Inpainting con SVD

Objetivos

General

Comprender, aplicar y analizar el método de restitución de imágenes incompletas mediante aproximación de rango bajo usando la descomposición en valores singulares (SVD).

Específicos

- Revisar brevemente la teoría de matrices de rango bajo, contextualizando su papel en la reconstrucción de datos incompletos.
- Implementar un algoritmo iterativo de completado de matrices para recuperar imágenes dañadas o incompletas, preservando los valores conocidos y reconstruyendo las regiones faltantes,
- Evaluar la calidad de la restitución mediante métricas cuantitativas y análisis visual comparativo.
- (Opcional) Explorar la aplicación del método tanto en imágenes con daño artificial como en imágenes deterioradas o incompletas provenientes de fuentes reales.

Introducción

En diversos campos del procesamiento de imágenes y los datos, es común enfrentarse a situaciones donde la información está incompleta o deteriorada. Este problema aparece en escenarios como:

- Imágenes antiguas o escaneadas con zonas dañadas,
- Fotografías transmitidas con pérdida de información,
- Datos corrompidos o con ruido impulsivo,
- Defectos en sensores o fallos de almacenamiento.

La restitución de imágenes incompletas, también conocida como inpainting o completado de matrices, busca reconstruir las regiones faltantes de una imagen basándose en la información disponible. Existen múltiples enfoques para abordar este problema, desde métodos basados en interpolación hasta técnicas avanzadas de aprendizaje profundo.

En este notebook, exploraremos un enfoque clásico y eficiente basado en la descomposición en valores singulares (SVD). La idea fundamental es que muchas imágenes reales presentan una estructura redundante y pueden aproximarse por matrices de rango bajo. Esta propiedad se

puede aprovechar para recuperar los valores faltantes buscando la mejor aproximación de rango bajo compatible con los datos observados.

Planteamiento del problema de restitución

Dado una imagen incompleta representada como una matriz $A_{\text{obs}} \in \mathbb{R}^{m \times n}$, queremos recuperar una aproximación de la imagen completa A.

Se dispone de una **máscara de observación** $M \in \{0,1\}^{m \times n}$, donde:

- si $M_{i,j} = 1$: el valor $A_{i,j}$ es conocido.
- si $M_{i,j} = 0$: el valor $A_{i,j}$ es desconocido y debe ser estimado.

Objetivo: Encontrar una matriz A_{rec} que:

- Coincida con los valores conocidos: $(A_{\text{rec}})_{i,j} = (A_{\text{obs}})_{i,j} \quad \text{si } M_{i,j} = 1$
- Sea una **aproximación de rango bajo** de la imagen original.

Hipótesis: La imagen original puede ser aproximada por una matriz de **rango bajo** debido a su estructura redundante.

Estrategia: Resolver el problema mediante un **método iterativo** basado en SVD, que reconstruye las regiones faltantes aproximando la matriz completa al rango deseado k.

Teoría

Rango de una matriz

El rango de una matriz corresponde al número máximo de columnas (o filas) linealmente independientes que tiene la matriz. Es una medida de la *dimensión* del espacio generado por sus columnas o filas.

El rango de una matriz también corresponde al número de valores singulares no nulos de la matriz. Por lo que el rango de la matriz indica cuánta información relevante tiene la matriz.

Aproximación óptima mediante SVD

Teorema de Eckart–Young:

La mejor aproximación de rango k a A en norma 2 (o Frobenius) es
$A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T = U_k \sigma_k V_k^T$.

Interpretación:

El teorema dice que entre todas las matrices de rango k, la matriz A_k minimiza la distancia a A en términos de norma Frobenius y norma 2.

Inpainting iterativo por SVD

El inpainting de imágenes consiste en reconstruir las partes faltantes o dañadas de una imagen utilizando la información existente en la misma.

Este proceso se basa en la idea de que muchas imágenes (o matrices de datos) contienen información redundante y, por tanto, pueden representarse bien con una aproximación de bajo rango. Esto significa que los valores conocidos contienen suficiente estructura para predecir los valores faltantes.

El problema se convierte entonces en un completado de matrices de rango bajo.

