z", "a", "b", "c", "d", "e"]

# Eliminar del inicio: O(n) - ¡LENTO!
first = words.pop(0)  # Retorna "z", todos deben moverse

# Eliminar por valor: O(n) - busca y luego mueve
words.remove("c")     # Busca "c" y lo elimina
```

---

2. Slicing y Copias {#2-slicing}

2.1 Slicing Básico

```
nums = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# Sintaxis: list[start:stop:step]
nums[2:5]      # [2, 3, 4]      - desde índice 2 hasta 5 (no incluido)
nums[:3]       # [0, 1, 2]      - desde inicio hasta 3
nums[7:]       # [7, 8, 9]      - desde 7 hasta el final
nums[::2]      # [0, 2, 4, 6, 8] - cada 2 elementos
nums[::-1]     # [9, 8, ..., 0]  - reverso

# Índices negativos
nums[-3:]      # [7, 8, 9]      - últimos 3
nums[:-2]      # [0, 1, ..., 7] - todos menos últimos 2
```

2.2 Copia Superficial vs Profunda

```
# ⚠️ ASIGNACIÓN: NO ES COPIA, es alias
original = [1, 2, 3]
alias = original
alias[0] = 99
print(original)  # [99, 2, 3] ¡Original modificado!

# ✅ COPIA SUPERFICIAL: nueva lista, mismos objetos internos
original = [1, 2, 3]
copy1 = original[:]       # Slicing
copy2 = original.copy()   # Método copy
copy3 = list(original)    # Constructor

copy1[0] = 99
print(original)  # [1, 2, 3] ¡Original intacto!

# ⚠️ Con objetos anidados, copia superficial NO es suficiente
nested = [[1, 2], [3, 4]]
shallow = nested.copy()
shallow[0][0] = 99        # Modifica el objeto interno
print(nested)             # [[99, 2], [3, 4]] ¡Modificado!

# ✅ COPIA PROFUNDA: copia todo recursivamente
import copy
nested = [[1, 2], [3, 4]]
deep = copy.deepcopy(nested)
deep[0][0] = 99
print(nested)             # [[1, 2], [3, 4]] ¡Intacto!
```

2.3 Cuándo Importa

```
# ❌ Bug común: modificar lista mientras se itera
def remove_short_words_bad(words: list[str]) -> list[str]:
    for word in words:  # Itera sobre la misma lista
        if len(word) < 3:
            words.remove(word)  # ¡Modifica durante iteración!
    return words

# ✅ Solución 1: crear nueva lista
def remove_short_words_good(words: list[str]) -> list[str]:
    return [w for w in words if len(w) >= 3]

# ✅ Solución 2: iterar sobre copia
def remove_short_words_alt(words: list[str]) -> list[str]:
    for word in words[:]:  # Copia con [:]
        if len(word) < 3:
            words.remove(word)
    return words
```

---

3. Complejidad de Operaciones {#3-complejidad}

3.1 Tabla Completa

Operación	Complejidad	Ejemplo
Acceso `list[i]`	O(1)	`words[5]`
Asignar `list[i] = x`	O(1)	`words[5] = "new"`
`len(list)`	O(1)	`len(words)`

Operación	Complejidad	Ejemplo
`list.append(x)`	O(1)*	`words.append("x")`
`list.pop()`	O(1)	`words.pop()`
`list.insert(0, x)`	O(n)	`words.insert(0, "x")`
`list.pop(0)`	O(n)	`words.pop(0)`
`x in list`	O(n)	`"hello" in words`
`list.index(x)`	O(n)	`words.index("hello")`
`list.count(x)`	O(n)	`words.count("the")`
`list.remove(x)`	O(n)	`words.remove("hello")`
`list.sort()`	O(n log n)	`words.sort()`
Slice `list[a:b]`	O(b-a)	`words[5:10]`
`list.extend(k)`	O(k)	`words.extend(["a","b"])`

*Amortizado: ocasionalmente O(n) cuando se reasigna memoria.

3.2 Implicaciones Prácticas

```
# ❌ Ineficiente: insertar al inicio muchas veces → O(n²) total
def build_reversed_bad(items: list[str]) -> list[str]:
    result = []
    for item in items:
        result.insert(0, item)  # O(n) cada vez
    return result

# ✅ Eficiente: append y luego revertir → O(n) total
def build_reversed_good(items: list[str]) -> list[str]:
    result = []
    for item in items:
        result.append(item)  # O(1) cada vez
    result.reverse()  # O(n) una vez
    return result

# ✅ Más pythonic
def build_reversed_best(items: list[str]) -> list[str]:
    return items[::-1]
```

---

4. Strings: Inmutabilidad {#4-strings}

4.1 Strings Son Inmutables

```
text = "Hello"

# ❌ No puedes modificar un carácter
text[0] = "J"  # TypeError: 'str' object does not support item assignment

# ✅ Debes crear un nuevo string
text = "J" + text[1:]  # "Jello"

# Cada operación crea un NUEVO string
text = "Hello"
text = text + " World"  # Nuevo objeto, no modificación
```

```
text = text.lower()     # Nuevo objeto
text = text.strip()     # Nuevo objeto
```

4.2 Concatenación Eficiente

```
# ❌ Ineficiente: muchas concatenaciones → O(n²)
def build_string_bad(words: list[str]) -> str:
    result = ""
    for word in words:
        result = result + word + " "  # Crea nuevo string cada vez
    return result.strip()

# ✅ Eficiente: join → O(n)
def build_string_good(words: list[str]) -> str:
    return " ".join(words)

# Benchmark con 10,000 palabras:
# build_string_bad:  ~0.1s
# build_string_good: ~0.001s (100x más rápido)
```

4.3 Métodos de String Útiles

```
text = "  Hello, World! How are you?  "

# Limpieza
text.strip()      # "Hello, World! How are you?"
text.lower()      # "  hello, world! how are you?  "
text.upper()      # "  HELLO, WORLD! HOW ARE YOU?  "

# Búsqueda
text.find("World")     # 9 (índice) o -1 si no existe
text.count("o")        # 3
"Hello" in text        # True
text.startswith("  H") # True
text.endswith("?  ")   # True

# División
text.split()           # ["Hello,", "World!", "How", "are", "you?"]
text.split(",")        # ["  Hello", " World! How are you?  "]

# Reemplazo
text.replace("!", "")  # Sin signos de exclamación
text.replace(" ", "_") # Espacios por guiones bajos

# Verificación
"hello".isalpha()      # True (solo letras)
"hello123".isalnum()   # True (letras y números)
"123".isdigit()        # True (solo dígitos)
"   ".isspace()        # True (solo espacios)
```

---

5. Tokenización: Tu Primer Componente {#5-tokenizacion}

5.1 ¿Qué es Tokenización?

```
┌─────────────────────────────────────────────────────────────┐
│  TOKENIZACIÓN = Convertir texto en unidades procesables     │
│                                                             │
│  Entrada:  "Hello, World! How are you?"                     │
│  Salida:   ["hello", "world", "how", "are", "you"]          │
│                                                             │
```

```
|  Pasos típicos:                                                    |
|  1. Convertir a minúsculas                                         |
|  2. Eliminar puntuación                                            |
|  3. Dividir por espacios                                           |
|  4. Filtrar palabras vacías (stop words)                           |
└────────────────────────────────────────────────────────────────────┘
```

5.2 Tokenizador Básico

```
def tokenize_basic(text: str) -> list[str]:
    """Split text into lowercase words.

    Args:
        text: Input text to tokenize.

    Returns:
        List of lowercase tokens.

    Example:
        >>> tokenize_basic("Hello, World!")
        ['hello,', 'world!']
    """
    return text.lower().split()
```

5.3 Tokenizador con Limpieza de Puntuación

```
def remove_punctuation(text: str) -> str:
    """Remove all punctuation from text.

    Uses character-by-character filtering.

    Args:
        text: Text potentially containing punctuation.

    Returns:
        Text with punctuation replaced by spaces.
    """
    result = []
    for char in text:
        if char.isalnum() or char.isspace():
            result.append(char)
        else:
            result.append(' ')  # Reemplazar puntuación por espacio
    return ''.join(result)


def tokenize_clean(text: str) -> list[str]:
    """Tokenize text with punctuation removal.

    Args:
        text: Input text.

    Returns:
        List of clean, lowercase tokens.

    Example:
        >>> tokenize_clean("Hello, World! How are you?")
        ['hello', 'world', 'how', 'are', 'you']
    """
    cleaned = remove_punctuation(text)
    return cleaned.lower().split()
```

5.4 Tokenizador con Stop Words

```
# Stop words comunes en inglés
STOP_WORDS: frozenset[str] = frozenset({
    "a", "an", "the", "and", "or", "but", "is", "are", "was", "were",
    "be", "been", "being", "have", "has", "had", "do", "does", "did",
    "will", "would", "could", "should", "may", "might", "must",
    "i", "you", "he", "she", "it", "we", "they", "me", "him", "her",
    "us", "them", "my", "your", "his", "its", "our", "their",
    "this", "that", "these", "those", "what", "which", "who", "whom",
    "in", "on", "at", "by", "for", "with", "about", "to", "from",
    "of", "as", "if", "then", "than", "so", "no", "not", "only"
})


def tokenize(
    text: str,
    remove_stopwords: bool = True,
    min_length: int = 2
) -> list[str]:
    """Full tokenization pipeline.

    Args:
        text: Input text to tokenize.
        remove_stopwords: Whether to filter out stop words.
        min_length: Minimum token length to keep.

    Returns:
        List of processed tokens.

    Example:
        >>> tokenize("The quick brown fox jumps over the lazy dog.")
        ['quick', 'brown', 'fox', 'jumps', 'over', 'lazy', 'dog']
    """
    # 1. Remove punctuation
    cleaned = remove_punctuation(text)

    # 2. Lowercase and split
    tokens = cleaned.lower().split()

    # 3. Filter by length
    tokens = [t for t in tokens if len(t) >= min_length]

    # 4. Remove stop words
    if remove_stopwords:
        tokens = [t for t in tokens if t not in STOP_WORDS]

    return tokens
```

5.5 Clase Tokenizer (Aplicando OOP)

```
class Tokenizer:
    """Configurable text tokenizer.

    Attributes:
        stop_words: Set of words to filter out.
        min_length: Minimum token length.

    Example:
        >>> tokenizer = Tokenizer()
        >>> tokenizer.tokenize("Hello, World!")
        ['hello', 'world']
```

```
    """

    DEFAULT_STOP_WORDS: frozenset[str] = STOP_WORDS

    def __init__(
        self,
        stop_words: set[str] | None = None,
        min_length: int = 2
    ) -> None:
        """Initialize tokenizer with configuration.

        Args:
            stop_words: Custom stop words (None uses defaults).
            min_length: Minimum token length to keep.
        """
        self.stop_words: frozenset[str] = (
            frozenset(stop_words) if stop_words is not None
            else self.DEFAULT_STOP_WORDS
        )
        self.min_length: int = min_length

    def _remove_punctuation(self, text: str) -> str:
        """Remove punctuation from text."""
        return ''.join(
            c if c.isalnum() or c.isspace() else ' '
            for c in text
        )

    def tokenize(self, text: str) -> list[str]:
        """Tokenize text into clean tokens.

        Args:
            text: Input text.

