

概率图模型

胡香江

一. 简介

1. 定义

概率图模型是一个结合了不确定性（概率论）和逻辑结构（图论）来表示真实世界复杂现象的模型。

很多模型（例如Kalman filters, hidden Markov models, Ising models）都可以归纳为概率图模型。

2. 分类

主要有两类：贝叶斯网络（Bayesian network）和马尔科夫网络（MRFs）

其中，贝叶斯网络又称directed graphical models，马尔科夫网络又称undirected graphical models。

3. 条件独立

其中表示概率图模型中最重要的概念是条件独立。

Definition 2.1

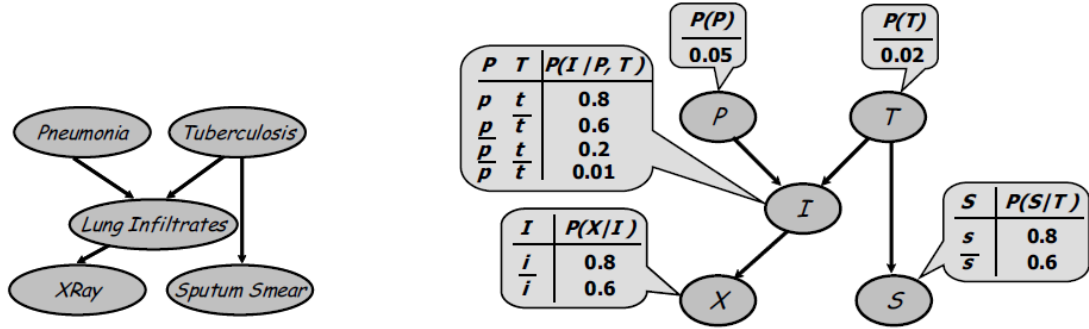
Let \mathbf{X} , \mathbf{Y} , and \mathbf{Z} be sets of random variables. \mathbf{X} is *conditionally independent* of \mathbf{Y} given \mathbf{Z} in a distribution P if

$$P(\mathbf{X} = \mathbf{x}, \mathbf{Y} = \mathbf{y} \mid \mathbf{Z} = \mathbf{z}) = P(\mathbf{X} = \mathbf{x} \mid \mathbf{Z} = \mathbf{z})P(\mathbf{Y} = \mathbf{y} \mid \mathbf{Z} = \mathbf{z})$$

for all values $\mathbf{x} \in \text{Val}(\mathbf{X})$, $\mathbf{y} \in \text{Val}(\mathbf{Y})$ and $\mathbf{z} \in \text{Val}(\mathbf{Z})$. ■

4. 贝叶斯网络部分

贝叶斯网络是基于一个有方向的无环图(DAG)来表示的，每个节点表示一个随机变量，箭头表示一个随机变量对另一个有影响。可以用下面的例子来说明这一关系：



那么，根据有向图中子节点母节点关系，我们可以定义图中所有随机变量联合密度分布的分解表达式：

Definition 2.2

Let \mathcal{G} be a Bayesian network graph over the variables X_1, \dots, X_n . We say that a distribution $P_{\mathcal{B}}$ over the same space *factorizes according to* \mathcal{G} if $P_{\mathcal{B}}$ can be expressed as a product

$$P_{\mathcal{B}}(X_1, \dots, X_n) = \prod_{i=1}^n P(X_i \mid \text{Pa}_{X_i}). \quad (2.1)$$

A *Bayesian network* is a pair $(\mathcal{G}, \theta_{\mathcal{G}})$ where $P_{\mathcal{B}}$ factorizes over \mathcal{G} , and where $P_{\mathcal{B}}$ is specified as set of CPDs associated with \mathcal{G} 's nodes, denoted $\theta_{\mathcal{G}}$. ■

这一规则可以成为贝叶斯网络的链规则，我们可以通过这一规则，随机变量分布可以计算成为一系列因子的乘积。

在贝叶斯网络中，另外值得关注的点是其中关于条件独立性的假设。从另一个角度来看，贝叶斯网络也可以看成一种在一系列条件独立假设下关于分布的表示方式。

首先，我们有局部Markov假设：

Definition 2.3

Given a BN network structure \mathcal{G} over random variables X_1, \dots, X_n , let $NonDescendants_{X_i}$ denote the variables in the graph that are not descendants of X_i . Then \mathcal{G} encodes the following set of conditional independence assumptions, called the *local Markov assumptions*:

For each variable X_i , we have that

$$(X_i \perp NonDescendants_{X_i} \mid \mathbf{Pa}_{X_i}),$$

In other words, the local Markov assumptions state that each node X_i is independent of its nondescendants given its parents. ■

也就是随机变量 \mathbf{x} 在给定其母节点分布的情况下，与其他非母节点条件独立。通过该假设，我们很容易分解联合密度分布，有以下两个定理：

Theorem 2.6

Let \mathcal{G} be a BN graph over a set of random variables \mathcal{X} and let P be a joint distribution over the same space. If all the local Markov properties associated with \mathcal{G} hold in P , then P factorizes according to \mathcal{G} .