🔧 **Máscara de observación**

Para saber qué valores de la imagen se deben reconstruir, se utiliza una máscara binaria M:

- Si el píxel (i,j)(i,j) es conocido, $M[i,j] = 1$.
- Si el píxel (i,j)(i,j) es faltante (se debe reconstruir), $M[i,j] = 0$.

🔁 **Esquema del algoritmo iterativo:**

Se realiza un proceso iterativo donde se alterna entre:

- Aproximar la imagen incompleta mediante una matriz de rango bajo usando SVD truncado.
- Restaurar los valores conocidos en cada iteración para respetar los datos originales.

📝 Algoritmo paso a paso

Recibe una imagen A, una máscara M, un rango máximo k y un máximo de iteraciones max_iter.

1. **Inicialización**: Comenzar con una copia A_{rec} de la imagen A. Los valores faltantes pueden inicializarse arbitrariamente (por ejemplo, en cero).
2. **Repetir hasta alcanzar el número máximo de iteraciones**:

   a. **Descomposición SVD**: Obtener la descomposición en valores singulares de la matriz actual A_{rec}:

   $$A_{\text{rec}} = U\Sigma V^T$$

   b. **Truncado de valores singulares**: Conservar solo los k valores singulares más grandes en Σ. El resto se remplaza por ceros.

   c. **Reconstrucción de rango bajo**: Construir la matriz aproximada de rango k:

   $$A_{\text{lowrank}} = U_k \Sigma_k V_k^T$$

   d. **Actualización de los valores faltantes**: Actualizar los valores de A_{rec}:
   - Los **píxeles conocidos** se mantienen igual (se respetan los datos originales).
   - Los **píxeles faltantes** se completan con los valores de A_{lowrank}.

3. **Resultado final**: La matriz A_{rec} contiene la imagen reconstruida tras las iteraciones.

Implementación

```
begin
    using LinearAlgebra
    using Images
    using ImageIO
    using Plots
    using Statistics
end
```

Generar máscara de observación

`generate_mask`

Genera una máscara binaria de observación de zonas blancas en una imagen en escala de grises.

Argumentos:

- `img_gray::Array{Gray}`: Imagen en escala de grises.
- `threshold::Float64=0.95`: Umbral para detectar zonas faltantes.
  - Píxeles con valor >= threshold se consideran dañados o faltantes.

Retorna `M::Array{Float64}`: Máscara binaria donde 1.0 indica píxel conocido y 0.0 indica píxel faltante.

Inicialización

Podemos inicializar los valores faltantes de cada matriz a un valor promedio de su fila o columna, esto permite dar un mejor punto de partida al algoritmo

`initialize_missing`

Rellena los valores faltantes de `A_obs` usando una estimación inicial.

Argumentos:

- `A_obs`: Matriz de la imagen observada.
- `M`: Máscara de observación (1 conocido, 0 faltante).
- `method`: `"row"`, `"col"` o `"global"`.

Retorna la matriz inicializada.

```
# Explicar por qué sirve para inpainting
# De dónde salió el alg?
# utilizar SVD propio
# Límite miércoles
```

Aplicar algoritmo de in-painting por SVD

¿Qué hace el algoritmo?

Es un algoritmo iterativo que rellena los valores faltantes de una matriz (por ejemplo, una imagen dañada). Para esto busca la matriz más sencilla (suavizada) que:

- Respete los valores conocidos.
- Tenga rango bajo.

¿Por qué funciona?

- El SVD captura la estructura global del algoritmo,
- el truncado de rango impone regularidad y elimina el ruido,
- las iteraciones sucesivas fuerzan que la solución sea consistente con datos conocidos

Con funciones de Julia

`svd_inpainting`

Completa una imagen incompleta utilizando un método iterativo basado en SVD. Busca la mejor aproximación de rango bajo que coincida con los valores conocidos

Argumentos:

- `A_obs::Array{Float64}`: Matriz de la imagen observada (con valores faltantes).
- `M::Array{Float64}`: Máscara binaria de observación.
  - `M[i,j] = 1` si el píxel (i,j) es conocido.
  - `M[i,j] = 0` si el píxel (i,j) es faltante.
- `k::Int=50`: Número de valores singulares conservados (rango deseado).
- `max_iter::Int=30`: Número máximo de iteraciones.