        Returns:
            List of processed tokens.
        """
        cleaned = self._remove_punctuation(text)
        tokens = cleaned.lower().split()

        return [
            token for token in tokens
            if len(token) >= self.min_length
            and token not in self.stop_words
        ]

    def __repr__(self) -> str:
        return (
            f"Tokenizer(stop_words={len(self.stop_words)} words, "
            f"min_length={self.min_length})"
        )
```

5.6 Análisis de Complejidad

```
┌─────────────────────────────────────────────────────────────┐
│  COMPLEJIDAD DE tokenize(text)                              │
│                                                             │
│  Sea n = len(text), m = número de tokens                    │
│                                                             │
│  1. remove_punctuation: O(n) - recorre cada carácter        │
│  2. lower(): O(n) - recorre cada carácter                   │
│  3. split(): O(n) - recorre buscando espacios               │
```

```
|  4. Filtrar por longitud: O(m) - recorre tokens                |
|  5. Filtrar stop words: O(m) - lookup O(1) por token           |
|                                                                |
|  TOTAL: O(n + m) ≈ O(n) ya que m ≤ n                           |
└────────────────────────────────────────────────────────────────┘
```

⚠️ Errores Comunes

Error 1: Modificar lista durante iteración

```
# ❌ Bug: resultado impredecible
words = ["a", "the", "b", "an", "c"]
for word in words:
    if word in {"the", "an"}:
        words.remove(word)
# Resultado: ["a", "b", "c"] pero puede fallar

# ✅ Correcto: list comprehension
words = [w for w in words if w not in {"the", "an"}]
```

Error 2: Concatenar strings en loop

```
# ❌ O(n²) - crea nuevo string cada vez
result = ""
for word in words:
    result += word + " "

# ✅ O(n) - usa join
result = " ".join(words)
```

Error 3: Olvidar que strings son inmutables

```
# ❌ No hace nada
text = "hello"
text.upper()  # Retorna nuevo string, no modifica
print(text)   # "hello" (sin cambios)

# ✅ Asignar resultado
text = text.upper()
print(text)   # "HELLO"
```

🔧 Ejercicios Prácticos

Ejercicio 4.1: Manipulación de Listas

Ver EJERCICIOS.md

Ejercicio 4.2: Tokenizador Básico

Ver EJERCICIOS.md

Ejercicio 4.3: Análisis de Complejidad

Ver EJERCICIOS.md

📚 Recursos Externos

Recurso	Tipo	Prioridad
Python Lists	Docs	🔴 Obligatorio
String Methods	Docs	🔴 Obligatorio
Time Complexity	Wiki	🟡 Recomendado

🔗 Referencias del Glosario

- Array
- String
- Inmutabilidad
- Tokenización

🧭 Navegación

← Anterior	Índice	Siguiente →
03_LOGICA_DISCRETA	00_INDICE	05_HASHMAPS_SETS

Anexo DSA - Hash Maps y Sets

Guía MS in AI Pathway

DUQUEOM · 2025

Anexo DSA - Hash Maps y Sets

> ⚠️ **MÓDULO OPCIONAL:** Este módulo NO es requerido para el Pathway. Es útil para entrevistas técnicas.
> 🎯 **Objetivo:** Dominar diccionarios y sets en Python.

🧠 Analogía: El Índice de un Libro vs Leer Página por Página

```
┌─────────────────────────────────────────────────────────────────┐
│                                                                 │
│   LISTA = LIBRO SIN ÍNDICE                                      │
│   ─────────────────────────                                     │
│   Para encontrar "recursión" debes leer página por página → O(n)│
│                                                                 │
│   DICCIONARIO = LIBRO CON ÍNDICE ALFABÉTICO                     │
│   ──────────────────────────────────────────                    │
│   Buscas "recursión" en el índice → página 142 → directo → O(1) │
│                                                                 │
│   ¿CÓMO FUNCIONA EL "ÍNDICE"?                                   │
│   ──────────────────────────────                                │
│   HASH FUNCTION: Convierte "recursión" → número → posición en memoria
│   "recursión" → hash() → 7293847 → slot 47 en el array interno  │
│                                                                 │
└─────────────────────────────────────────────────────────────────┘
```

📋 Contenido

---

1. Cómo Funciona un Hash Map {#1-como-funciona}

1.1 La Función Hash

```
┌─────────────────────────────────────────────────────────────┐
│  HASH FUNCTION: Convierte cualquier dato en un número       │
│                                                             │
│  "hello" → hash("hello") → 2314058222102390712              │
│  "world" → hash("world") → 6736076307280336625              │
│                                                             │
│  PROPIEDADES IMPORTANTES:                                   │
│  ✅ Mismo input → siempre mismo output (determinista)       │
│  ✅ Rápido de calcular                                      │
│  ✅ Distribuye bien los valores (pocos "choques")           │
│  ❌ Diferente input puede dar mismo output (colisión)       │
└─────────────────────────────────────────────────────────────┘
```

1.2 Del Hash a la Posición

```
# Internamente, un diccionario es un array
# El hash determina dónde guardar el valor
```

```
def simplified_hash_position(key: str, array_size: int) -> int:
    """Simplified example of how position is calculated.

    Real implementation is more complex.
    """
    hash_value = hash(key)
    position = hash_value % array_size  # Módulo para que quepa
    return position

# Ejemplo conceptual (NO es implementación real)
# dict con 8 slots internos:
# "hello" → hash → 2314058... → 2314058 % 8 = 2 → slot[2]
# "world" → hash → 6736076... → 6736076 % 8 = 1 → slot[1]
```

1.3 Por Qué es O(1)

```
┌──────────────────────────────────────────────────────────┐
│  LISTA: Buscar "hello" en ["world", "python", "hello", ...]  │
│  ──────────────────────────────────────────────────────  │
│  1. Comparar con "world" → NO                            │
│  2. Comparar con "python" → NO                           │
│  3. Comparar con "hello" → SÍ                            │
│  → Peor caso: revisar TODOS los n elementos → O(n)       │
│                                                          │
│  DICCIONARIO: Buscar "hello"                             │
│  ──────────────────────────                              │
│  1. Calcular hash("hello") → 2314058                     │
│  2. Ir directo a slot[2314058 % size]                    │
│  3. Verificar que la clave coincide                      │
│  → Siempre ~3 pasos, sin importar tamaño → O(1)          │
└──────────────────────────────────────────────────────────┘
```

---

2. Diccionarios en Python {#2-diccionarios}

2.1 Creación y Acceso Básico

```
# Crear diccionarios
word_counts: dict[str, int] = {"hello": 5, "world": 3}
empty: dict[str, int] = {}
from_pairs = dict([("a", 1), ("b", 2)])

# Acceso: O(1)
count = word_counts["hello"]  # 5
# word_counts["missing"]  # KeyError!

# Acceso seguro: O(1)
count = word_counts.get("hello")      # 5
count = word_counts.get("missing")    # None
count = word_counts.get("missing", 0) # 0 (default)

# Verificar existencia: O(1)
if "hello" in word_counts:
    print("Found!")

# Asignar: O(1)
word_counts["new"] = 10
word_counts["hello"] = 6  # Sobrescribe
```

2.2 Métodos Importantes

```
word_counts = {"hello": 5, "world": 3, "python": 7}

# Obtener claves, valores, pares
keys = word_counts.keys()       # dict_keys(['hello', 'world', 'python'])
values = word_counts.values()   # dict_values([5, 3, 7])
items = word_counts.items()     # dict_items([('hello', 5), ...])

# Iterar
for word in word_counts:        # Itera sobre claves
    print(word)

for word, count in word_counts.items():
    print(f"{word}: {count}")

# Eliminar: O(1)
del word_counts["hello"]
count = word_counts.pop("world")  # Retorna valor y elimina
count = word_counts.pop("missing", 0)  # Default si no existe

# Actualizar con otro diccionario
word_counts.update({"new": 1, "python": 10})

# setdefault: obtener o insertar default
word_counts.setdefault("java", 0)  # Inserta "java": 0 si no existe
```

2.3 defaultdict: Diccionario con Default Automático

```
from collections import defaultdict

# ❌ Con dict normal, necesitas verificar existencia
word_counts: dict[str, int] = {}
for word in ["a", "b", "a", "c", "a"]:
    if word not in word_counts:
        word_counts[word] = 0
    word_counts[word] += 1

# ✅ Con defaultdict, el default se crea automáticamente
word_counts: defaultdict[str, int] = defaultdict(int)  # int() = 0
for word in ["a", "b", "a", "c", "a"]:
    word_counts[word] += 1  # Si no existe, crea con valor 0

print(dict(word_counts))  # {'a': 3, 'b': 1, 'c': 1}

# defaultdict con lista
index: defaultdict[str, list[int]] = defaultdict(list)
index["hello"].append(1)  # Crea lista vacía si no existe
index["hello"].append(5)
print(dict(index))  # {'hello': [1, 5]}
```

2.4 Counter: Diccionario para Contar

```
from collections import Counter

words = ["apple", "banana", "apple", "cherry", "banana", "apple"]

# Contar frecuencias
counts = Counter(words)
print(counts)  # Counter({'apple': 3, 'banana': 2, 'cherry': 1})
```

```
# Acceso como diccionario
print(counts["apple"])   # 3
print(counts["missing"]) # 0 (no KeyError!)

# Métodos útiles
print(counts.most_common(2))  # [('apple', 3), ('banana', 2)]

# Operaciones matemáticas
more_words = Counter(["apple", "date"])
total = counts + more_words  # Suma conteos
```

---

3. Sets: Conjuntos con Hash {#3-sets}

3.1 Internamente, un Set es un Dict sin Valores

```
┌─────────────────────────────────────────────────────┐
│  SET: Solo almacena las claves, sin valores asociados │
│                                                     │
│  Internamente:                                      │
│  set({"a", "b", "c"}) ≈ {"a": None, "b": None, "c": None} │
│                                                     │
│  Por eso tiene las mismas complejidades O(1) que dict │
└─────────────────────────────────────────────────────┘
```

3.2 Operaciones y Complejidad

```
words: set[str] = {"hello", "world"}

# Agregar: O(1)
words.add("python")

# Verificar: O(1) - ¡Esta es la operación clave!
if "hello" in words:
    print("Found!")

# Eliminar: O(1)
words.remove("hello")     # KeyError si no existe
words.discard("missing")  # No error si no existe

# Operaciones de conjuntos: O(min(len(a), len(b)))
a = {1, 2, 3}
b = {2, 3, 4}
union = a | b          # {1, 2, 3, 4}
intersection = a & b   # {2, 3}
difference = a - b     # {1}
```

3.3 Cuándo Usar Set vs List

Operación	List	Set	Usar Set cuando...
`x in collection`	O(n)	O(1)	Muchas búsquedas
Mantener orden	✅	❌	Orden no importa
Permitir duplicados	✅	❌	Solo necesitas únicos
Acceso por índice	✅	❌	No necesitas índices

```
# ❌ Lento: verificar stop words en lista
stop_words_list = ["the", "a", "an", "and", "or", "but", ...]
```

```
def is_stopword_slow(word: str) -> bool:
    return word in stop_words_list  # O(n) cada vez

# ✅ Rápido: verificar en set
stop_words_set = {"the", "a", "an", "and", "or", "but", ...}

def is_stopword_fast(word: str) -> bool:
    return word in stop_words_set  # O(1) cada vez
```

---

4. Colisiones y Resolución {#4-colisiones}

4.1 ¿Qué es una Colisión?

```
┌─────────────────────────────────────────────────────────┐
│  COLISIÓN: Dos claves diferentes → mismo slot           │
│                                                         │
│  "hello" → hash → 47293 % 8 = 5 → slot[5]               │
│  "world" → hash → 82645 % 8 = 5 → slot[5]  ← ¡MISMO SLOT!│
│                                                         │
│  Python resuelve esto con "open addressing":            │
│  Si slot[5] está ocupado, busca slot[6], slot[7], etc.  │
└─────────────────────────────────────────────────────────┘
```

4.2 Por Qué Sigue Siendo O(1)

```
Python mantiene el diccionario "poco lleno" (load factor < 2/3)
Cuando se llena demasiado, lo hace más grande y redistribuye

Con buen factor de carga:
- Promedio: 1-2 comparaciones por búsqueda → O(1) amortizado
- Peor caso (muy raro): O(n) si todas las claves colisionan
```

4.3 Qué Puede Ser Clave de Diccionario

```
# ✅ HASHABLE: tipos inmutables
d = {}
d["string"] = 1        # str: OK
d[42] = 2              # int: OK
d[3.14] = 3            # float: OK
d[(1, 2, 3)] = 4       # tuple: OK
d[frozenset({1,2})] = 5  # frozenset: OK

# ❌ NO HASHABLE: tipos mutables
# d[[1, 2, 3]] = 6     # TypeError: unhashable type: 'list'
# d[{1, 2}] = 7        # TypeError: unhashable type: 'set'
# d[{"a": 1}] = 8      # TypeError: unhashable type: 'dict'

# ¿Por qué? Si el objeto cambia, su hash cambiaría
# y no lo encontraríamos donde lo guardamos
```

---

5. Aplicación: Contador de Frecuencias {#5-aplicacion}

5.1 Contador Manual

```
def count_word_frequencies(tokens: list[str]) -> dict[str, int]:
    """Count frequency of each word in token list.

    Args:
```

```
        tokens: List of words to count.

    Returns:
        Dictionary mapping words to their counts.

    Complexity:
        O(n) where n = len(tokens)

    Example:
        >>> count_word_frequencies(["a", "b", "a"])
        {'a': 2, 'b': 1}
    """
    frequencies: dict[str, int] = {}

    for token in tokens:
        # O(1) lookup + O(1) assignment
        frequencies[token] = frequencies.get(token, 0) + 1

    return frequencies
```

5.2 Con defaultdict

```
from collections import defaultdict

def count_frequencies_defaultdict(tokens: list[str]) -> dict[str, int]:
    """Count frequencies using defaultdict.

    Cleaner than manual .get() approach.
    """
    frequencies: defaultdict[str, int] = defaultdict(int)

    for token in tokens:
        frequencies[token] += 1

    return dict(frequencies)
```

5.3 Con Counter (Una Línea)

```
from collections import Counter

def count_frequencies_counter(tokens: list[str]) -> dict[str, int]:
    """Count frequencies using Counter.

    Most Pythonic approach.
    """
    return dict(Counter(tokens))
```

5.4 Benchmark Comparativo

```
import time
from collections import Counter, defaultdict

def benchmark_frequency_counters(tokens: list[str]) -> None:
    """Compare performance of different counting methods."""

