Implementación de Algoritmo PCA en Lenguaje C

Análisis de Componentes Principales desde Cero

Inteligencia Artificial - ELP 8012
Evaluación 3

Jorge Sanchez
Flavio Arregoces
Cristian Gonzales

Programa de Ingeniería de Sistemas
Universidad del Norte

Profesor: Eduardo Zurek, Ph.D.

Octubre 20, 2025

Índice

1. Resumen

Este documento presenta la implementación completa de un algoritmo de Análisis de Componentes Principales (PCA) desarrollado desde cero en lenguaje C. El proyecto incluye un pipeline automatizado que genera datos sintéticos, ejecuta el algoritmo de reducción dimensional y valida los resultados contra la implementación de referencia de scikit-learn.

Palabras clave: PCA, Reducción dimensional, Algoritmos en C, Descomposición eigen, Validación numérica.

2. Introducción

2.1. Contexto del Problema

El Análisis de Componentes Principales (PCA) es una técnica fundamental en aprendizaje automático y análisis de datos que permite reducir la dimensionalidad de conjuntos de datos complejos mientras se preserva la mayor cantidad posible de información. Esta técnica es especialmente útil en:

- Visualización de datos de alta dimensión
- Compresión de información
- Eliminación de ruido
- Preprocesamiento para otros algoritmos

2.2. Objetivos del Proyecto

1. Implementar el algoritmo PCA completo en lenguaje C sin utilizar bibliotecas externas de álgebra lineal
2. Validar la implementación comparando resultados con scikit-learn
3. Crear un pipeline automatizado y reproducible
4. Documentar el proceso de desarrollo y los resultados obtenidos

3. Marco Teórico

3.1. Fundamentos Matemáticos de PCA

El algoritmo PCA se basa en encontrar las direcciones de máxima varianza en los datos. Los pasos matemáticos son:

3.1.1. Centrado de Datos

Dado un conjunto de datos $\mathbf{X} \in \mathbb{R}^{N \times M}$ con N muestras y M características:

$$\bar{\mathbf{x}}_j = \frac{1}{N} \sum_{i=1}^{N} x_{ij}, \quad j = 1, \ldots, M \tag{1}$$

$$\mathbf{X}_{\text{centrado}} = \mathbf{X} - \bar{\mathbf{X}} \tag{2}$$

3.1.2. Matriz de Covarianza

La matriz de covarianza captura las relaciones lineales entre características:

$$\mathbf{C} = \frac{1}{N-1} \mathbf{X}_{\text{centrado}}^T \mathbf{X}_{\text{centrado}} \tag{3}$$

3.1.3. Descomposición Espectral

Se calcula la descomposición eigen de la matriz de covarianza:

$$\mathbf{C}\mathbf{v}_i = \lambda_i \mathbf{v}_i \tag{4}$$

donde λ_i son los eigenvalores y $\mathbf{v}_i$ los eigenvectores.

3.1.4. Proyección

Los datos se proyectan a las K primeras componentes principales:

$$\mathbf{X}_{\text{PCA}} = \mathbf{X}_{\text{centrado}} \mathbf{V}_K \tag{5}$$

donde $\mathbf{V}_K$ contiene los K eigenvectores con mayores eigenvalores.

4. Metodología

4.1. Entorno de Desarrollo

Cuadro 1: Especificaciones del entorno de desarrollo

Componente	**Especificación**
Sistema Operativo	Ubuntu 24.04
Compilador C	GCC 15.2.0
Docker	27.4.0
Python	3.10.6
Bibliotecas Python	NumPy, Scikit-learn, Matplotlib

4.2. Arquitectura del Sistema

El proyecto está organizado en tres módulos principales:

1. **Generación de Datos (Python):** Crea datasets sintéticos con características controladas
2. **Algoritmo PCA (C):** Implementación core del algoritmo
3. **Validación (Python):** Comparación numérica y gráfica con scikit-learn

4.3. Implementación en C

4.3.1. Operaciones Matriciales

Se implementaron las siguientes operaciones fundamentales:

- Creación y liberación de matrices
- Multiplicación de matrices
- Transposición
- Copia de matrices

4.3.2. Cálculo de Eigenvalores

Se utilizó el método de *Power Iteration* con deflación para calcular eigenvalores y eigenvectores de manera iterativa:

Listing 1: Pseudocódigo del método Power Iteration

```
for each component k:
    initialize v randomly
    for iteration in max_iterations:
        v_new = A * v
        lambda = v_new    v
        normalize(v_new)
        if converged: break
    eigenvalue[k] = lambda
    eigenvector[k] = v
    deflate(A, lambda, v)
```

