

# Métodos Probabilísticos para la Inteligencia Artificial

## Proyecto Final

Arturo Márquez Flores

Maestría en Inteligencia Artificial

Universidad Veracruzana

CIIA – Centro de Investigación en Inteligencia Artificial

Sebastián Camacho No 5, Xalapa, Ver., México 91000

[arturomf94@gmail.com](mailto:arturomf94@gmail.com)

<https://github.com/arturomf94/mp-mia>

14 de Diciembre del 2018

## 1. Introducción

Este proyecto final consiste en el desarrollo de un programa que elimine ruido uniforme (también conocido como *sal y pimienta*) de imágenes binarias<sup>1</sup>. El programa desarrollado aquí puede tomar como input cualquier imagen (de cualquier tamaño) y retornar otra imagen  *limpia* después de añadir ruido a su versión binaria y escalada. En este reporte se describe el modelo utilizado, así como el programa desarrollado y algunos ejemplos de ilustración.

## 2. Modelo

El modelo utilizado para el programa de eliminación de ruido consiste en utilizar la representación de una imagen como un modelo gráfico no-dirigido (en particular, un campo aleatorio de Markov) definido por un grafo no-dirigido  $G = (V, E)$  y un conjunto de variables aleatorias  $W = (W_v)_{v \in V}$  cuya distribución puede expresarse como el producto de *funciones potenciales*  $\phi[W]$  [Pri12]. De tal manera que se tiene lo siguiente:

$$Pr(W) = \frac{1}{Z} \prod_{c=1}^C \phi_c[W]$$

---

<sup>1</sup>Aunque este método puede extenderse a el caso no-binario, por simplicidad, se mantiene el supuesto de que cada pixel en la imagen toma un valor de dos opciones: 0 o 1 (negro o blanco)

En particular, este tipo de modelos satisfacen las *propiedades de Markov* [Wik18a], que indican que:

- Dos variables no adyacentes cualesquiera son condicionalmente independientes dadas todas las demás variables.
  - Una variable es condicionalmente independiente de todas las otras variables dadas sus vecinos.
  - Dos subconjuntos de variables cualesquiera son condicionalmente independientes dado un subconjunto que las separa.

Esto quiere decir que estos modelos son útiles para representaciones de imágenes, ya que establecen relaciones locales entre cada pixel y sus vecinos. Además, existen métodos de inferencia bien establecidos para estos modelos. Por ejemplo, se puede a ser inferencia por *máximo a posteriori*, la cual consiste en estimar los parámetros que maximizan la probabilidad condicional dada una observación de un conjunto de datos. Es decir, si  $W = (W_v)_{v \in V}$  es un conjunto de parámetros con distribución condicional  $Pr(W|X = x)$ , entonces el conjunto  $\hat{w} = (\hat{w}_v)_{v \in V}$  que maximiza  $Pr(W|X = x)$  es una estimación por *máximo a posteriori* [Wik18b].

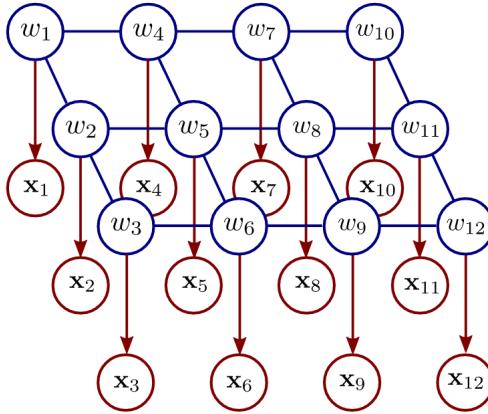


Figura 1: Modelo

En la Figura 1<sup>2</sup> podemos observar un ejemplo ilustrativo del modelo. En particular, en los campos aleatorios de Markov con esta forma, la factorización de  $Pr(W)$  puede ser de tal manera que:

$$Pr(W) = \frac{1}{Z} \prod_{i,j \in E}^C \phi_{i,j}[w_i, w_j]$$

<sup>2</sup>Imagen tomada de [Pri12].