    # Method 1: Manual with .get()
    start = time.time()
    freq = {}
    for t in tokens:
        freq[t] = freq.get(t, 0) + 1
    manual_time = time.time() - start
```

```
    # Method 2: defaultdict
    start = time.time()
    freq = defaultdict(int)
    for t in tokens:
        freq[t] += 1
    defaultdict_time = time.time() - start

    # Method 3: Counter
    start = time.time()
    freq = Counter(tokens)
    counter_time = time.time() - start

    print(f"Manual:      {manual_time:.4f}s")
    print(f"defaultdict: {defaultdict_time:.4f}s")
    print(f"Counter:     {counter_time:.4f}s")

# Con 1,000,000 tokens:
# Manual:      0.0800s
# defaultdict: 0.0750s
# Counter:     0.0650s  ← Más rápido (implementado en C)
```

5.5 Construyendo hacia el Índice Invertido

```
from collections import defaultdict

def build_term_document_map(
    documents: list[tuple[int, list[str]]]
) -> dict[str, set[int]]:
    """Build mapping from terms to document IDs.

    This is the core of an inverted index.

    Args:
        documents: List of (doc_id, tokens) pairs.

    Returns:
        Dictionary mapping each term to set of doc IDs containing it.

    Example:
        >>> docs = [(1, ["hello", "world"]), (2, ["hello", "python"])]
        >>> build_term_document_map(docs)
        {'hello': {1, 2}, 'world': {1}, 'python': {2}}
    """
    term_to_docs: defaultdict[str, set[int]] = defaultdict(set)

    for doc_id, tokens in documents:
        for token in tokens:
            term_to_docs[token].add(doc_id)

    return dict(term_to_docs)
```

⚠️ Errores Comunes

Error 1: Modificar dict mientras iteras

```
# ❌ RuntimeError: dictionary changed size during iteration
word_counts = {"a": 1, "b": 2, "c": 3}
for word in word_counts:
    if word_counts[word] < 2:
```

```
        del word_counts[word]

# ✅ Iterar sobre copia de claves
for word in list(word_counts.keys()):
    if word_counts[word] < 2:
        del word_counts[word]

# ✅ O crear nuevo diccionario
word_counts = {w: c for w, c in word_counts.items() if c >= 2}
```

Error 2: Asumir orden en versiones antiguas

```
# Python 3.7+: dict mantiene orden de inserción
# Python < 3.7: NO garantiza orden

# Si necesitas orden garantizado, usa:
from collections import OrderedDict
```

Error 3: Usar objeto mutable como clave

```
# ❌ TypeError
cache = {}
cache[[1, 2, 3]] = "result"  # Lista no es hashable

# ✅ Convertir a tupla
cache[tuple([1, 2, 3])] = "result"
```

🔧 Ejercicios Prácticos

Ejercicio 5.1: Contador de Frecuencias

Ver EJERCICIOS.md

Ejercicio 5.2: Benchmark List vs Set

Ver EJERCICIOS.md

Ejercicio 5.3: Term-Document Map

Ver EJERCICIOS.md

📚 Recursos Externos

Recurso	Tipo	Prioridad
Python Dict Implementation	Video	🟡 Recomendado
Time Complexity	Wiki	🔴 Obligatorio
collections Module	Docs	🟡 Recomendado

🔗 Referencias del Glosario

- Hash Map
- Hash Function
- Colisión
- Set

- O(1) Amortizado

🧭 Navegación

← Anterior	Índice	Siguiente →
04_ARRAYS_STRINGS	00_INDICE	06_INVERTED_INDEX

Anexo DSA - Recursión

Guía MS in AI Pathway

DUQUEOM · 2025

Anexo DSA - Recursión y Divide & Conquer

> ⚠️ **MÓDULO OPCIONAL:** Este módulo NO es requerido para el Pathway. Es útil para entrevistas técnicas.
> 🎯 **Objetivo:** Dominar el pensamiento recursivo.

🧠 Analogía: Las Muñecas Rusas (Matryoshkas)

```
┌─────────────────────────────────────────────────────────────┐
│                                                             │
│   RECURSIÓN = Resolver un problema resolviéndolo para una versión menor
│   ──────────────────────────────────────────────────────────
│                                                             │
│   Muñecas Rusas:                                            │
│                                                             │
│   ┌─────────┐                                               │
│   │ ┌─────┐ │                                               │
│   │ │ ┌─┐ │ │   ← Caso base: la muñeca más pequeña (sólida)
│   │ │ │●│ │ │                                               │
│   │ │ └─┘ │ │   ← Cada muñeca "contiene" una versión menor  │
│   │ └─────┘ │                                               │
│   └─────────┘                                               │
│                                                             │
│   Para abrir TODAS las muñecas:                             │
│   1. ¿Es la muñeca sólida? → PARAR (caso base)              │
│   2. Si no, abrir esta muñeca y REPETIR con la de adentro   │
│                                                             │
└─────────────────────────────────────────────────────────────┘
```

📋 Contenido

---

1. ¿Qué es Recursión? {#1-que-es}

1.1 Definición

```
┌─────────────────────────────────────────────────────────────┐
│  RECURSIÓN: Una función que se llama a sí misma             │
│                                                             │
│  def funcion():                                             │
│      ...                                                    │
│      funcion()  ← Se llama a sí misma                       │
│      ...                                                    │
│                                                             │
│  ⚠️ Sin condición de parada → recursión infinita → crash    │
└─────────────────────────────────────────────────────────────┘
```

1.2 ¿Por Qué Usar Recursión?

```
PROBLEMAS NATURALMENTE RECURSIVOS:
──────────────────────────────────

1. Estructuras de datos recursivas
   - Árboles: un nodo tiene hijos que son árboles
   - Listas enlazadas: una lista es un nodo + otra lista
   - Sistemas de archivos: carpetas contienen carpetas

2. Problemas que se reducen a versiones menores
   - Factorial: n! = n × (n-1)!
   - Fibonacci: fib(n) = fib(n-1) + fib(n-2)
   - Ordenamiento: ordenar lista = ordenar sublistas + combinar
```

---

2. Caso Base y Caso Recursivo {#2-casos}

2.1 Los Dos Ingredientes Esenciales

```
def recursive_function(problem):
    # 1. CASO BASE: problema tan pequeño que se resuelve directamente
    if problem_is_trivial(problem):
        return trivial_solution

    # 2. CASO RECURSIVO: reducir el problema y llamar recursivamente
    smaller_problem = reduce(problem)
    return combine(recursive_function(smaller_problem))
```

2.2 Ejemplo: Factorial

```
def factorial(n: int) -> int:
    """Calculate n! = n × (n-1) × (n-2) × ... × 1

    Base case: 0! = 1
    Recursive: n! = n × (n-1)!

    Example:
        >>> factorial(5)
        120  # 5 × 4 × 3 × 2 × 1
    """
    # Caso base
    if n <= 1:
        return 1

    # Caso recursivo
    return n * factorial(n - 1)


# Traza de ejecución:
# factorial(4)
#   → 4 * factorial(3)
#       → 3 * factorial(2)
#           → 2 * factorial(1)
#               → 1  (caso base)
#           → 2 * 1 = 2
#       → 3 * 2 = 6
#   → 4 * 6 = 24
```

2.3 Ejemplo: Suma de Lista

```
def sum_list(numbers: list[int]) -> int:
    """Sum all numbers in list using recursion.

    Base case: empty list → 0
    Recursive: sum = first + sum(rest)

    Example:
        >>> sum_list([1, 2, 3, 4])
        10
    """
    # Caso base: lista vacía
    if not numbers:
        return 0

    # Caso recursivo: primer elemento + suma del resto
    return numbers[0] + sum_list(numbers[1:])


# Alternativa más eficiente (evita crear sublistas)
def sum_list_efficient(numbers: list[int], index: int = 0) -> int:
    """Sum using index instead of slicing."""
    # Caso base: índice fuera de rango
    if index >= len(numbers):
        return 0

    # Caso recursivo
    return numbers[index] + sum_list_efficient(numbers, index + 1)
```

---

3. El Call Stack {#3-call-stack}

3.1 Visualización del Stack

```
┌─────────────────────────────────────────────────────────────┐
│  CALL STACK: Pila de llamadas a funciones                   │
│                                                             │
│  Cada llamada recursiva agrega un "frame" al stack          │
│  Cuando termina, se "desapila" y retorna al anterior        │
│                                                             │
│  factorial(4):                                              │
│                                                             │
│  LLAMANDO (stack crece →)          RETORNANDO (stack decrece ←) │
│                                                             │
│   ┌──────────────────┐              ┌──────────────────┐    │
│   │ factorial(1) = 1 │ ←base       │ factorial(1) = 1 │ →return  │
│   ├──────────────────┤              ├──────────────────┤    │
│   │ factorial(2)     │              │ factorial(2) = 2 │ →return  │
│   ├──────────────────┤              ├──────────────────┤    │
│   │ factorial(3)     │              │ factorial(3) = 6 │ →return  │
│   ├──────────────────┤              ├──────────────────┤    │
│   │ factorial(4)     │              │ factorial(4) = 24│ →return  │
│   └──────────────────┘              └──────────────────┘    │
└─────────────────────────────────────────────────────────────┘
```

3.2 Límite de Recursión

```
import sys

# Python tiene un límite por defecto
```

```
print(sys.getrecursionlimit())  # 1000 (típicamente)

# Excederlo causa RecursionError
def infinite_recursion():
    return infinite_recursion()

# infinite_recursion()  # RecursionError: maximum recursion depth exceeded

# Puedes aumentar el límite (con cuidado)
sys.setrecursionlimit(2000)
```

3.3 Visualizar la Recursión

```
def factorial_verbose(n: int, depth: int = 0) -> int:
    """Factorial with execution trace."""
    indent = "  " * depth
    print(f"{indent}factorial({n})")

    if n <= 1:
        print(f"{indent}→ returning 1 (base case)")
        return 1

    result = n * factorial_verbose(n - 1, depth + 1)
    print(f"{indent}→ returning {n} * ... = {result}")
    return result

# factorial_verbose(4) muestra:
# factorial(4)
#   factorial(3)
#     factorial(2)
#       factorial(1)
#       → returning 1 (base case)
#     → returning 2 * ... = 2
#   → returning 3 * ... = 6
# → returning 4 * ... = 24
```

---

4. Ejemplos Clásicos {#4-ejemplos}

4.1 Fibonacci

```
def fibonacci(n: int) -> int:
    """Calculate nth Fibonacci number.

    Sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, ...

    Base cases: fib(0) = 0, fib(1) = 1
    Recursive: fib(n) = fib(n-1) + fib(n-2)

    ⚠️ This naive version is O(2^n) - very slow!
    See memoization section for optimization.
    """
    if n <= 0:
        return 0
    if n == 1:
        return 1

    return fibonacci(n - 1) + fibonacci(n - 2)
```

4.2 Búsqueda en Lista

```
def search_recursive(
    items: list[any],
    target: any,
    index: int = 0
) -> int:
    """Search for target in list, return index or -1.

    Base cases:
    - Index out of bounds → not found (-1)
    - Found target → return index

    Recursive: check next index
    """
    # Caso base: fin de lista
    if index >= len(items):
        return -1

    # Caso base: encontrado
    if items[index] == target:
        return index

    # Caso recursivo: buscar en el resto
    return search_recursive(items, target, index + 1)
```

4.3 Contar Ocurrencias

```
def count_occurrences(items: list[any], target: any) -> int:
    """Count how many times target appears in list.

    Base case: empty list → 0
    Recursive: (1 if first matches else 0) + count(rest)
    """
    if not items:
        return 0

    first_match = 1 if items[0] == target else 0
    return first_match + count_occurrences(items[1:], target)
```

4.4 Invertir String

```
def reverse_string(s: str) -> str:
    """Reverse a string recursively.

    Base case: empty or single char → return as is
    Recursive: last char + reverse(rest)

    Example:
        >>> reverse_string("hello")
        'olleh'
    """
    if len(s) <= 1:
        return s

    return s[-1] + reverse_string(s[:-1])
```

4.5 Palíndromo

```
def is_palindrome(s: str) -> bool:
    """Check if string is a palindrome.

    Base cases:
    - Length 0 or 1 → True
    - First != Last → False

    Recursive: check first == last, then inner string

    Example:
        >>> is_palindrome("radar")
        True
    """
    # Normalizar: quitar espacios y minúsculas
    s = s.lower().replace(" ", "")

    if len(s) <= 1:
        return True

    if s[0] != s[-1]:
        return False

    return is_palindrome(s[1:-1])
```

---

5. Divide & Conquer {#5-divide-conquer}

5.1 El Patrón

```
┌─────────────────────────────────────────────────────────┐
│  DIVIDE & CONQUER (Divide y Vencerás)                   │
│                                                         │
│  1. DIVIDIR: Partir el problema en subproblemas más pequeños    │
│  2. CONQUISTAR: Resolver cada subproblema (recursivamente)      │
│  3. COMBINAR: Unir las soluciones parciales                     │
│                                                         │
│  Ejemplos clásicos:                                     │
│  - MergeSort: dividir lista, ordenar mitades, combinar          │
│  - QuickSort: particionar, ordenar particiones                  │
│  - Binary Search: buscar en mitad correcta                      │
└─────────────────────────────────────────────────────────┘
```

5.2 Merge Sort (Ejemplo Perfecto)

```
def merge_sort(items: list[int]) -> list[int]:
    """Sort list using merge sort algorithm.