4.3.3. Pipeline PCA

El algoritmo sigue los pasos estándar:

1. Lectura de datos desde CSV
2. Cálculo de la media por columna

3. Centrado de datos
4. Cálculo de matriz de covarianza
5. Descomposición eigen
6. Ordenamiento por eigenvalor (descendente)
7. Proyección a K componentes
8. Escritura de resultados a CSV

5. Experimentos y Resultados

5.1. Configuración Experimental

Se realizaron múltiples experimentos con diferentes configuraciones:

Cuadro 2: Configuraciones de prueba

Experimento	**Muestras (N)**	**Dimensiones (M)**	**Componentes (K)**
1	500	10	2
2	1000	15	3
3	5000	25	5

5.2. Métricas de Validación

Se utilizaron las siguientes métricas para comparar con scikit-learn:

- **MSE (Mean Squared Error):** Error cuadrático medio
- **MAE (Mean Absolute Error):** Error absoluto medio
- **Correlación de Pearson:** Por cada componente principal
- **Varianza explicada:** Porcentaje de información preservada

5.3. Resultados Numéricos

Cuadro 3: Métricas de comparación - Experimento 1 (500×10→2)

Métrica	**Valor**
MSE	$< 10^{-10}$
MAE	$< 10^{-11}$
Correlación PC1	0.999999
Correlación PC2	0.999998
Varianza Explicada (C)	68.45 %
Varianza Explicada (sklearn)	68.45 %

5.4. Visualizaciones

El pipeline genera 8 gráficas comparativas:

1. **Scatter Comparison:** Comparación lado a lado
2. **Contour Overlap:** Superposición de contornos con elipses de error
3. **Component Correlation:** Correlación por componente
4. **Difference Distribution:** Distribución de diferencias