Por lo tanto, el problema de maximización de  $\Pr(W|X = x)$  se puede expresar de la siguiente forma:

$$\hat{w} = \operatorname{argmin}_w \sum_{v \in V} U_v(w_v) + \sum_{(i,j) \in E} P_{i,j}(w_i, w_j)$$

Aquí,  $U_v(w_v)$  es el *término unario* que puede interpretarse como una medición de la *compatibilidad* del dato  $x_v$  con el parámetro  $w_v$ . Por otro lado  $P_{i,j}(w_i, w_j)$  es el *término de pares* que puede interpretarse como una medición de la compatibilidad entre  $w_i$  y  $w_j$ .

Para el caso binario donde, para cada  $v \in V$ ,  $w_v$  es 0 o 1, podemos definir que para cualquier par  $i, j \in V$ :  $P_{i,j}(0, 0) = P_{i,j}(1, 1) = 0$  y  $P_{i,j}(1, 0) = P_{i,j}(0, 1) = \theta$ .

Este es un problema de optimización que puede resolverse mediante el método de *corte mínimo*, que ya fue revisado en clase. Hacer esto consiste en construir un grafo con pesos, tal que resolverlo mediante el *corte mínimo* sea equivalente a la inferencia por *máximo a posteriori*. Podemos observar en la Figura 2 un grafo con esta estructura que ilustra este método.

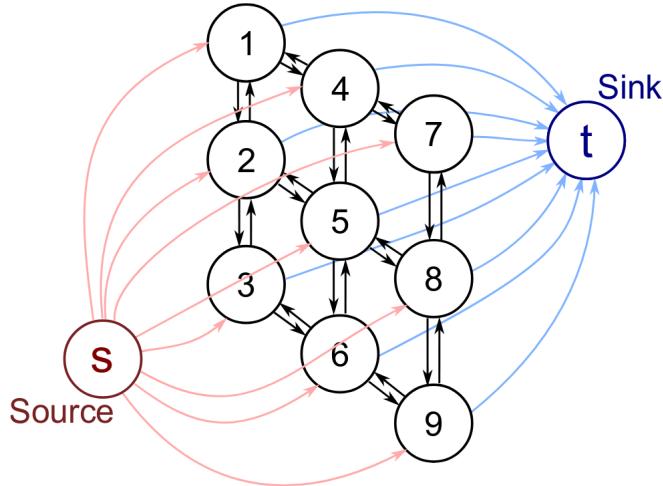


Figura 2: Corte Mínimo

Con el método de corte mínimo podemos entonces estimar  $w$ , para eliminar el ruido observado en una imagen. La siguiente sección describe una implementación de este método en Python.

### 3. Implementación en Python

El código puede encontrarse en [este repositorio](#), donde además se incluyen unas imágenes que sirven como ejemplo. El script principal es **main.py**. Para que funcione, se deben guardar las imágenes que se quieren utilizar en la carpeta llamada **original\_images**. Este script toma la imagen, la escala, la convierte a una imagen binaria y la guarda en la carpeta **binary\_images**. Después, se agrega ruido *uniforme* sobre la imagen, con la función en el script llamado **noise.py**. La imagen creada por este script corresponde a la *observación* del modelo gráfico. Por lo tanto, con esta imagen se construye el grafo que se utilizará para resolver el *corte mínimo*. La creación del grafo se hace con las funciones en el script llamado **graph.py**. Una vez que se construye el grafo, se implementa la función de corte mínimo de la librería *networkx*. Esto nos permite restaurar la imagen; es decir, eliminar su ruido. Los siguientes puntos son descripciones breves de los parámetros utilizados en el programa:

- **width\_objective** - Es la base (en pixeles) de la imagen que se va a trabajar.
- **noise\_percentage = .01** - La proporción de pixeles que son afectados por el ruido.
- **pairwise\_cost = 1** - El costo de pares entre nodos.
- **unary\_matching\_cost\_source = .9** - El costo del eje que une la fuente y un pixel con su mismo valor.
- **unary\_matching\_cost\_sink = .9** - El costo del eje que une el destino y un pixel con su mismo valor.
- **unary\_nonmatching\_cost\_source = 1** - El costo del eje que une la fuente y un pixel con un valor diferente.
- **unary\_nonmatching\_cost\_sink = 1** - El costo del eje que une el destino y un pixel con un valor diferente.

### 4. Resultados

Los siguientes ejemplos ilustran las etapas de las imágenes utilizadas en el proceso con los siguientes parámetros:

- **width\_objective = 150**
- **noise\_percentage = .1**
- **pairwise\_cost = 1**
- **unary\_matching\_cost\_source = .9**
- **unary\_matching\_cost\_sink = .9**

- unary\_nonmatching\_cost\_source = 1
- unary\_nonmatching\_cost\_sink = 1

#### 4.1. Ejemplo 1

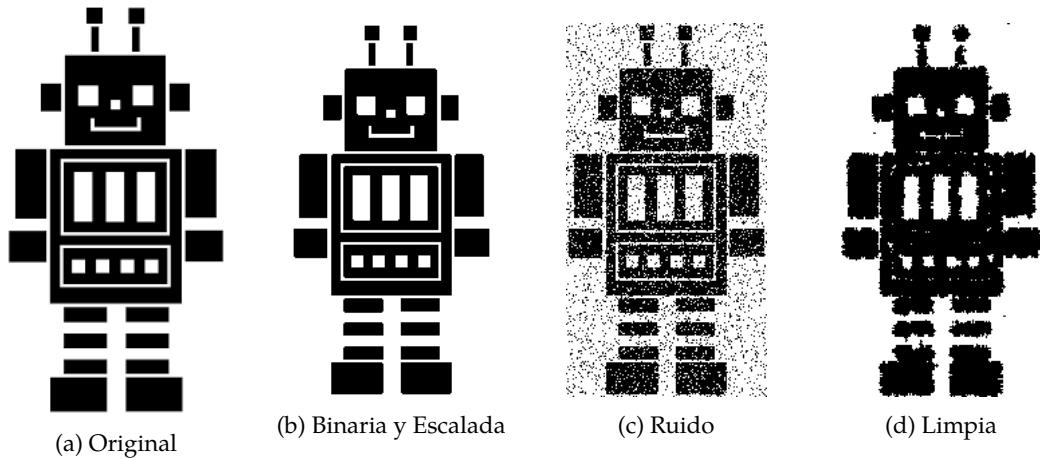


Figura 3: Ejemplo 1

#### 4.2. Ejemplo 2

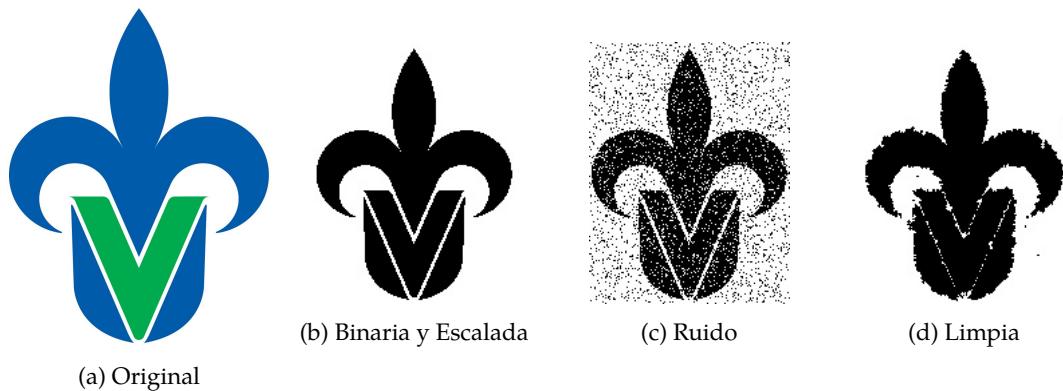


Figura 4: Ejemplo 2

#### 4.3. Ejemplo 3

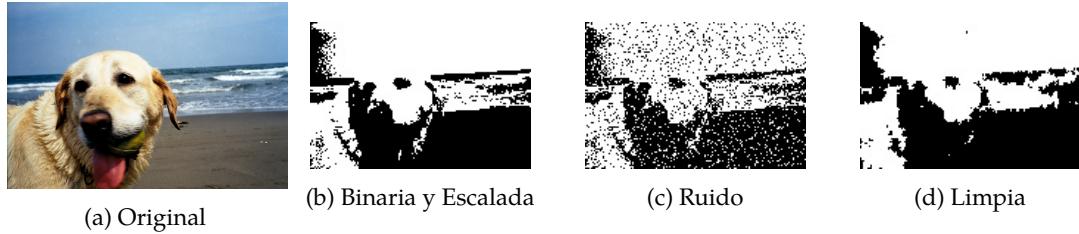


Figura 5: Ejemplo 3

#### 4.4. Ejemplo 4

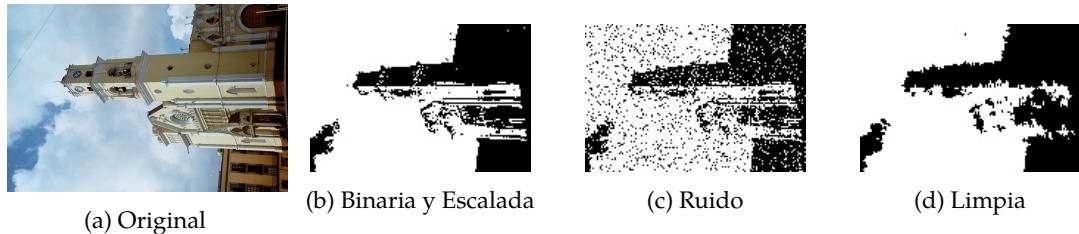


Figura 6: Ejemplo 4

#### 4.5. Ejemplo 4

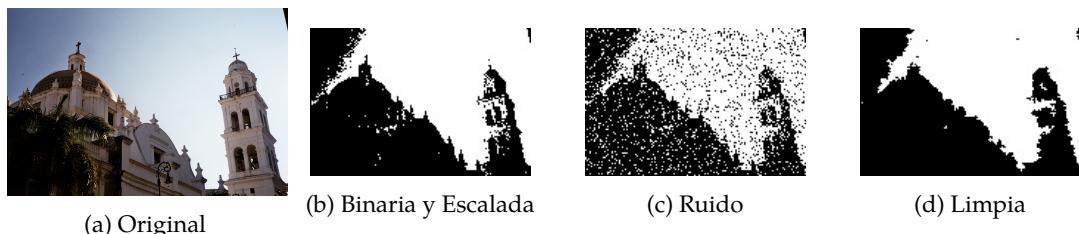


Figura 7: Ejemplo 4

## Referencias

- [Pri12] S.J.D. Prince. *Computer Vision: Models Learning and Inference*. Cambridge University Press, 2012.
- [Wik18a] Wikipedia. Campo aleatorio de Markov — Wikipedia, the free encyclopedia. <http://es.wikipedia.org/w/index.php?title=Campo%20aleatorio%20de%20Markov&oldid=112334356>, 2018. [En línea; Última visita: 11-Diciembre-2018].
- [Wik18b] Wikipedia. Maximum a posteriori estimation — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Maximum%20a%20posteriori%20estimation&oldid=864489517>, 2018. [En línea; Última visita: 11-Diciembre-2018].