    Divide: split list in half
    Conquer: recursively sort each half
    Combine: merge sorted halves

    Complexity: O(n log n) always
    """
    # Base case: 0 or 1 elements already sorted
    if len(items) <= 1:
        return items

    # DIVIDE: split in half
    mid = len(items) // 2
```

```
    left = items[:mid]
    right = items[mid:]

    # CONQUER: sort each half recursively
    left_sorted = merge_sort(left)
    right_sorted = merge_sort(right)

    # COMBINE: merge sorted halves
    return merge(left_sorted, right_sorted)


def merge(left: list[int], right: list[int]) -> list[int]:
    """Merge two sorted lists into one sorted list.

    Uses two-pointer technique.
    Complexity: O(n + m)
    """
    result = []
    i = j = 0

    # Compare elements from both lists
    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1

    # Add remaining elements
    result.extend(left[i:])
    result.extend(right[j:])

    return result
```

5.3 Visualización de Merge Sort

```
┌──────────────────────────────────────────────────────────┐
│  merge_sort([38, 27, 43, 3, 9, 82, 10])                  │
│                                                          │
│  DIVIDIR:                                                │
│                    [38, 27, 43, 3, 9, 82, 10]            │
│                             /            \               │
│                [38, 27, 43]                [3, 9, 82, 10]│
│                 /       \                  /        \    │
│            [38, 27]    [43]            [3, 9]    [82, 10]│
│             /    \                     /   \      /    \ │
│           [38]  [27]                [3]   [9]  [82]  [10]│
│                                                          │
│  COMBINAR (merge):                                       │
│           [27, 38] ← merge [38],[27]   [3, 9] [10, 82]   │
│                  \    /                    \    /       │
│              [27, 38, 43]              [3, 9, 10, 82]    │
│                       \                  /               │
│                  [3, 9, 10, 27, 38, 43, 82]              │
└──────────────────────────────────────────────────────────┘
```

5.4 Máximo de Lista (Divide & Conquer)

```
def find_max_dc(items: list[int]) -> int:
    """Find maximum using divide and conquer.
```

```
    Base cases:
    - Single element → that element
    - Two elements → larger of the two

    Recursive: max of (max left half, max right half)
    """
    if len(items) == 0:
        raise ValueError("Cannot find max of empty list")

    if len(items) == 1:
        return items[0]

    if len(items) == 2:
        return items[0] if items[0] > items[1] else items[1]

    mid = len(items) // 2
    left_max = find_max_dc(items[:mid])
    right_max = find_max_dc(items[mid:])

    return left_max if left_max > right_max else right_max
```

---

6. Optimización con Memoization {#6-memoization}

6.1 El Problema con Fibonacci Naive

```
┌─────────────────────────────────────────────────────────────┐
│  fib(5) calcula fib(3) DOS veces, fib(2) TRES veces, etc.   │
│                                                             │
│                      fib(5)                                 │
│                    /       \                                │
│               fib(4)        fib(3)        ← fib(3) calculado 2x │
│              /     \        /    \                         │
│          fib(3)  fib(2)  fib(2) fib(1)   ← fib(2) calculado 3x  │
│          /   \                                              │
│      fib(2) fib(1)                                          │
│                                                             │
│  Complejidad: O(2^n) - ¡Exponencial!                        │
└─────────────────────────────────────────────────────────────┘
```

6.2 Memoization: Recordar Resultados

```
def fibonacci_memo(n: int, cache: dict[int, int] | None = None) -> int:
    """Fibonacci with memoization.

    Cache stores already computed values to avoid redundant work.

    Complexity: O(n) time, O(n) space
    """
    if cache is None:
        cache = {}

    # Check cache first
    if n in cache:
        return cache[n]

    # Base cases
    if n <= 0:
        return 0
    if n == 1:
        return 1
```

```
    # Compute and cache
    result = fibonacci_memo(n - 1, cache) + fibonacci_memo(n - 2, cache)
    cache[n] = result

    return result


# Comparación de tiempos:
# fibonacci(35)      → ~3 segundos
# fibonacci_memo(35) → <0.001 segundos
```

6.3 Usando functools.lru_cache

```
from functools import lru_cache

@lru_cache(maxsize=None)  # Cache ilimitado
def fibonacci_cached(n: int) -> int:
    """Fibonacci with automatic memoization."""
    if n <= 0:
        return 0
    if n == 1:
        return 1
    return fibonacci_cached(n - 1) + fibonacci_cached(n - 2)


# Ver estadísticas del cache
print(fibonacci_cached.cache_info())
# CacheInfo(hits=48, misses=51, maxsize=None, currsize=51)

# Limpiar cache
fibonacci_cached.cache_clear()
```

⚠️ Errores Comunes

Error 1: Olvidar el caso base

```
# ❌ Sin caso base → RecursionError
def countdown_bad(n):
    print(n)
    countdown_bad(n - 1)  # Nunca termina

# ✅ Con caso base
def countdown_good(n):
    if n <= 0:
        print("Done!")
        return
    print(n)
    countdown_good(n - 1)
```

Error 2: No reducir el problema

```
# ❌ El problema no se reduce
def broken_sum(items):
    if not items:
        return 0
    return items[0] + broken_sum(items)  # Misma lista!

# ✅ Reducir correctamente
def working_sum(items):
```

```
    if not items:
        return 0
    return items[0] + working_sum(items[1:])  # Lista más corta
```

Error 3: Crear copias innecesarias

```
# ❌ Ineficiente: crea nueva lista cada vez
def sum_slow(items):
    if not items:
        return 0
    return items[0] + sum_slow(items[1:])  # items[1:] crea copia

# ✅ Eficiente: usar índice
def sum_fast(items, index=0):
    if index >= len(items):
        return 0
    return items[index] + sum_fast(items, index + 1)
```

🔧 Ejercicios Prácticos

Ejercicio 7.1: Factorial y Fibonacci

Ver EJERCICIOS.md

Ejercicio 7.2: Suma y Máximo Recursivos

Ver EJERCICIOS.md

Ejercicio 7.3: Merge de Listas Ordenadas

Ver EJERCICIOS.md

📚 Recursos Externos

Recurso	Tipo	Prioridad
Grokking Algorithms Ch.3-4	Libro	🔴 Obligatorio
Recursion Visualizer	Herramienta	🟡 Recomendado
MIT Divide & Conquer	Curso	🟢 Complementario

🔗 Referencias del Glosario

- Recursión
- Caso Base
- Call Stack
- Divide & Conquer
- Memoization

🧭 Navegación

← Anterior	Índice	Siguiente →
06_INVERTED_INDEX	00_INDICE	08_SORTING

Anexo DSA - Ordenamiento

Guía MS in AI Pathway

DUQUEOM · 2025

Anexo DSA - Algoritmos de Ordenamiento

> ⚠️ **MÓDULO OPCIONAL:** Este módulo NO es requerido para el Pathway. Es útil para entrevistas técnicas.
> 🎯 **Objetivo:** Implementar QuickSort y MergeSort desde cero.

🧠 Analogía: Ordenando Cartas

```
┌─────────────────────────────────────────────────────────────┐
│                                                             │
│   QUICKSORT = El método del "pivote"                        │
│   ──────────────────────────────────                        │
│                                                             │
│   1. Elige una carta (pivote): por ejemplo, el 7            │
│   2. Separa: menores a la izquierda, mayores a la derecha   │
│   3. Ahora el 7 está en su lugar correcto                   │
│   4. Repite con cada grupo                                  │
│                                                             │
│   [3, 8, 2, 7, 1, 9, 4]  → pivote = 7                       │
│   [3, 2, 1, 4] [7] [8, 9]  → 7 en su lugar                  │
│   Repetir para [3,2,1,4] y [8,9]                            │
│                                                             │
│   MERGESORT = El método de "dividir y fusionar"             │
│   ─────────────────────────────────────────                 │
│                                                             │
│   1. Divide el mazo en dos mitades                          │
│   2. Ordena cada mitad (recursivamente)                     │
│   3. Fusiona las dos mitades ordenadas                      │
│                                                             │
└─────────────────────────────────────────────────────────────┘
```

📋 Contenido

---

1. Por Qué Importan los Algoritmos de Sorting {#1-importancia}

1.1 Sorting es Fundamental

```
┌─────────────────────────────────────────────────────────────┐
│  APLICACIONES DE SORTING                                    │
│                                                             │
│  • Búsqueda binaria: requiere datos ordenados               │
│  • Ranking de resultados: ordenar por relevancia            │
│  • Eliminación de duplicados: ordenar + recorrer            │
│  • Mediana, percentiles: ordenar + acceder por índice       │
│  • Sistemas de bases de datos: índices ordenados            │
│                                                             │
│  EN ARCHIMEDES INDEXER:                                     │
```

```
┌──────────────────────────────────────────────────────┐
│  Ordenar resultados de búsqueda por score de relevancia  │
└──────────────────────────────────────────────────────┘
```

1.2 Complejidades de Referencia

Algoritmo	Mejor	Promedio	Peor	Espacio
Bubble Sort	O(n)	O(n²)	O(n²)	O(1)
Selection Sort	O(n²)	O(n²)	O(n²)	O(1)
Insertion Sort	O(n)	O(n²)	O(n²)	O(1)
QuickSort	O(n log n)	O(n log n)	O(n²)	O(log n)
MergeSort	O(n log n)	O(n log n)	O(n log n)	O(n)
Python's Timsort	O(n)	O(n log n)	O(n log n)	O(n)

---

2. QuickSort: El Favorito en la Práctica {#2-quicksort}

2.1 El Algoritmo

```
┌──────────────────────────────────────────────────────┐
│  QUICKSORT - Pasos:                                  │
│                                                      │
│  1. Si la lista tiene 0 o 1 elementos, ya está ordenada  │
│  2. Elegir un PIVOTE (elemento de referencia)        │
│  3. PARTICIONAR: reorganizar para que:               │
│     - Elementos < pivote queden a la izquierda       │
│     - Elementos >= pivote queden a la derecha        │
│  4. Recursivamente ordenar izquierda y derecha       │
│  5. Concatenar: izquierda + pivote + derecha         │
└──────────────────────────────────────────────────────┘
```

2.2 Implementación Básica (Fácil de Entender)

```
def quicksort_simple(items: list[int]) -> list[int]:
    """QuickSort with simple partitioning.

    This version creates new lists (not in-place).
    Easier to understand but uses more memory.

    Complexity:
        Time: O(n log n) average, O(n²) worst
        Space: O(n) for new lists

    Example:
        >>> quicksort_simple([3, 1, 4, 1, 5, 9, 2, 6])
        [1, 1, 2, 3, 4, 5, 6, 9]
    """
    # Base case: already sorted
    if len(items) <= 1:
        return items

    # Choose pivot (last element for simplicity)
    pivot = items[-1]

    # Partition into three groups
    less = [x for x in items[:-1] if x < pivot]
    equal = [x for x in items if x == pivot]
    greater = [x for x in items[:-1] if x > pivot]
```

```
    # Recursively sort and concatenate
    return quicksort_simple(less) + equal + quicksort_simple(greater)
```

2.3 Implementación In-Place (Eficiente en Memoria)

```
def quicksort(items: list[int]) -> list[int]:
    """QuickSort with in-place partitioning.

    Modifies the original list.

    Returns:
        The same list, now sorted.
    """
    _quicksort_helper(items, 0, len(items) - 1)
    return items


def _quicksort_helper(items: list[int], low: int, high: int) -> None:
    """Recursive helper for in-place quicksort."""
    if low < high:
        # Partition and get pivot position
        pivot_index = _partition(items, low, high)

        # Recursively sort elements before and after partition
        _quicksort_helper(items, low, pivot_index - 1)
        _quicksort_helper(items, pivot_index + 1, high)


def _partition(items: list[int], low: int, high: int) -> int:
    """Partition array around pivot (last element).

    Lomuto partition scheme.

    Returns:
        Final position of pivot.
    """
    pivot = items[high]
    i = low - 1  # Index of smaller element

    for j in range(low, high):
        if items[j] < pivot:
            i += 1
            items[i], items[j] = items[j], items[i]

    # Place pivot in correct position
    items[i + 1], items[high] = items[high], items[i + 1]
    return i + 1
```

2.4 Visualización de Partición

```
Inicial: [8, 3, 1, 7, 0, 10, 2]  (pivot = 2)

j=0: 8 < 2? NO  → [8, 3, 1, 7, 0, 10, 2]  i=-1
j=1: 3 < 2? NO  → [8, 3, 1, 7, 0, 10, 2]  i=-1
j=2: 1 < 2? SÍ  → [1, 3, 8, 7, 0, 10, 2]  i=0 (swap 8↔1)
j=3: 7 < 2? NO  → [1, 3, 8, 7, 0, 10, 2]  i=0
j=4: 0 < 2? SÍ  → [1, 0, 8, 7, 3, 10, 2]  i=1 (swap 3↔0)
j=5: 10< 2? NO  → [1, 0, 8, 7, 3, 10, 2]  i=1

Final: colocar pivot en i+1=2
       [1, 0, 2, 7, 3, 10, 8]
```

```
              ↑ pivot en posición correcta

Izquierda: [1, 0] (todos < 2)
Derecha:   [7, 3, 10, 8] (todos > 2)
```

2.5 Random Pivot (Evitar O(n²))

```
import random


def quicksort_random(items: list[int]) -> list[int]:
    """QuickSort with random pivot selection.