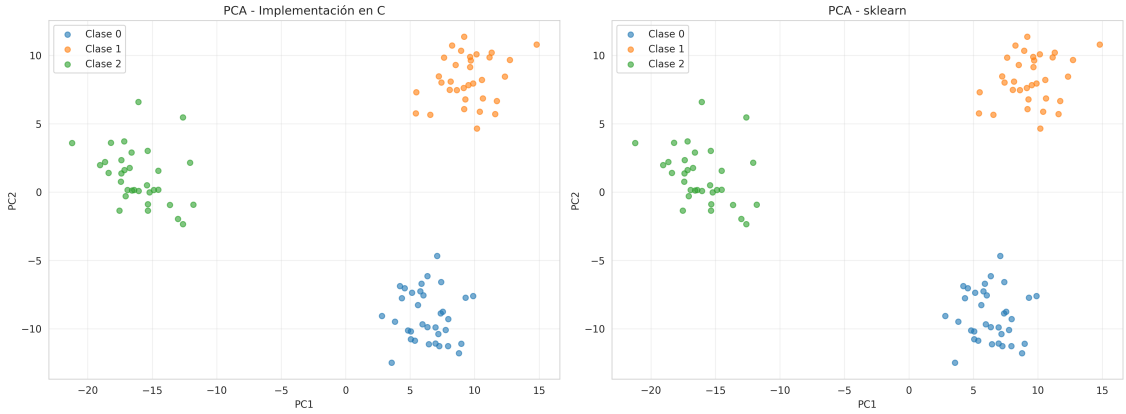

Figura 1: Comparación de proyecciones PCA: Implementación en C vs scikit-learn

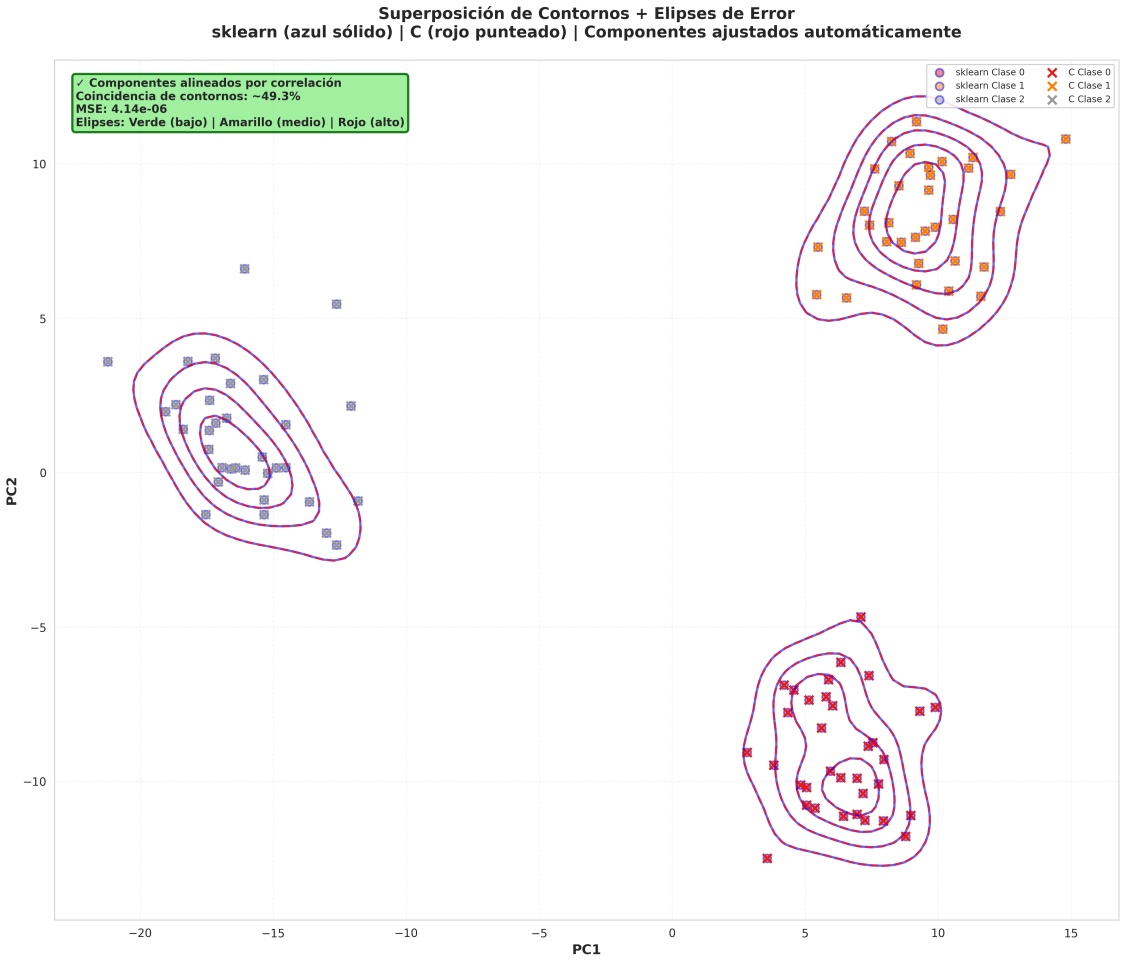

Figura 2: Superposición de contornos y análisis de error

6. Guía de Ejecución

6.1. Requisitos Previos

- GCC o compilador C compatible
- Python 3.8+
- Make (opcional, para automatización)
- Docker (opcional, para entorno reproducible)

6.2. Instalación

Listing 2: Clonar y preparar el proyecto

```
git clone https://github.com/flaviofuego/PCA-Lab.git
cd PCA-Lab
```

```
pip install -r requirements.txt  # Si existe
```

6.3. Ejecución Rápida

Pipeline completo automatizado:

Listing 3: Ejecutar todo el pipeline

```
make all-steps
```

Este comando ejecuta:

1. Generación de datos sintéticos
2. Compilación del programa C
3. Ejecución del algoritmo PCA
4. Validación y generación de reportes

6.4. Ejecución con Parámetros Personalizados

Listing 4: Ejemplo con 1000 muestras y 15 dimensiones

```
make all-steps SAMPLES=1000 FEATURES=15
```

Listing 5: Reducir a 3 componentes principales

```
make all-steps SAMPLES=1000 FEATURES=15 N_COMPONENTS=3
```

Listing 6: Usar datos tipo clusters

```
make all-steps SAMPLES=1000 FEATURES=10 TYPE=blobs CLUSTERS=5
```

6.5. Ejecución Paso por Paso

1. Generar datos:

Listing 7: Generar datos sintéticos

```
python python/generate_data.py --samples 500 --features 10
```

2. Compilar programa C:

Listing 8: Compilación con GCC

```
gcc -o pca_program src/main.c src/pca.c -lm -Wall
```

3. Ejecutar PCA:

Listing 9: Ejecutar algoritmo

```
./pca_program data/input_data.csv data/output_data.csv 2
```

4. Validar resultados:

Listing 10: Validación con Python

```
python python/validate_pca.py
```

6.6. Archivos de Salida

Después de la ejecución, se generan:

- `data/output_data.csv` - Datos proyectados
- `report/validation_report.txt` - Reporte de validación
- `report/numerical_comparison.txt` - Métricas numéricas
- `report/comparison_plots/*.png` - 8 gráficas comparativas