Retorna la matriz reconstruida de la imagen `A_rec::Array{Float64}`.

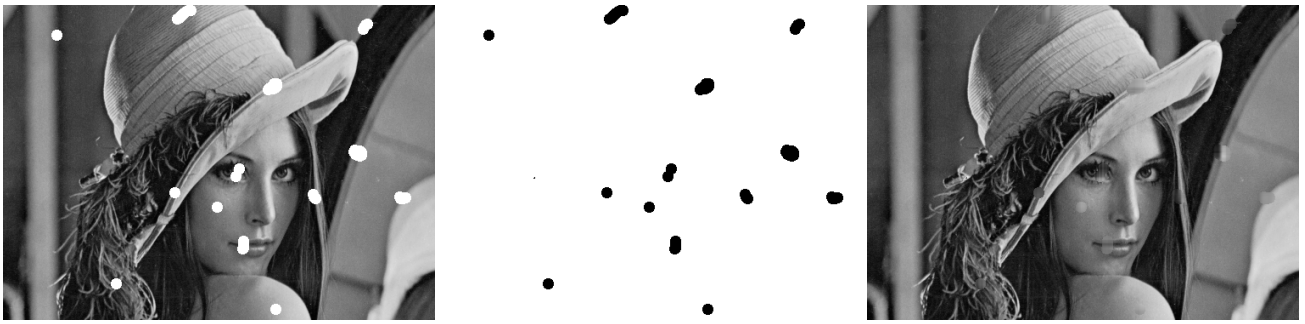
```
"""
Completa una imagen incompleta utilizando un método iterativo basado en SVD.
Busca la mejor aproximación de rango bajo que coincida con los valores conocidos

Argumentos:
- `A_obs::Array{Float64}`: Matriz de la imagen observada (con valores faltantes).
- `M::Array{Float64}`: Máscara binaria de observación.
    - `M[i,j] = 1` si el píxel (i,j) es conocido.
    - `M[i,j] = 0` si el píxel (i,j) es faltante.
- `k::Int=50`: Número de valores singulares conservados (rango deseado).
- `max_iter::Int=30`: Número máximo de iteraciones.

Retorna la matriz reconstruida de la imagen `A_rec::Array{Float64}`.
"""
function svd_inpainting(A_obs::Matrix{Float64}, M::Matrix{Float64}, k::Int=50, max_iter::Int=30)
    # Inicializamos la matriz de reconstrucción con los datos observados
    A_rec = copy(A_obs)

    for iter in 1:max_iter
        # Descomposición SVD de la imagen actual
        U, S, V = svd(A_rec)

        # Truncamos los valores singulares a rango k
        S_trunc = Diagonal(vcat(S[1:k], zeros(length(S) - k)))

        # Reconstruimos la versión de rango bajo
        A_lowrank = U * S_trunc * V'

        # Actualizamos: mantenemos los píxeles conocidos y sustituimos los faltantes
        A_rec = M .* A_obs + (1 .- M) .* A_lowrank
    end

    return A_rec
end
```