    Random pivot prevents worst case O(n²) on sorted input.
    """
    _quicksort_random_helper(items, 0, len(items) - 1)
    return items


def _quicksort_random_helper(items: list[int], low: int, high: int) -> None:
    if low < high:
        pivot_index = _partition_random(items, low, high)
        _quicksort_random_helper(items, low, pivot_index - 1)
        _quicksort_random_helper(items, pivot_index + 1, high)


def _partition_random(items: list[int], low: int, high: int) -> int:
    """Partition with random pivot."""
    # Choose random pivot and swap to end
    random_index = random.randint(low, high)
    items[random_index], items[high] = items[high], items[random_index]

    return _partition(items, low, high)
```

---

3. MergeSort: Estable y Predecible {#3-mergesort}

3.1 El Algoritmo

```
┌─────────────────────────────────────────────────────────┐
│  MERGESORT - Pasos:                                     │
│                                                         │
│  1. Si la lista tiene 0 o 1 elementos, ya está ordenada │
│  2. DIVIDIR: partir la lista en dos mitades             │
│  3. CONQUISTAR: ordenar cada mitad recursivamente       │
│  4. COMBINAR: fusionar las dos mitades ordenadas        │
│                                                         │
│  La "magia" está en el paso de MERGE:                   │
│  - Dos listas ordenadas se pueden fusionar en O(n)      │
└─────────────────────────────────────────────────────────┘
```

3.2 Implementación Completa

```
def mergesort(items: list[int]) -> list[int]:
    """Sort list using merge sort algorithm.

    Creates new lists (not in-place).

    Complexity:
        Time: O(n log n) always
        Space: O(n) for temporary arrays
```

```
    Example:
        >>> mergesort([3, 1, 4, 1, 5, 9, 2, 6])
        [1, 1, 2, 3, 4, 5, 6, 9]
    """
    # Base case
    if len(items) <= 1:
        return items.copy()

    # Divide
    mid = len(items) // 2
    left = items[:mid]
    right = items[mid:]

    # Conquer (recursively sort)
    left_sorted = mergesort(left)
    right_sorted = mergesort(right)

    # Combine (merge)
    return _merge(left_sorted, right_sorted)


def _merge(left: list[int], right: list[int]) -> list[int]:
    """Merge two sorted lists into one sorted list.

    Uses two-pointer technique.

    Complexity: O(n + m) where n, m are list lengths
    """
    result = []
    i = j = 0

    # Compare elements from both lists
    while i < len(left) and j < len(right):
        if left[i] <= right[j]:  # <= makes it stable
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1

    # Add remaining elements (one list is exhausted)
    result.extend(left[i:])
    result.extend(right[j:])

    return result
```

3.3 Visualización de Merge

```
Fusionar [1, 3, 5] con [2, 4, 6]:

i=0, j=0: 1 vs 2 → tomar 1    result=[1]
i=1, j=0: 3 vs 2 → tomar 2    result=[1, 2]
i=1, j=1: 3 vs 4 → tomar 3    result=[1, 2, 3]
i=2, j=1: 5 vs 4 → tomar 4    result=[1, 2, 3, 4]
i=2, j=2: 5 vs 6 → tomar 5    result=[1, 2, 3, 4, 5]
i=3, j=2: (left agotada)      result=[1, 2, 3, 4, 5, 6]

Final: [1, 2, 3, 4, 5, 6]
```

3.4 MergeSort In-Place (Opcional, Más Complejo)

```
def mergesort_inplace(items: list[int]) -> list[int]:
    """In-place merge sort using auxiliary array.

    More memory efficient than creating many small lists.
    """
    aux = items.copy()
    _mergesort_inplace_helper(items, aux, 0, len(items) - 1)
    return items


def _mergesort_inplace_helper(
    items: list[int],
    aux: list[int],
    low: int,
    high: int
) -> None:
    if low >= high:
        return

    mid = (low + high) // 2
    _mergesort_inplace_helper(items, aux, low, mid)
    _mergesort_inplace_helper(items, aux, mid + 1, high)
    _merge_inplace(items, aux, low, mid, high)


def _merge_inplace(
    items: list[int],
    aux: list[int],
    low: int,
    mid: int,
    high: int
) -> None:
    # Copy to auxiliary array
    for k in range(low, high + 1):
        aux[k] = items[k]

    i = low
    j = mid + 1

    for k in range(low, high + 1):
        if i > mid:
            items[k] = aux[j]
            j += 1
        elif j > high:
            items[k] = aux[i]
            i += 1
        elif aux[j] < aux[i]:
            items[k] = aux[j]
            j += 1
        else:
            items[k] = aux[i]
            i += 1
```

---

4. Comparación y Cuándo Usar Cada Uno {#4-comparacion}

4.1 Tabla Comparativa

Aspecto	QuickSort	MergeSort
Complejidad promedio	O(n log n)	O(n log n)
Peor caso	O(n²)	O(n log n)
Espacio	O(log n)	O(n)
Estable	❌ No	✅ Sí
In-place	✅ Sí	❌ No (típicamente)
Cache-friendly	✅ Mejor	❌ Peor

4.2 ¿Qué Significa "Estable"?

```
# Elementos con mismo valor mantienen orden relativo

data = [("Alice", 25), ("Bob", 30), ("Carol", 25)]

# Ordenar por edad
# ESTABLE: Alice antes de Carol (original order preserved)
# sorted_stable = [("Alice", 25), ("Carol", 25), ("Bob", 30)]

# NO ESTABLE: Carol podría quedar antes de Alice
# sorted_unstable = [("Carol", 25), ("Alice", 25), ("Bob", 30)]
```

4.3 Cuándo Usar Cada Uno

```
┌──────────────────────────────────────────────────────────┐
│  USA QUICKSORT cuando:                                   │
│  • Memoria es limitada (in-place)                        │
│  • No necesitas estabilidad                              │
│  • Datos son aleatorios (no ya ordenados)                │
│  • Quieres mejor rendimiento promedio en práctica        │
│                                                          │
│  USA MERGESORT cuando:                                   │
│  • Necesitas garantía O(n log n) siempre                 │
│  • Necesitas ordenamiento estable                        │
│  • Memoria no es problema                                │
│  • Datos podrían estar casi ordenados                    │
│                                                          │
│  EN ARCHIMEDES:                                          │
│  Usaremos QuickSort para ordenar resultados por score    │
│  porque raramente están pre-ordenados y queremos velocidad │
└──────────────────────────────────────────────────────────┘
```

---

5. Análisis de Complejidad Detallado {#5-analisis}

5.1 QuickSort: Por Qué O(n log n) Promedio

```
┌──────────────────────────────────────────────────────────┐
│  MEJOR CASO: Pivote divide perfectamente por la mitad    │
│                                                          │
│  Nivel 0: 1 problema de tamaño n                         │
│  Nivel 1: 2 problemas de tamaño n/2                      │
│  Nivel 2: 4 problemas de tamaño n/4                      │
```

```
│  ...                                                                │
│  Nivel log n: n problemas de tamaño 1                               │
│                                                                     │
│  Trabajo por nivel: O(n) (partición)                                │
│  Número de niveles: O(log n)                                        │
│  Total: O(n) × O(log n) = O(n log n)                                │
└─────────────────────────────────────────────────────────────────────┘
```

5.2 QuickSort: Por Qué O(n²) Peor Caso

```
┌─────────────────────────────────────────────────────────────────────┐
│  PEOR CASO: Lista ya ordenada + pivot siempre el último             │
│                                                                     │
│  [1, 2, 3, 4, 5]  pivot=5 → [1,2,3,4] [] + [5]                      │
│  [1, 2, 3, 4]     pivot=4 → [1,2,3]   [] + [4]                      │
│  [1, 2, 3]        pivot=3 → [1,2]     [] + [3]                      │
│  [1, 2]           pivot=2 → [1]       [] + [2]                      │
│                                                                     │
│  Cada nivel quita solo 1 elemento → n niveles                       │
│  Trabajo por nivel: O(n), O(n-1), O(n-2), ...                       │
│  Total: n + (n-1) + ... + 1 = n(n+1)/2 = O(n²)                      │
│                                                                     │
│  SOLUCIÓN: Random pivot evita esto en la práctica                   │
└─────────────────────────────────────────────────────────────────────┘
```

5.3 MergeSort: Siempre O(n log n)

```
┌─────────────────────────────────────────────────────────────────────┐
│  SIEMPRE divide exactamente por la mitad                            │
│                                                                     │
│  T(n) = 2×T(n/2) + O(n)                                             │
│         ↑         ↑                                                 │
│    2 subproblemas  merge                                            │
│    de tamaño n/2                                                    │
│                                                                     │
│  Por Master Theorem:                                                │
│  T(n) = O(n log n)                                                  │
│                                                                     │
│  No hay peor caso porque la división es siempre balanceada          │
└─────────────────────────────────────────────────────────────────────┘
```

5.4 Análisis de Espacio

```
# QuickSort: O(log n) espacio para call stack
# - Cada llamada recursiva usa espacio constante
# - Profundidad máxima: log n (caso promedio)
# - Profundidad máxima: n (peor caso)

# MergeSort: O(n) espacio para arrays temporales
# - Cada merge crea nuevo array
# - El array más grande es de tamaño n
# - Plus O(log n) para call stack
```

⚠️ Errores Comunes

Error 1: Off-by-one en partition

```
# ❌ Error común: incluir pivote en recursión
_quicksort_helper(items, low, pivot_index)  # Incluye pivote
```

```
_quicksort_helper(items, pivot_index, high)  # Pivote otra vez!

# ✅ Correcto: excluir pivote (ya está en su lugar)
_quicksort_helper(items, low, pivot_index - 1)
_quicksort_helper(items, pivot_index + 1, high)
```

Error 2: No manejar lista vacía

```
# ❌ Falla con lista vacía
def quicksort_bad(items):
    pivot = items[-1]  # IndexError!

# ✅ Manejar caso base
def quicksort_good(items):
    if len(items) <= 1:
        return items
    pivot = items[-1]
```

Error 3: Modificar lista durante iteración

```
# ❌ Confuso y propenso a errores
for i, item in enumerate(items):
    items[i], items[j] = ...  # Modifica mientras itera

# ✅ Usar índices explícitos
for j in range(low, high):
    if items[j] < pivot:
        i += 1
        items[i], items[j] = items[j], items[i]
```

🔧 Ejercicios Prácticos

Ejercicio 8.1: Implementar QuickSort

Ver EJERCICIOS.md

Ejercicio 8.2: Implementar MergeSort

Ver EJERCICIOS.md

Ejercicio 8.3: Ordenar por Score

Ver EJERCICIOS.md - Aplicar al ranking de Archimedes

📚 Recursos Externos

Recurso	Tipo	Prioridad
Visualgo Sorting	Visualización	🔴 Obligatorio
Grokking Algorithms Ch.4	Libro	🔴 Obligatorio
QuickSort Analysis	Video	🟡 Recomendado

🔗 Referencias del Glosario

- QuickSort

- MergeSort
- Partition
- Estabilidad
- In-Place

🧭 Navegación

← Anterior	Índice	Siguiente →
07_RECURSION	00_INDICE	09_BINARY_SEARCH

Anexo DSA - Trees y BST

Guía MS in AI Pathway

DUQUEOM · 2025

Anexo DSA - Árboles y Binary Search Trees

> ⚠️ **MÓDULO OPCIONAL:** Este módulo NO es requerido para el Pathway. Es útil para entrevistas técnicas.
> 🎯 **Objetivo:** Dominar árboles binarios, BST y sus traversals.

🧠 Analogía: El Árbol Genealógico

```
┌─────────────────────────────────────────────────────────────┐
│                                                             │
│   ÁRBOL = Estructura jerárquica como árbol genealógico      │
│   ──────────────────────────────────────────────            │
│                                                             │
│                        [Abuelo]           ← ROOT (raíz)     │
│                        /      \                             │
│                   [Padre]    [Tío]        ← INTERNAL NODES  │
│                   /    \        \                          │
│               [Hijo1] [Hijo2]  [Primo]    ← LEAVES (hojas)  │
│                                                             │
│   TÉRMINOS:                                                 │
│   • Root: Nodo sin padre (el de arriba)                     │
│   • Parent/Child: Relación directa                          │
│   • Siblings: Nodos con mismo padre                         │
│   • Leaf: Nodo sin hijos                                    │
│   • Height: Distancia máxima desde root a hoja              │
│   • Depth: Distancia desde root a un nodo                   │
│                                                             │
│   BINARY TREE = Cada nodo tiene máximo 2 hijos (left, right)│
│                                                             │
└─────────────────────────────────────────────────────────────┘
```

📋 Contenido

---

1. Binary Tree Básico {#1-binary-tree}

1.1 Estructura del Nodo

```python
from typing import Generic, TypeVar, Optional

T = TypeVar('T')


class TreeNode(Generic[T]):
    """A node in a binary tree.