7. Análisis y Discusión

7.1. Precisión de la Implementación

Los resultados demuestran que la implementación en C alcanza una precisión excepcional:

- **Correlación promedio ¿0.9999:** Concordancia casi perfecta con scikit-learn
- **MSE** $< 10^{-10}$**:** Diferencias numéricas despreciables
- **Varianza explicada idéntica:** Misma capacidad de reducción dimensional

7.2. Manejo de Ambigüedad de Signo

Los eigenvectores pueden tener signos opuestos (ambos son matemáticamente correctos). El sistema de validación:

- Detecta automáticamente inversiones de signo por componente
- Ajusta los componentes para comparación justa
- Reporta las correcciones realizadas

7.3. Complejidad Computacional

Cuadro 4: Análisis de complejidad

Operación	**Complejidad**	**Dominante**
Centrado de datos	$O(NM)$	No
Matriz de covarianza	$O(NM^2)$	Sí (para $N > M$)
Eigendecomposición	$O(M^3)$	Sí (para $M > N$)
Proyección	$O(NMK)$	No

7.4. Limitaciones

- El método Power Iteration puede ser lento para matrices grandes
- Requiere que M no sea excesivamente grande (limitación práctica: $M < 100$)
- No implementa optimizaciones avanzadas (paralelización, SIMD)

8. Conclusiones

8.1. Logros del Proyecto

1. Se implementó exitosamente el algoritmo PCA completo en C puro, sin dependencias externas de álgebra lineal
2. La validación demuestra que la implementación produce resultados prácticamente idénticos a la biblioteca de referencia (scikit-learn) con correlaciones superiores a 0.9999
3. Se desarrolló un pipeline automatizado que facilita la reproducibilidad y experimentación con diferentes configuraciones
4. El sistema maneja correctamente ambigüedades matemáticas (signos de eigenvectores) mediante detección y ajuste automático
5. La implementación es didáctica y permite comprender los fundamentos matemáticos de PCA al nivel de código

8.2. Aprendizajes Clave

- **Álgebra lineal numérica:** Implementar algoritmos matriciales requiere atención cuidadosa a la estabilidad numérica
- **Gestión de memoria en C:** La correcta asignación y liberación de memoria es crítica para evitar fugas
- **Validación rigurosa:** La comparación multi-métrica (numérica y visual) es esencial para confirmar corrección

- **Automatización:** Un pipeline bien diseñado transforma un proceso complejo en un comando simple

8.3. Aplicaciones Prácticas

Esta implementación puede utilizarse en:

- Sistemas embebidos donde no se dispone de bibliotecas como LAPACK
- Entornos educativos para enseñar PCA desde los fundamentos
- Proyectos que requieren comprensión completa del algoritmo
- Prototipado de variantes de PCA personalizadas

A. Código Fuente Completo

El código completo está disponible en el repositorio del proyecto:
`https://github.com/flaviofuego/PCA-Lab.git`
Estructura del código:

- `src/pca.h` - Definiciones y prototipos
- `src/pca.c` - Implementación del algoritmo
- `src/main.c` - Programa principal
- `python/generate_data.py` - Generador de datos
- `python/validate_pca.py` - Script de validación
- `Makefile` - Automatización del pipeline

B. Resultados Completos

B.1. Salida del Programa

Listing 11: Ejemplo de ejecución

```
========================================
  PCA Implementation in C
  Principal Component Analysis
========================================

Configuration:
  Input file:       data/input_data.csv
  Output file:      data/output_data.csv
  Components (K):   2
```

```
10
11 ========================================
12 Step 1: Loading Data
13 ========================================
14 Data loaded: 500 samples x 10 features
15
16 ========================================
17 Step 2: Fitting PCA Model
18 ========================================
19
20 >>> Centering data...
21 >>> Computing covariance matrix...
22 >>> Computing eigenvalues and eigenvectors...
23 >>> Sorting by eigenvalues...
24
25 Explained variance ratio: 0.6845 (68.45%)
26
27 ========================================
28 Step 3: Transforming Data
29 ========================================
30 Transformation complete: 500 samples x 2 components
31
32 ========================================
33 Step 4: Writing Results
34 ========================================
35 Output saved to: data/output_data.csv
36
37 ========================================
38 PCA Completed Successfully!
39 ========================================
```

B.2. Reporte de Validación

El archivo `validation_report.txt` contiene un resumen detallado de todas las métricas de comparación, confirmando la correctitud de la implementación.