Ejemplos

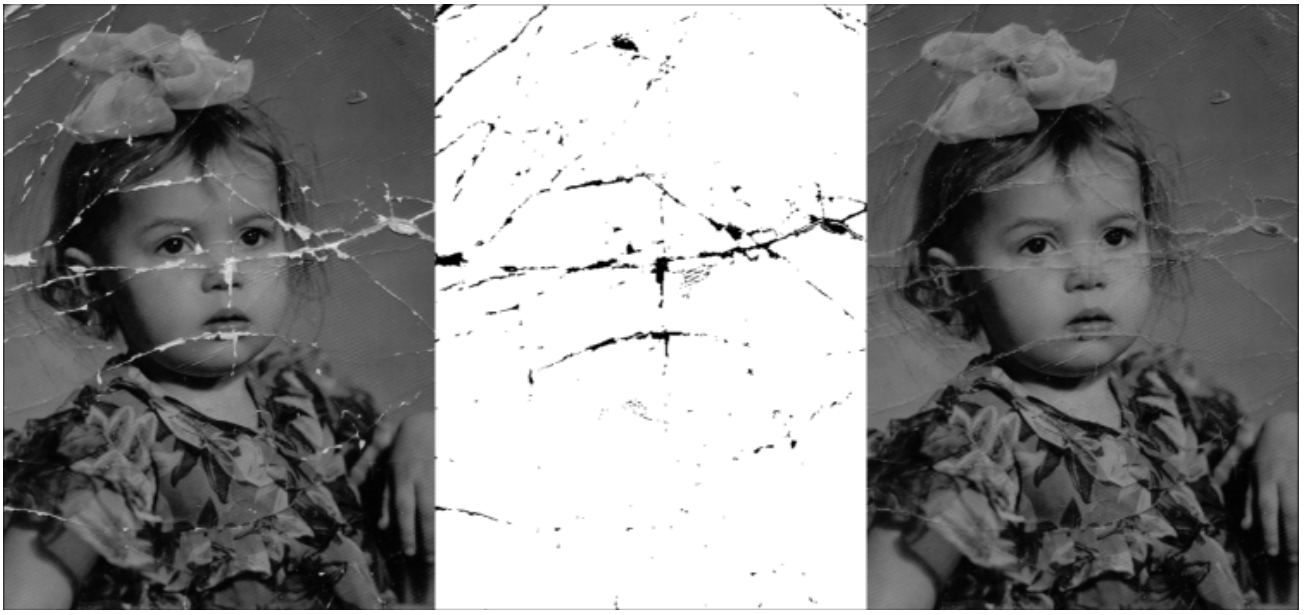
A continuación veremos dos ejemplos de imágenes, la primera ha sido dañada artificialmente y la segunda ha tenido daño real.

En las gráficas podremos ver, en orden:

- la imagen original
- la máscara utilizada
- un posible resultado.



```
begin
    matrix_ex1 = Float64.(Gray.(load("imagen_ejemplo.png")))
    mask_ex1 = generate_mask(matrix_ex1, .95)
    init_ex1 = initialize_missing(matrix_ex1, mask_ex1; method="global");

    mosaicview(
    Gray.(matrix_ex1),
    Gray.(mask_ex1),
    Gray.(svd_inpainting(matrix_ex1, mask_ex1, 10, 10));
    ncol=3
)
end
```



```
begin
    matrix_ex2 = Float64.(Gray.(load("imagen_ejemplo_6.jpeg")))
    mask_ex2 = generate_mask(matrix_ex2, .6)
    init_ex2 = initialize_missing(matrix_ex2, mask_ex2; method="global");

    mosaicview(
    Gray.(matrix_ex2),
    Gray.(mask_ex2),
    Gray.(svd_inpainting(matrix_ex2, mask_ex2, 10, 10));
    ncol=3
)
end
```

Experimentos y análisis

Con daño simulado:

- calcular errores contra la original

Con daño real:

- mostrar que se puede hacer (análisis subjetivo)

Ambos:

- Comparación visual para diferentes valores en la restitución (k y max_iter)

```
md"
## Experimentos y análisis
Con daño simulado:
* calcular errores contra la original

Con daño real:
* mostrar que se puede hacer (análisis subjetivo)

Ambos:
* Comparación visual para diferentes valores en la restitución (k y max_iter)
"
```

`reconstruct_all`

Aplica SVD inpainting a cada inicialización.

Argumentos:

- `initializations` : Diccionario de matrices inicializadas.
- `mask` : Máscara binaria.

Retorna un diccionario de reconstrucciones.

`show_reconstructions`

Muestra las reconstrucciones en un `mosaicview` en formato 2x2.

Argumentos:

- `recs` : Diccionario de imágenes reconstruidas.
- `ncol` : Número de columnas en el mosaico

Comparación de inicializaciones para diferentes valores de k

`generate_initializations`

Genera distintas inicializaciones para los valores faltantes de `A`.