    Attributes:
        value: Data stored in this node.
        left: Reference to left child (or None).
        right: Reference to right child (or None).
```

```
    """

    def __init__(self, value: T) -> None:
        self.value: T = value
        self.left: Optional[TreeNode[T]] = None
        self.right: Optional[TreeNode[T]] = None

    def __repr__(self) -> str:
        return f"TreeNode({self.value})"

    def is_leaf(self) -> bool:
        """Check if node has no children."""
        return self.left is None and self.right is None


class BinaryTree(Generic[T]):
    """Basic binary tree structure."""

    def __init__(self) -> None:
        self.root: Optional[TreeNode[T]] = None

    def is_empty(self) -> bool:
        return self.root is None
```

1.2 Construir un Árbol Manualmente

```
#        10
#       /  \
#      5    15
#     / \   / \
#    3   7 12  20

root = TreeNode(10)
root.left = TreeNode(5)
root.right = TreeNode(15)
root.left.left = TreeNode(3)
root.left.right = TreeNode(7)
root.right.left = TreeNode(12)
root.right.right = TreeNode(20)
```

---

2. Traversals (Recorridos) {#2-traversals}

2.1 Los Tres Traversals DFS

```
┌─────────────────────────────────────────────────────────────┐
│  TRES FORMAS DE RECORRER UN ÁRBOL (DFS)                     │
│                                                             │
│        1                                                    │
│       / \                                                   │
│      2   3                                                  │
│     / \                                                     │
│    4   5                                                    │
│                                                             │
│  INORDER (Left, Root, Right):   4, 2, 5, 1, 3               │
│  → En BST: ¡sale ORDENADO!                                  │
│                                                             │
│  PREORDER (Root, Left, Right):  1, 2, 4, 5, 3               │
│  → Útil para copiar/serializar árbol                        │
│                                                             │
│  POSTORDER (Left, Right, Root): 4, 5, 2, 3, 1               │
```

```
│  → Útil para eliminar árbol (hijos antes que padre)              │
└──────────────────────────────────────────────────────────────────┘
```

2.2 Implementación Recursiva

```
def inorder_recursive(node: Optional[TreeNode[T]]) -> list[T]:
    """Inorder traversal: Left, Root, Right.

    For BST, returns elements in sorted order.

    Time: O(n) - visit each node once
    Space: O(h) - recursion stack, h = height
    """
    if node is None:
        return []

    result = []
    result.extend(inorder_recursive(node.left))
    result.append(node.value)
    result.extend(inorder_recursive(node.right))
    return result


def preorder_recursive(node: Optional[TreeNode[T]]) -> list[T]:
    """Preorder traversal: Root, Left, Right."""
    if node is None:
        return []

    result = [node.value]
    result.extend(preorder_recursive(node.left))
    result.extend(preorder_recursive(node.right))
    return result


def postorder_recursive(node: Optional[TreeNode[T]]) -> list[T]:
    """Postorder traversal: Left, Right, Root."""
    if node is None:
        return []

    result = []
    result.extend(postorder_recursive(node.left))
    result.extend(postorder_recursive(node.right))
    result.append(node.value)
    return result
```

2.3 Implementación Iterativa (con Stack)

```
def inorder_iterative(root: Optional[TreeNode[T]]) -> list[T]:
    """Inorder using explicit stack instead of recursion.

    Important: Shows how recursion uses the call stack.
    """
    result = []
    stack: list[TreeNode[T]] = []
    current = root

    while current is not None or stack:
        # Go as far left as possible
        while current is not None:
            stack.append(current)
            current = current.left
```

```
        # Process current node
        current = stack.pop()
        result.append(current.value)

        # Move to right subtree
        current = current.right

    return result


def preorder_iterative(root: Optional[TreeNode[T]]) -> list[T]:
    """Preorder using stack."""
    if root is None:
        return []

    result = []
    stack = [root]

    while stack:
        node = stack.pop()
        result.append(node.value)

        # Push right first so left is processed first (LIFO)
        if node.right:
            stack.append(node.right)
        if node.left:
            stack.append(node.left)

    return result
```

2.4 Level Order (BFS)

```
from collections import deque


def level_order(root: Optional[TreeNode[T]]) -> list[list[T]]:
    """Level order traversal using queue (BFS).

    Returns nodes level by level.

    Example:
        [1]
        [2, 3]
        [4, 5]
    """
    if root is None:
        return []

    result = []
    queue = deque([root])

    while queue:
        level_size = len(queue)
        current_level = []

        for _ in range(level_size):
            node = queue.popleft()
            current_level.append(node.value)

            if node.left:
                queue.append(node.left)
            if node.right:
```

```
                queue.append(node.right)

        result.append(current_level)

    return result
```

---

3. Binary Search Tree (BST) {#3-bst}

3.1 Propiedad del BST

```
┌─────────────────────────────────────────────────────────┐
│  BST PROPERTY:                                          │
│  Para cada nodo:                                        │
│  • Todos los valores en subárbol izquierdo < valor del nodo    │
│  • Todos los valores en subárbol derecho > valor del nodo      │
│                                                         │
│  VÁLIDO BST:           INVÁLIDO BST:                    │
│        10                    10                         │
│       /  \                  /  \                       │
│      5    15               5    15                      │
│     / \                   / \                           │
│    3   7                 3   12  ← 12 > 10 pero está en left!  │
│                                                         │
│  BENEFICIO: Búsqueda O(log n) en promedio               │
└─────────────────────────────────────────────────────────┘
```

3.2 Implementación de BST

```
class BST(Generic[T]):
    """Binary Search Tree implementation.

    Maintains BST property: left < root < right.

    Average case complexities (balanced):
        - search: O(log n)
        - insert: O(log n)
        - delete: O(log n)

    Worst case (unbalanced/skewed):
        - All operations: O(n)
    """

    def __init__(self) -> None:
        self.root: Optional[TreeNode[T]] = None
        self._size: int = 0

    def __len__(self) -> int:
        return self._size

    def is_empty(self) -> bool:
        return self.root is None

    def insert(self, value: T) -> None:
        """Insert value maintaining BST property. O(h)"""
        self.root = self._insert_recursive(self.root, value)
        self._size += 1

    def _insert_recursive(
        self,
        node: Optional[TreeNode[T]],
```

```
        value: T
    ) -> TreeNode[T]:
        """Recursive helper for insert."""
        if node is None:
            return TreeNode(value)

        if value < node.value:
            node.left = self._insert_recursive(node.left, value)
        elif value > node.value:
            node.right = self._insert_recursive(node.right, value)
        # If equal, we don't insert (no duplicates)

        return node

    def search(self, value: T) -> bool:
        """Search for value in BST. O(h)"""
        return self._search_recursive(self.root, value)

    def _search_recursive(
        self,
        node: Optional[TreeNode[T]],
        value: T
    ) -> bool:
        """Recursive helper for search."""
        if node is None:
            return False

        if value == node.value:
            return True
        elif value < node.value:
            return self._search_recursive(node.left, value)
        else:
            return self._search_recursive(node.right, value)

    def search_iterative(self, value: T) -> bool:
        """Iterative search - often preferred."""
        current = self.root

        while current is not None:
            if value == current.value:
                return True
            elif value < current.value:
                current = current.left
            else:
                current = current.right

        return False

    def find_min(self) -> Optional[T]:
        """Find minimum value. O(h)"""
        if self.root is None:
            return None

        current = self.root
        while current.left is not None:
            current = current.left
        return current.value

    def find_max(self) -> Optional[T]:
        """Find maximum value. O(h)"""
        if self.root is None:
            return None
```

```
        current = self.root
        while current.right is not None:
            current = current.right
        return current.value

    def inorder(self) -> list[T]:
        """Return sorted list of all values."""
        return inorder_recursive(self.root)
```

---

4. Operaciones en BST {#4-operaciones}

4.1 Delete (La Más Compleja)

```
def delete(self, value: T) -> None:
    """Delete value from BST. O(h)

    Three cases:
    1. Leaf node: just remove
    2. One child: replace with child
    3. Two children: replace with inorder successor
    """
    self.root = self._delete_recursive(self.root, value)

def _delete_recursive(
    self,
    node: Optional[TreeNode[T]],
    value: T
) -> Optional[TreeNode[T]]:
    """Recursive helper for delete."""
    if node is None:
        return None

    if value < node.value:
        node.left = self._delete_recursive(node.left, value)
    elif value > node.value:
        node.right = self._delete_recursive(node.right, value)
    else:
        # Found node to delete

        # Case 1: Leaf node
        if node.left is None and node.right is None:
            self._size -= 1
            return None

        # Case 2: One child
        if node.left is None:
            self._size -= 1
            return node.right
        if node.right is None:
            self._size -= 1
            return node.left

        # Case 3: Two children
        # Find inorder successor (smallest in right subtree)
        successor = self._find_min_node(node.right)
        node.value = successor.value
        node.right = self._delete_recursive(node.right, successor.value)

    return node

def _find_min_node(self, node: TreeNode[T]) -> TreeNode[T]:
```

```
    """Find node with minimum value in subtree."""
    current = node
    while current.left is not None:
        current = current.left
    return current
```

4.2 Validar si es BST

```
def is_valid_bst(root: Optional[TreeNode[int]]) -> bool:
    """Check if tree is valid BST.

    Uses inorder traversal: should be sorted.
    """
    def inorder(node: Optional[TreeNode[int]]) -> list[int]:
        if node is None:
            return []
        return inorder(node.left) + [node.value] + inorder(node.right)

    values = inorder(root)

    # Check if sorted
    for i in range(len(values) - 1):
        if values[i] >= values[i + 1]:
            return False
    return True


def is_valid_bst_efficient(
    root: Optional[TreeNode[int]],
    min_val: float = float('-inf'),
    max_val: float = float('inf')
) -> bool:
    """Check BST validity with range checking. O(n) time, O(h) space."""
    if root is None:
        return True

    if root.value <= min_val or root.value >= max_val:
        return False

    return (
        is_valid_bst_efficient(root.left, min_val, root.value) and
        is_valid_bst_efficient(root.right, root.value, max_val)
    )
```

---

5. Análisis de Complejidad {#5-analisis}

5.1 Complejidades

Operación	Balanced BST	Skewed BST
Search	O(log n)	O(n)
Insert	O(log n)	O(n)
Delete	O(log n)	O(n)
Traversal	O(n)	O(n)
Min/Max	O(log n)	O(n)

5.2 Por Qué se Desbalancea

```
Insertar 1, 2, 3, 4, 5 en orden:

    1
     \
      2
       \
        3
         \
          4
           \
            5

→ Se convierte en linked list → O(n) para todo
→ Solución: Árboles balanceados (AVL, Red-Black)
```

⚠️ Errores Comunes

Error 1: Confundir traversals

```
# Memorizar: "Inorder = In order" (para BST)
# Inorder de BST siempre da elementos ORDENADOS
```

Error 2: No manejar caso vacío

```
# ❌
def find_min(root):
    while root.left:  # AttributeError si root es None
        root = root.left

# ✅
def find_min(root):
    if root is None:
        return None
    while root.left:
        root = root.left
    return root.value
```

🔧 Ejercicios Prácticos

Ejercicio 14.1: Implementar BST con insert y search

Ejercicio 14.2: Implementar los 3 traversals

Ejercicio 14.3: Validar si árbol es BST

Ejercicio 14.4: Encontrar altura del árbol

📚 Recursos Externos

Recurso	Tipo	Prioridad
Visualgo BST	Visual	🔴 Obligatorio
Abdul Bari Trees	Video	🔴 Obligatorio

🧭 Navegación

← Anterior	Índice	Siguiente →
13_LINKED_LISTS	00_INDICE	15_GRAPHS

Anexo DSA - Graphs, BFS, DFS

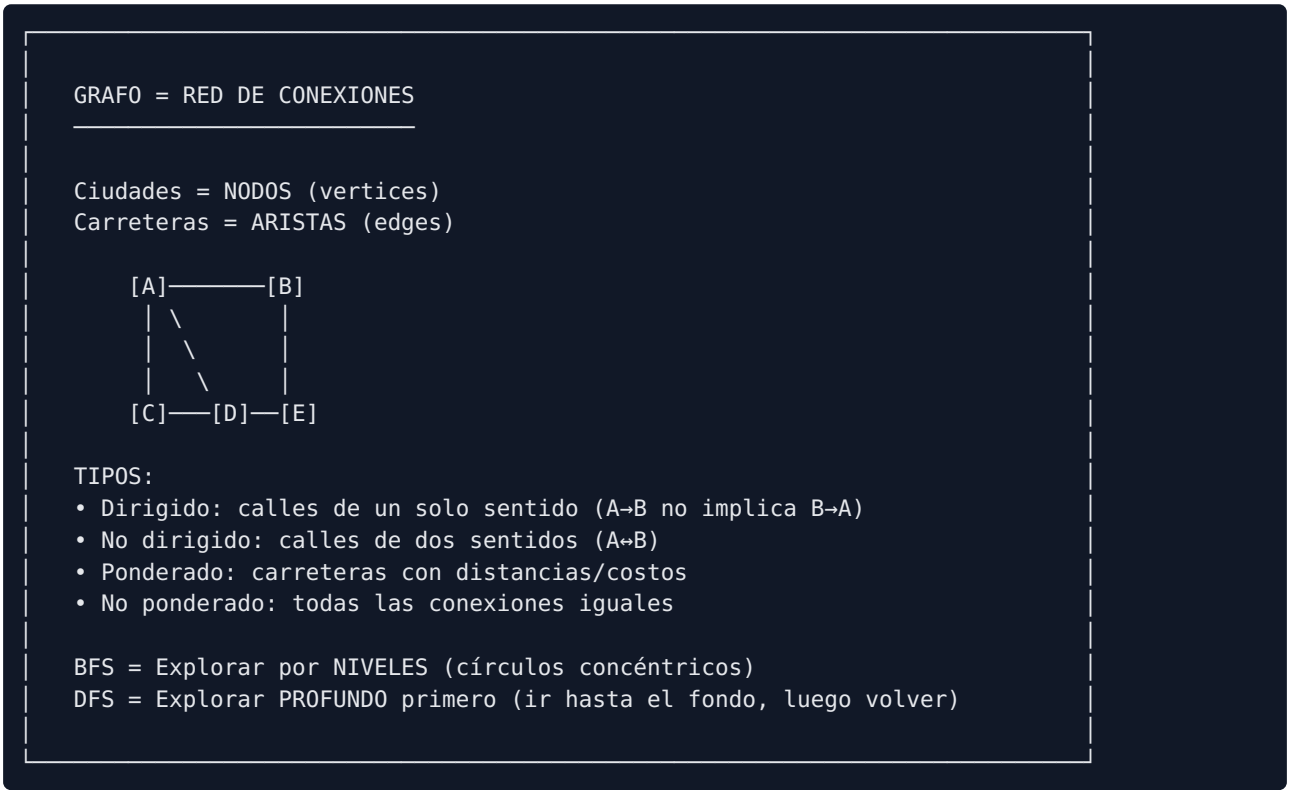
Guía MS in AI Pathway

DUQUEOM · 2025

Anexo DSA - Grafos, BFS y DFS

> ⚠️ **MÓDULO OPCIONAL:** Este módulo NO es requerido para el Pathway. Es útil para entrevistas técnicas.
> 🎯 **Objetivo:** Dominar grafos y sus algoritmos de recorrido.

🧠 Analogía: Mapa de Ciudades y Carreteras



📋 Contenido

---

1. Representación de Grafos {#1-representacion}

1.1 Adjacency List (Lista de Adyacencia)