Theorem 2.7

Let \mathcal{G} be a BN graph over a set of random variables \mathcal{X} and let P be a joint distribution over the same space. If P factorizes according to \mathcal{G} , then all the local Markov properties associated with \mathcal{G} hold in P .

5. 马尔科夫网络

马尔科夫网络通常也称为Markov Random Filed，是基于无向概率图的一类模型。当我们无法推断随机变量之间逻辑的方向性时，马尔科夫网络是一个非常有用的模型。马尔科夫网络节点同样表示随机变量，其边代表相邻的随机变量之间存在相互关系。

马尔科夫网络通常用因子的表示形式来参数化：

Definition 2.8

Let \mathbf{D} be a set of random variables. We define a *factor* to be a function from $Val(\mathbf{D})$ to \mathbb{R}^+ . ■

Definition 2.9

Let \mathcal{H} be a Markov network structure. A distribution $P_{\mathcal{H}}$ *factorizes* over \mathcal{H} if it is associated with

- a set of subsets D_1, \dots, D_m , where each D_i is a complete subgraph of \mathcal{H} ;
- factors $\pi_1[D_1], \dots, \pi_m[D_m]$,

such that

$$P_{\mathcal{H}}(X_1, \dots, X_n) = \frac{1}{Z} P'(X_1, \dots, X_n),$$

where

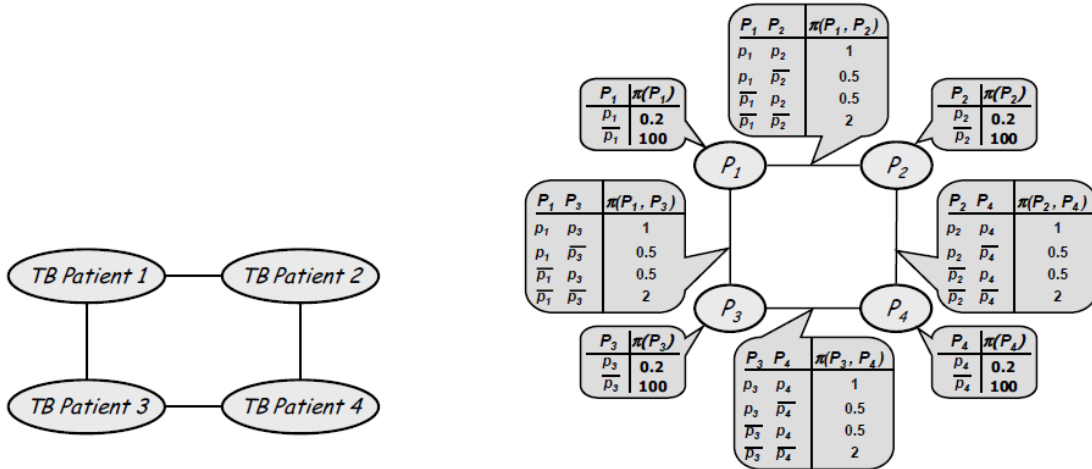
$$P'_{\mathcal{H}}(X_1, \dots, X_n) = \pi_1[D_1] \times \pi_2[D_2] \times \dots \times \pi_m[D_m]$$

is an unnormalized measure and

$$Z = \sum_{X_1, \dots, X_n} P'_{\mathcal{H}}(X_1, \dots, X_n)$$

is a normalizing constant called the *partition function*. A distribution P that factorizes over \mathcal{H} is also called a *Gibbs distribution* over \mathcal{H} . (The naming convention has roots in statistical physics.) ■

由于每个完整的子图是一些团簇的子集，我们可以简化参数化过程通过只给团簇引入因子值。这些因子又可以称为团势。可以用下面的例子来说明这一关系：



同样，在马尔科夫网络中，独立性非常重要。马尔科夫网络中的图结构可以看成一组独立性假设的合并。那么，我们有以下的独立性假定：

Definition 2.10

Let \mathcal{H} be an undirected graph. Then for each node $X \in \mathcal{X}$, the Markov blanket of X , denoted $\mathcal{N}_{\mathcal{H}}(X)$, is the set of neighbors of X in the graph (those that share an edge with X). We define the *local Markov independencies* associated with \mathcal{H} to be

$$\mathcal{I}_{\ell}(\mathcal{H}) = \{(X \perp \mathcal{X} - \{X\} - \mathcal{N}_{\mathcal{H}}(X) \mid \mathcal{N}_{\mathcal{H}}(X)) : X \in \mathcal{X}\}.$$