Retorna un diccionario con las inicializaciones: `:row`, `:col`, `:global`, `:no_init`

`compare_initilizations`

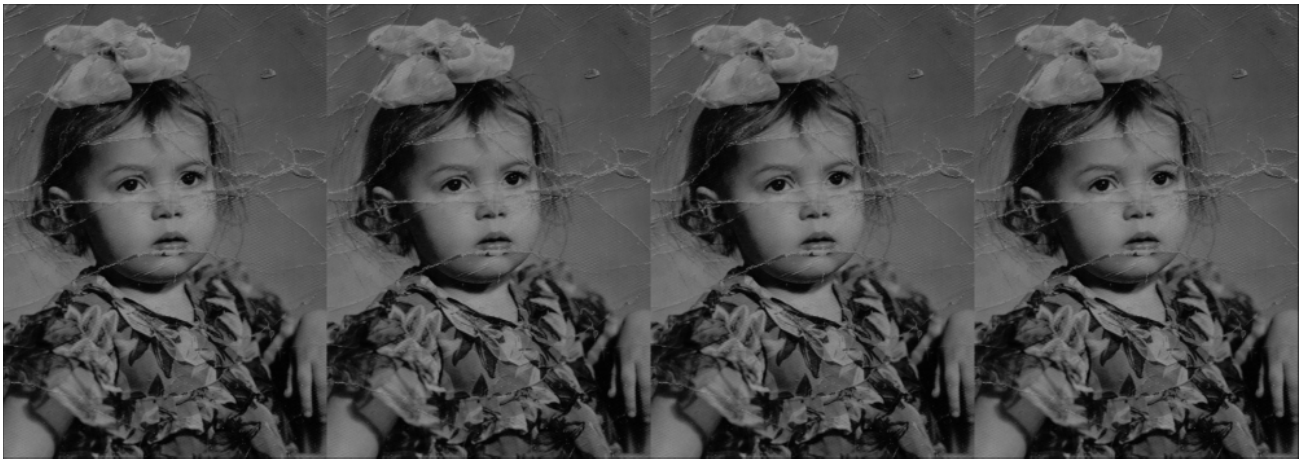
Ejecuta todo el proceso de comparación.

```
"""
Ejecuta todo el proceso de comparación.
"""
function compare_initilizations(A, mask; k=50, max_iter=10, ncol=4,
verbose=false)
    inits = generate_initializations(A, mask)
    recs = reconstruct_all(inits, mask; k=k, max_iter=max_iter)
    if verbose
        println("Orden de las imágenes: ", collect(keys(recs)))
    end
    show_reconstructions(recs, ncol=ncol)
end
```

Para $k = 5$:

```
compare_initilizations(matrix_ex1, mask_ex1; k=5, verbose=true)
```

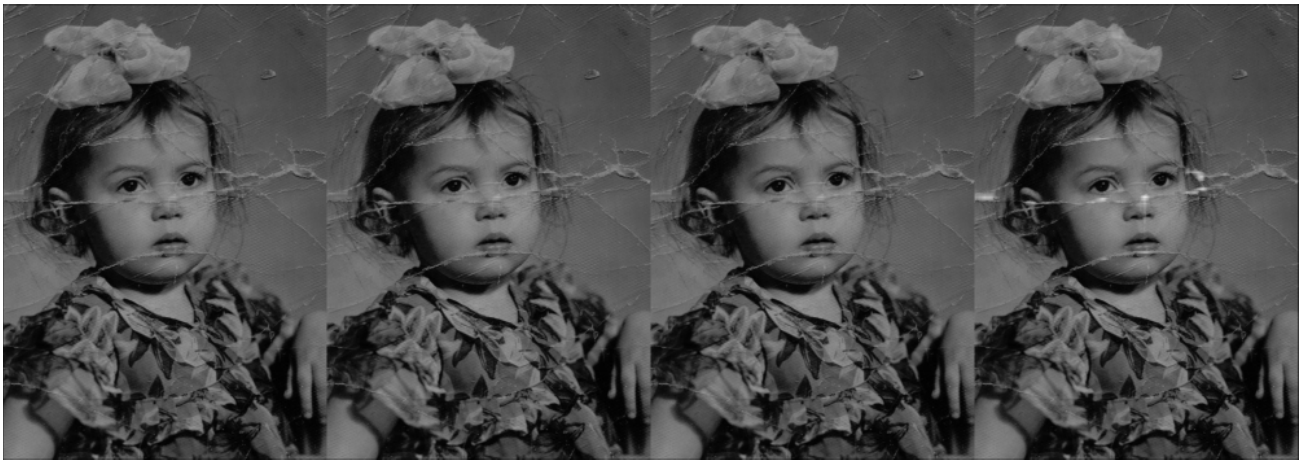
```
Orden de las imágenes: Any[:global, :row, :col, :no_init]
```



```
compare_initilizations(matrix_ex2, mask_ex2; k=5)
```

Para $k = 30$:

```
compare_initilizations(matrix_ex1, mask_ex1; k=30)
```



```
compare_initilizations(matrix_ex2, mask_ex2; k=30)
```

Resultado:

Para $k = 5$, en ambas imágenes la inicialización o no inicialización de la imagen no genera cambios remarcables.