```
from collections import defaultdict
from typing import TypeVar, Generic

T = TypeVar('T')


class Graph(Generic[T]):
    """Unweighted graph using adjacency list.

    Most common representation. Good for sparse graphs.
```

```
    Space: O(V + E)
    """

    def __init__(self, directed: bool = False) -> None:
        self.adjacency: dict[T, list[T]] = defaultdict(list)
        self.directed = directed

    def add_vertex(self, vertex: T) -> None:
        """Add vertex without edges."""
        if vertex not in self.adjacency:
            self.adjacency[vertex] = []

    def add_edge(self, source: T, destination: T) -> None:
        """Add edge between vertices.

        For undirected graph, adds both directions.
        """
        self.adjacency[source].append(destination)

        if not self.directed:
            self.adjacency[destination].append(source)

    def get_neighbors(self, vertex: T) -> list[T]:
        """Get all neighbors of a vertex."""
        return self.adjacency.get(vertex, [])

    def get_vertices(self) -> list[T]:
        """Get all vertices."""
        return list(self.adjacency.keys())

    def __repr__(self) -> str:
        return f"Graph({dict(self.adjacency)})"


# Ejemplo de uso
graph = Graph[str](directed=False)
graph.add_edge("A", "B")
graph.add_edge("A", "C")
graph.add_edge("B", "D")
graph.add_edge("C", "D")
print(graph.get_neighbors("A"))  # ['B', 'C']
```

1.2 Adjacency Matrix (Matriz de Adyacencia)

```
class GraphMatrix:
    """Graph using adjacency matrix.

    Good for dense graphs or when need O(1) edge lookup.
    Space: O(V²)
    """

    def __init__(self, num_vertices: int) -> None:
        self.num_vertices = num_vertices
        # matrix[i][j] = 1 if edge from i to j
        self.matrix: list[list[int]] = [
            [0] * num_vertices for _ in range(num_vertices)
        ]

    def add_edge(self, source: int, dest: int) -> None:
        """Add edge (undirected)."""
        self.matrix[source][dest] = 1
        self.matrix[dest][source] = 1
```

```
    def has_edge(self, source: int, dest: int) -> bool:
        """Check if edge exists. O(1)"""
        return self.matrix[source][dest] == 1

    def get_neighbors(self, vertex: int) -> list[int]:
        """Get neighbors. O(V)"""
        return [i for i, val in enumerate(self.matrix[vertex]) if val == 1]
```

1.3 Cuándo Usar Cada Representación

Operación	Adj List	Adj Matrix
Space	O(V + E)	O(V²)
Check edge	O(degree)	O(1)
Get neighbors	O(1)	O(V)
Add edge	O(1)	O(1)
Mejor para	Sparse graphs	Dense graphs

---

2. BFS (Breadth-First Search) {#2-bfs}

2.1 Concepto

```
┌─────────────────────────────────────────────────────────┐
│  BFS = Buscar por NIVELES (como ondas en el agua)       │
│                                                         │
│  Desde A:                                               │
│       [A]──[B]──[D]                                     │
│        │    │                                          │
│       [C]──[E]                                          │
│                                                         │
│  Nivel 0: A                                             │
│  Nivel 1: B, C (vecinos de A)                           │
│  Nivel 2: D, E (vecinos de B y C no visitados)          │
│                                                         │
│  ORDEN DE VISITA: A → B → C → D → E                     │
│                                                         │
│  USA QUEUE (FIFO) para procesar en orden de llegada     │
└─────────────────────────────────────────────────────────┘
```

2.2 Implementación

```
from collections import deque


def bfs(graph: Graph[T], start: T) -> list[T]:
    """Breadth-First Search traversal.

    Visits nodes level by level using a queue.

    Args:
        graph: The graph to traverse.
        start: Starting vertex.

    Returns:
        List of vertices in BFS order.

    Time: O(V + E)
```

```
    Space: O(V) for visited set and queue
    """
    visited: set[T] = set()
    result: list[T] = []
    queue: deque[T] = deque([start])

    visited.add(start)

    while queue:
        vertex = queue.popleft()  # FIFO
        result.append(vertex)

        for neighbor in graph.get_neighbors(vertex):
            if neighbor not in visited:
                visited.add(neighbor)
                queue.append(neighbor)

    return result


def bfs_with_levels(graph: Graph[T], start: T) -> list[list[T]]:
    """BFS that returns nodes grouped by level."""
    visited: set[T] = set()
    levels: list[list[T]] = []
    queue: deque[T] = deque([start])

    visited.add(start)

    while queue:
        level_size = len(queue)
        current_level: list[T] = []

        for _ in range(level_size):
            vertex = queue.popleft()
            current_level.append(vertex)

            for neighbor in graph.get_neighbors(vertex):
                if neighbor not in visited:
                    visited.add(neighbor)
                    queue.append(neighbor)

        levels.append(current_level)

    return levels
```

2.3 Shortest Path (Unweighted)

```
def shortest_path_bfs(
    graph: Graph[T],
    start: T,
    end: T
) -> list[T] | None:
    """Find shortest path in unweighted graph.

    BFS guarantees shortest path in unweighted graphs
    because it explores level by level.

    Returns:
        List of vertices from start to end, or None if no path.
    """
    if start == end:
        return [start]
```

```
    visited: set[T] = set()
    queue: deque[tuple[T, list[T]]] = deque([(start, [start])])
    visited.add(start)

    while queue:
        vertex, path = queue.popleft()

        for neighbor in graph.get_neighbors(vertex):
            if neighbor == end:
                return path + [neighbor]

            if neighbor not in visited:
                visited.add(neighbor)
                queue.append((neighbor, path + [neighbor]))

    return None  # No path found
```

---

3. DFS (Depth-First Search) {#3-dfs}

3.1 Concepto

```
┌─────────────────────────────────────────────────────────┐
│  DFS = Ir lo más PROFUNDO posible, luego retroceder     │
│                                                         │
│  Desde A:                                               │
│       [A]──[B]──[D]                                     │
│        │    │                                          │
│       [C]──[E]                                          │
│                                                         │
│  Camino: A → B → D (fondo!) → back → E → back → C       │
│                                                         │
│  ORDEN DE VISITA: A → B → D → E → C                     │
│  (puede variar según orden de vecinos)                  │
│                                                         │
│  USA STACK (LIFO) o RECURSIÓN                           │
└─────────────────────────────────────────────────────────┘
```

3.2 Implementación Recursiva

```
def dfs_recursive(graph: Graph[T], start: T) -> list[T]:
    """Depth-First Search using recursion.

    Time: O(V + E)
    Space: O(V) for visited + O(V) for call stack
    """
    visited: set[T] = set()
    result: list[T] = []

    def _dfs(vertex: T) -> None:
        visited.add(vertex)
        result.append(vertex)

        for neighbor in graph.get_neighbors(vertex):
            if neighbor not in visited:
                _dfs(neighbor)

    _dfs(start)
    return result
```

3.3 Implementación Iterativa (con Stack)

```
def dfs_iterative(graph: Graph[T], start: T) -> list[T]:
    """Depth-First Search using explicit stack.

    Avoids recursion limit issues for large graphs.

    Note: Order may differ slightly from recursive
    due to stack vs recursion mechanics.
    """
    visited: set[T] = set()
    result: list[T] = []
    stack: list[T] = [start]

    while stack:
        vertex = stack.pop()  # LIFO

        if vertex not in visited:
            visited.add(vertex)
            result.append(vertex)

            # Add neighbors to stack (reverse for same order as recursive)
            for neighbor in reversed(graph.get_neighbors(vertex)):
                if neighbor not in visited:
                    stack.append(neighbor)

    return result
```

3.4 Detectar Ciclos con DFS

```
def has_cycle_undirected(graph: Graph[T]) -> bool:
    """Detect cycle in undirected graph using DFS."""
    visited: set[T] = set()

    def _dfs(vertex: T, parent: T | None) -> bool:
        visited.add(vertex)

        for neighbor in graph.get_neighbors(vertex):
            if neighbor not in visited:
                if _dfs(neighbor, vertex):
                    return True
            elif neighbor != parent:
                # Found visited node that's not parent = cycle
                return True

        return False

    # Check all components (graph may be disconnected)
    for vertex in graph.get_vertices():
        if vertex not in visited:
            if _dfs(vertex, None):
                return True

    return False
```

---

4. Aplicaciones Comunes {#4-aplicaciones}

4.1 Encontrar Todos los Caminos

```
def find_all_paths(
    graph: Graph[T],
    start: T,
    end: T
) -> list[list[T]]:
    """Find all paths from start to end using DFS."""
    all_paths: list[list[T]] = []

    def _dfs(vertex: T, path: list[T]) -> None:
        if vertex == end:
            all_paths.append(path.copy())
            return

        for neighbor in graph.get_neighbors(vertex):
            if neighbor not in path:  # Avoid cycles
                path.append(neighbor)
                _dfs(neighbor, path)
                path.pop()  # Backtrack

    _dfs(start, [start])
    return all_paths
```

4.2 Componentes Conexos

```
def count_connected_components(graph: Graph[T]) -> int:
    """Count number of connected components."""
    visited: set[T] = set()
    count = 0

    for vertex in graph.get_vertices():
        if vertex not in visited:
            # BFS/DFS from this vertex marks all reachable
            bfs_mark_visited(graph, vertex, visited)
            count += 1

    return count


def bfs_mark_visited(
    graph: Graph[T],
    start: T,
    visited: set[T]
) -> None:
    """Mark all reachable vertices as visited."""
    queue: deque[T] = deque([start])
    visited.add(start)

    while queue:
        vertex = queue.popleft()
        for neighbor in graph.get_neighbors(vertex):
            if neighbor not in visited:
                visited.add(neighbor)
                queue.append(neighbor)
```

---

5. Comparación BFS vs DFS {#5-comparacion}

5.1 Cuándo Usar Cada Uno

Aspecto	BFS	DFS
Estructura	Queue	Stack/Recursión
Explora	Por niveles	Por profundidad
Shortest Path	✅ Garantizado*	❌ No garantizado
Memoria	O(ancho del grafo)	O(profundidad)
Grafos anchos	❌ Mucha memoria	✅ Mejor
Grafos profundos	✅ Mejor	❌ Stack overflow

* Solo para grafos no ponderados

5.2 Resumen de Uso

```
USA BFS cuando:
• Necesitas shortest path (no ponderado)
• Explorar por niveles
• Grafos muy profundos (evita stack overflow)

USA DFS cuando:
• Necesitas explorar todos los caminos
• Detectar ciclos
• Topological sort
• Grafos muy anchos (menos memoria)
```

⚠️ Errores Comunes

Error 1: Olvidar marcar como visitado ANTES de agregar a queue/stack

```
# ❌ Puede agregar mismo nodo múltiples veces
if neighbor not in visited:
    queue.append(neighbor)
    # visited.add(neighbor)  # ¡Falta!

# ✅ Marcar inmediatamente
if neighbor not in visited:
    visited.add(neighbor)  # Antes de agregar
    queue.append(neighbor)
```

Error 2: No manejar grafos desconectados

```
# ❌ Solo visita un componente
def bfs_bad(graph, start):
    # Solo desde start...