In other words, the Markov assumptions state that X is independent of the rest of the nodes in the graph given its immediate neighbors. ■

也就是 \mathbf{x} 在给定了他临接节点分布的条件下， \mathbf{x} 与其余节点独立。

其次，我们可以通过定义active paths

Definition 2.11

Let \mathcal{H} be a Markov network structure, and $X_1 \dots X_k$ be a path in \mathcal{H} . Let $\mathbf{E} \subseteq \mathcal{X}$ be a set of *observed variables*. The path $X_1 \dots X_k$ is *active* given \mathbf{E} if none of the X_i 's, $i = 1, \dots, k$, is in \mathbf{E} . ■

来进一步定义全局马尔科夫性：

Definition 2.12

We say that a set of nodes \mathbf{Z} *separates* \mathbf{X} and \mathbf{Y} in \mathcal{H} , denoted $sep_{\mathcal{H}}(\mathbf{X}; \mathbf{Y} \mid \mathbf{Z})$, if there is no active path between any node $X \in \mathbf{X}$ and $Y \in \mathbf{Y}$ given \mathbf{Z} . We define the *global Markov assumptions* associated with \mathcal{H} to be

$$\mathcal{I}(\mathcal{H}) = \{(\mathbf{X} \perp \mathbf{Y} \mid \mathbf{Z}) : sep_{\mathcal{H}}(\mathbf{X}; \mathbf{Y} \mid \mathbf{Z})\}. \blacksquare$$

与贝叶斯网络类似，我们同样有关于联合密度分布的两个分解定理：

Theorem 2.13

Let P be a distribution over \mathcal{X} , and \mathcal{H} a Markov network structure over \mathcal{X} . If P is a Gibbs distribution over \mathcal{H} , then all the local Markov properties associated with \mathcal{H} hold in P .

The other direction, which goes from the global independence properties of a distribution to its factorization, is known as the *Hammersley-Clifford theorem*. Unlike for Bayesian networks, this direction does not hold in general. It only holds under the additional assumption that P is a positive distribution.

Theorem 2.14

Let P be a positive distribution over \mathcal{X} , and \mathcal{H} a Markov network graph over \mathcal{X} . If all of the independence constraints implied by \mathcal{H} hold in P , then P is a Gibbs distribution over \mathcal{H} .

二. 参数估计与推断

1. 参数估计

贝叶斯网络：

估计贝叶斯网络的 CPD 表格中的数值很简单，就是计算训练数据中事件发生的次数。也就是说，如果要估计 $p(\mathbf{A}=\mathbf{s1} \mid \mathbf{B}=\mathbf{i1})$ ，我们只需要计算 $\mathbf{A}=\mathbf{s1}$ 且 $\mathbf{B}=\mathbf{i1}$ 的数据点在 $\mathbf{B}=\mathbf{i1}$ 的数据点总量中所占的比例。尽管这种方法看起来似乎是特定于这个问题的，但事实证明这样获得的参数能够最大化被观察到的数据的可能性。

马尔科夫网络：

上述计数方法对马尔可夫网络没有统计学上的支持（因此会得到次优的参数）。所以我们需要使用更加复杂的技术。这些技术背后的基本思想是梯度下降——我们定义一些描述其概率分布的参数，然后使用梯度下降来寻找能最大化被观察数据的可能性的参数值。

2. 推断简介

首先，推断是我们打造概率图模型框架的原因——要根据我们已知的信息去做出预测。但是推断在计算上十分困难，在某些特定类型图中我们可以相当高效地执行推断，但一般的图规模一旦太大就难以计算。所以我们需要在近似算法与精确推断之间对于具体问题权衡。

3.推断应用

在实际应用中，我们可以使用推理来解答一些问题：

（1）边际推理（marginal inference）：寻找一个特定变量的概率分布。比如，给定一个带有变量 A、B、C 和 D 的图，其中 A 取值 1、2 和 3，求 $p(A=1)$ 、 $p(A=2)$ 和 $p(A=3)$ 。

（2）后验推理（posterior inference）：给定某些显变量 v_E （E 表示证据（evidence）），其取值为 e，求某些隐藏变量 v_H 的后验分布 $p(v_H|v_E=e)$ 。