Para $k = 30$, los huecos tienden al color de base, por lo que la imagen no inicializada es más claramente erronea. Sin embargo no se ven diferencias mayores entre los 3 métodos de inicialización, por lo que solo vamos a utilizar el global de ahora en adelante.

Diferentes valores de k

Queremos un mosaico que muestre las imágenes con k=5,10,20,40 por e.g.

Veremos si se cumple que:

- Un k pequeño implica una imagen muy suavizada. Solo se preservan patrones globales y se pierde el detalle.
- Un k intermedio reconstruye patrones generales y algo de textura.
- Un k grande preserva detalles finos, pero también amplifica errores si el daño es severo.

```
mosaicview(
    Gray.(svd_inpainting(init_ex1, mask_ex1, 5)),
    Gray.(svd_inpainting(init_ex1, mask_ex1, 10)),
    Gray.(svd_inpainting(init_ex1, mask_ex1, 30)),
    Gray.(svd_inpainting(init_ex1, mask_ex1, 75));
    ncol = 4
)
```

```
mosaicview(
    Gray.(svd_inpainting(init_ex2, mask_ex2, 5)),
    Gray.(svd_inpainting(init_ex2, mask_ex2, 10)),
    Gray.(svd_inpainting(init_ex2, mask_ex2, 30)),
    Gray.(svd_inpainting(init_ex2, mask_ex2, 75));
    ncol = 4
)
```

Resultado

En efecto, k se balancea entre ser muy suave (siguiendo los patrones generales más fuertes) y estar sobreajustado.

En la imagen dañada artificialmente, que tiene daños redondeados y grandes, el valor mediano de $k = 30$. En la imagen real, que tiene daños largos y lineares, los valores menores de $k = 5, 10$ actuaron mejor.

Reflexión y aprendizajes

El principal aprendizaje que me dejó este notebook fue el de cambiar de enfoque o de tarea cuando me encuentro bloqueada con algún problema. Trabajar de forma modular y avanzar en otras partes del proyecto mientras maduro una idea resultó ser una estrategia muy útil.

El proyecto me permitió ver la importancia de la matriz SVD al ver cómo captura globalmente la información de una matriz. Además pude aplicar de forma práctica el teorema de las aproximaciones óptimas del SVD.

Declaración sobre el uso de inteligencia artificial y fuentes externas

Durante la realización de este trabajo se utilizó **asistencia de inteligencia artificial (IA)** para generar y mejorar partes del código y de los textos explicativos, con el fin de estructurar mejor el contenido y optimizar la implementación.

Uso de IA:

Se utilizó **ChatGPT (OpenAI)** de forma activa para:

- Redacción y revisión de objetivos, introducción y estructura del notebook.
- Comprensión de conceptos y algoritmos.
- Generación y refactorización de código en Julia, en particular:
    - **Inicialización de valores faltantes con medias locales.**
    - **Algoritmo iterativo de SVD inpainting.**
    - **Construcción de funciones modulares para comparación entre métodos.**
    - **Visualización de resultados con etiquetas usando `mosaicview` y `Luxor.jl`.**

Algunos prompts utilizados:

- "Puedes explicarme el teorema de Eckart-Young?"
- “Explicame el algoritmo de inpainting por SVD.”
- “Quiero imprimir un mosaico de restituciones usando el método de inicialización global para diferentes valores de k.”

Fuentes externas consultadas:

- Marangoz, S. (2023). *Image Imputation with SVD*. https://salihmarangoz.github.io/blog/Image-Imputation-with-SVD
- Golub, G., & Van Loan, C. (2013). *Matrix Computations* (4th ed.). Johns Hopkins University Press.