# ✅ Iterar sobre todos los vértices
def bfs_all(graph):
    visited = set()
    for vertex in graph.get_vertices():
        if vertex not in visited:
            bfs(graph, vertex)  # Visita este componente
```

🔧 Ejercicios Prácticos

Ejercicio 15.1: Implementar BFS

Ejercicio 15.2: Implementar DFS recursivo e iterativo

Ejercicio 15.3: Shortest path con BFS

Ejercicio 15.4: Detectar ciclo en grafo

📚 Recursos Externos

Recurso	Tipo	Prioridad
Visualgo Graph	Visual	🔴 Obligatorio
Abdul Bari BFS/DFS	Video	🔴 Obligatorio

🧭 Navegación

← Anterior	Índice	Siguiente →
14_TREES	00_INDICE	16_DYNAMIC_PROGRAMMING

Anexo DSA - Dynamic Programming

Guía MS in AI Pathway

DUQUEOM · 2025

Anexo DSA - Dynamic Programming

> ⚠️ **MÓDULO OPCIONAL:** Este módulo NO es requerido para el Pathway. Es útil para entrevistas técnicas.
> 🎯 **Objetivo:** Dominar la técnica de DP para resolver problemas de optimización.

🧠 Analogía: No Calcular lo Mismo Dos Veces

```
┌─────────────────────────────────────────────────────────────────┐
│                                                                 │
│   DYNAMIC PROGRAMMING = Recordar para no recalcular             │
│   ──────────────────────────────────────────────                │
│                                                                 │
│   FIBONACCI NAIVE:                                              │
│                fib(5)                                           │
│              /      \                                           │
│          fib(4)      fib(3)          ← fib(3) se calcula 2 veces!
│          /    \       /   \                                     │
│       fib(3) fib(2) fib(2) fib(1)     ← fib(2) se calcula 3 veces!
│         ...                                                     │
│                                                                 │
│   DYNAMIC PROGRAMMING:                                          │
│   ┌──────────────────────────────┐                              │
│   │ Ya calculé fib(3)? → Buscar  │                              │
│   │ No calculado? → Calcular y   │                              │
│   │                 GUARDAR      │                              │
│   └──────────────────────────────┘                              │
│                                                                 │
│   REQUISITOS para usar DP:                                      │
│   1. OPTIMAL SUBSTRUCTURE: Solución óptima usa soluciones óptimas de
│      subproblemas                                               │
│   2. OVERLAPPING SUBPROBLEMS: Los mismos subproblemas se repiten
│                                                                 │
└─────────────────────────────────────────────────────────────────┘
```

📋 Contenido

---

1. Conceptos Fundamentales {#1-conceptos}

1.1 ¿Qué es Dynamic Programming?

```
DP = Técnica de optimización que:
1. Divide problema en subproblemas
2. Resuelve cada subproblema UNA SOLA VEZ
3. Guarda resultados para reusar

NO es:
- Un algoritmo específico
```

```
- Solo memorización
- Aplicable a cualquier problema
```

1.2 Dos Enfoques

Top-Down (Memoization)	Bottom-Up (Tabulation)
Recursivo + cache	Iterativo + tabla
Empieza del problema grande	Empieza de casos base
Solo calcula lo necesario	Calcula todo
Más intuitivo	Más eficiente (no call stack)

---

2. Top-Down (Memoization) {#2-top-down}

2.1 Fibonacci con Memoization

```
def fibonacci_memo(n: int, memo: dict[int, int] | None = None) -> int:
    """Fibonacci with memoization (top-down DP).

    Time: O(n) - each value computed once
    Space: O(n) - for memo dict + call stack

    Example:
        >>> fibonacci_memo(10)
        55
    """
    if memo is None:
        memo = {}

    # Check cache first
    if n in memo:
        return memo[n]

    # Base cases
    if n <= 1:
        return n

    # Compute and cache
    memo[n] = fibonacci_memo(n - 1, memo) + fibonacci_memo(n - 2, memo)
    return memo[n]


# Con decorator
from functools import lru_cache

@lru_cache(maxsize=None)
def fibonacci_lru(n: int) -> int:
    """Fibonacci with automatic memoization."""
    if n <= 1:
        return n
    return fibonacci_lru(n - 1) + fibonacci_lru(n - 2)
```

2.2 Template Top-Down

```
def solve_top_down(problem):
    memo = {}

    def dp(state):
        # 1. Check cache
```

```
        if state in memo:
            return memo[state]

        # 2. Base case
        if is_base_case(state):
            return base_value

        # 3. Recurrence relation
        result = combine(dp(smaller_states))

        # 4. Cache and return
        memo[state] = result
        return result

    return dp(initial_state)
```

---

3. Bottom-Up (Tabulation) {#3-bottom-up}

3.1 Fibonacci Bottom-Up

```
def fibonacci_bottom_up(n: int) -> int:
    """Fibonacci with tabulation (bottom-up DP).

    Builds solution from base cases up.

    Time: O(n)
    Space: O(n) for the table

    Example:
        >>> fibonacci_bottom_up(10)
        55
    """
    if n <= 1:
        return n

    # Table to store computed values
    dp = [0] * (n + 1)

    # Base cases
    dp[0] = 0
    dp[1] = 1

    # Fill table from base cases up
    for i in range(2, n + 1):
        dp[i] = dp[i - 1] + dp[i - 2]

    return dp[n]


def fibonacci_optimized(n: int) -> int:
    """Fibonacci with O(1) space.

    Only need previous two values.
    """
    if n <= 1:
        return n

    prev2 = 0  # fib(i-2)
    prev1 = 1  # fib(i-1)

    for _ in range(2, n + 1):
```

```
        current = prev1 + prev2
        prev2 = prev1
        prev1 = current

    return prev1
```

3.2 Template Bottom-Up

```
def solve_bottom_up(problem):
    # 1. Define table size and initialize
    dp = initialize_table(problem_size)

    # 2. Set base cases
    dp[base_indices] = base_values

    # 3. Fill table iteratively
    for state in all_states_in_order:
        dp[state] = combine(dp[smaller_states])

    # 4. Return final answer
    return dp[final_state]
```

---

4. Problemas Clásicos {#4-clasicos}

4.1 Climbing Stairs

```
def climb_stairs(n: int) -> int:
    """Number of ways to climb n stairs (1 or 2 steps at a time).

    Recurrence: ways(n) = ways(n-1) + ways(n-2)
    (Same as Fibonacci!)

    Example:
        >>> climb_stairs(4)
        5  # [1,1,1,1], [1,1,2], [1,2,1], [2,1,1], [2,2]
    """
    if n <= 2:
        return n

    prev2 = 1  # ways to reach step 1
    prev1 = 2  # ways to reach step 2

    for _ in range(3, n + 1):
        current = prev1 + prev2
        prev2 = prev1
        prev1 = current

    return prev1
```

4.2 Coin Change (Minimum Coins)

```
def coin_change(coins: list[int], amount: int) -> int:
    """Find minimum coins needed to make amount.

    Classic DP problem.

    Recurrence:
        dp[a] = min(dp[a - coin] + 1) for all coins where coin <= a

    Example:
```

```
        >>> coin_change([1, 2, 5], 11)
        3  # 5 + 5 + 1

    Time: O(amount * len(coins))
    Space: O(amount)
    """
    # dp[i] = minimum coins to make amount i
    dp = [float('inf')] * (amount + 1)
    dp[0] = 0  # 0 coins to make amount 0

    for a in range(1, amount + 1):
        for coin in coins:
            if coin <= a and dp[a - coin] != float('inf'):
                dp[a] = min(dp[a], dp[a - coin] + 1)

    return dp[amount] if dp[amount] != float('inf') else -1
```

4.3 Longest Common Subsequence (LCS)

```
def lcs(text1: str, text2: str) -> int:
    """Find length of longest common subsequence.

    Subsequence: characters in same order but not necessarily contiguous.

    Example:
        >>> lcs("abcde", "ace")
        3  # "ace"

    Recurrence:
        If text1[i] == text2[j]: dp[i][j] = dp[i-1][j-1] + 1
        Else: dp[i][j] = max(dp[i-1][j], dp[i][j-1])

    Time: O(m * n)
    Space: O(m * n)
    """
    m, n = len(text1), len(text2)

    # dp[i][j] = LCS of text1[:i] and text2[:j]
    dp = [[0] * (n + 1) for _ in range(m + 1)]

    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if text1[i - 1] == text2[j - 1]:
                dp[i][j] = dp[i - 1][j - 1] + 1
            else:
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])

    return dp[m][n]
```

4.4 0/1 Knapsack

```
def knapsack(weights: list[int], values: list[int], capacity: int) -> int:
    """0/1 Knapsack: maximize value within weight capacity.

    Each item can be taken at most once.

    Example:
        >>> knapsack([1, 2, 3], [6, 10, 12], 5)
        22  # items with weight 2 and 3

    Recurrence:
        dp[i][w] = max(
```

```
            dp[i-1][w],                         # don't take item i
            dp[i-1][w-weight[i]] + value[i]     # take item i
        )

    Time: O(n * capacity)
    Space: O(n * capacity) or O(capacity) optimized
    """
    n = len(weights)

    # dp[i][w] = max value using first i items with capacity w
    dp = [[0] * (capacity + 1) for _ in range(n + 1)]

    for i in range(1, n + 1):
        for w in range(capacity + 1):
            # Don't take item i
            dp[i][w] = dp[i - 1][w]

            # Take item i (if it fits)
            if weights[i - 1] <= w:
                dp[i][w] = max(
                    dp[i][w],
                    dp[i - 1][w - weights[i - 1]] + values[i - 1]
                )

    return dp[n][capacity]
```

4.5 Maximum Subarray (Kadane's Algorithm)

```
def max_subarray(nums: list[int]) -> int:
    """Find contiguous subarray with maximum sum.

    Kadane's Algorithm - clever DP.

    Recurrence:
        max_ending_here = max(num, max_ending_here + num)

    Example:
        >>> max_subarray([-2, 1, -3, 4, -1, 2, 1, -5, 4])
        6  # [4, -1, 2, 1]

    Time: O(n)
    Space: O(1)
    """
    max_sum = nums[0]
    current_sum = nums[0]

    for num in nums[1:]:
        # Either extend current subarray or start new
        current_sum = max(num, current_sum + num)
        max_sum = max(max_sum, current_sum)

    return max_sum
```

---

5. Framework para Resolver DP {#5-framework}

5.1 Pasos para Resolver

```
1. IDENTIFICAR: ¿Es un problema de DP?
   - ¿Tiene optimal substructure?
   - ¿Hay overlapping subproblems?
```

```
   - ¿Pide optimizar algo o contar combinaciones?

2. DEFINIR ESTADO:
   - ¿Qué representa dp[i] o dp[i][j]?
   - ¿Qué información necesito para resolver el problema?

3. ENCONTRAR RECURRENCIA:
   - ¿Cómo se relaciona dp[i] con estados anteriores?
   - Escribir la fórmula matemática

4. IDENTIFICAR CASOS BASE:
   - ¿Cuáles son los subproblemas triviales?
   - ¿Qué valores conozco directamente?

5. DETERMINAR ORDEN DE CÁLCULO:
   - ¿En qué orden llenar la tabla?
   - Asegurar que dependencias ya estén calculadas

6. IMPLEMENTAR:
   - Top-down (más intuitivo) o
   - Bottom-up (más eficiente)
```

5.2 Señales de que es DP

```
KEYWORDS que indican DP:
- "minimum/maximum"
- "count the number of ways"
- "is it possible"
- "longest/shortest"
- "optimal"

PATRONES comunes:
- Secuencias/arrays: dp[i] = f(dp[i-1], dp[i-2], ...)
- Dos secuencias: dp[i][j] = f(dp[i-1][j], dp[i][j-1], ...)
- Intervalos: dp[i][j] = f(dp[i+1][j], dp[i][j-1])
- Capacidad: dp[i][w] = f(dp[i-1][w], dp[i-1][w-item])
```

⚠️ Errores Comunes

Error 1: Recurrencia incorrecta

```
# ❌ Mal: no considera todos los casos
dp[i] = dp[i-1] + something

# ✅ Asegurar que considera TODAS las opciones
dp[i] = max/min over ALL valid transitions
```

Error 2: Orden de cálculo incorrecto

```
# ❌ Usar valores no calculados aún
for i in range(n):
    dp[i] = dp[i+1] + ...  # dp[i+1] no existe!

# ✅ Calcular dependencias primero
for i in range(n-1, -1, -1):  # Reverse
    dp[i] = dp[i+1] + ...
```

🔧 Ejercicios Prácticos

Ejercicio 16.1: Fibonacci con memo y tabulation

Ejercicio 16.2: Coin Change

Ejercicio 16.3: Longest Common Subsequence

Ejercicio 16.4: 0/1 Knapsack

📚 Recursos Externos

Recurso	Tipo	Prioridad
MIT DP Lecture	Video	🔴 Obligatorio
Dynamic Programming Patterns	Guía	🔴 Obligatorio

🧭 Navegación

← Anterior	Índice	Siguiente →
15_GRAPHS	00_INDICE	17_GREEDY