（3）最大后验（MAP）推理（maximum-a-posteriori inference）：给定某些显变量 v_E ，其取值为 e，求使其它变量 v_H 有最高概率的配置

解答这些问题本身可能就很有用，也可能被用作更大规模的任务的一部分。

4.推断中常见算法

（1）变量消除（Variable elimination）

使用条件概率的定义，我们可以将后验分布写作：

$$p(v_H|v_E = e) = \frac{p(v_H, v_E = e)}{p(v_E = e)}$$

我们可以怎样计算上式中的分子和分母呢？让我们用一个简单的例子进行说明。考虑一个有三个变量的网络，其联合分布定义如下：

A	B	C	p(A, B, C)
0	0	0	0.05
0	0	1	0.01
0	1	0	0.10
0	1	1	0.05
1	0	0	0.30
1	0	1	0.09
1	1	0	0.25
1	1	1	0.15

假设我们目标是计算 $p(A|B=1)$ 。注意这意味着我们需要计算 $p(A=0|B=1)$ 和 $p(A=1|B=1)$ ，这两个值的和应该为 1。使用上面的等式，我们可以写：

$$p(A = 0|B = 1) = \frac{p(A = 0, B = 1)}{p(B = 1)}$$

分子是 $A=0$ 且 $B=1$ 的概率。我们不关心 C 的值。所以我们会把 C 的所有值都加起来。（这是由于基本概率 $p(A=0, B=1, C=0)$ 和 $p(A=0, B=1, C=1)$ 是互斥事件，所以它们的联合概率 $p(A=0, B=1)$ 就是各个概率的总和。）

所以我们将第 3 和 4 行加起来得到 $p(A=0, B=1)=0.15$ 。类似地，将第 7 和 8 行加起来得到 $p(A=1, B=1)=0.40$ 。另外，我们可以求所有包含 $B=1$ 的行的总和来计算分母，即第 3、4、7、8 行，从而得到 $p(B=1)=0.55$ 。从而我们可以得到：

$$p(A=0|B=1) = 0.15 / 0.55 = 0.27$$

$$p(A=1|B=1) = 0.40 / 0.55 = 0.73$$

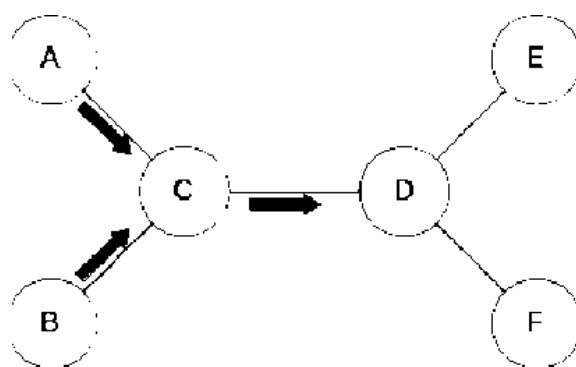
在上述计算中，我们做了一些重复的工作：将 3 和 4 行以及 7 和 8 行加了两次。计算

$p(B=1)$ 的更高效方法是直接将 $p(A=0, B=1)$ 和 $p(A=1, B=1)$ 的值加起来。这是变量消除的基本思想。一般来说, 当有很多变量时, 你不仅可以使用的分子的值来计算分母, 而且分子本身也可能会包含重复的计算。你可以使用动态编程来高效地使用之前已计算出的值。因为我们一次对一个变量进行求和, 从而可以消除这个变量, 所以对多个变量进行求和的过程相当于逐个消除这些变量。所以我们将这个过程称为「变量消除」。

我们也可以直接地将上述过程用于求解边缘推理或 MAP 推理问题。类似地, 也可以容易地将上述思想推广应用于马尔可夫网络。但是值得注意的是, 变量消除的时间复杂度取决于图结构以及你消除这些变量的顺序。在最糟糕的情况下, 时间复杂度会指数式增长。

(2) 置信度传播 (Belief Propagation)

我们刚才看到的变量消除算法只会得到一个最终分布。假设我们想找到所有变量的边缘分布。除了多次运行变量消除之外, 我们还有更聪明的方法。假设我们已经有一个图结构了。为了计算某个边缘, 我们需要对其在其它所有变量上的联合分布进行求和, 这相当于将整个图的信息聚合到一起。还有另一种聚合整个图的信息的方法——每个节点都检查其邻近节点, 然后以局部的方式近似变量的分布。然后, 每一对相邻节点都互相发送“消息”, 这些消息中包含了其局部分布。现在, 每个节点都检查其收到的消息, 然后将它们聚合起来以更新变量的概率分布。



在上图中, C 聚合了来自邻近节点 A 和 B 的信息, 然后再发送一个消息给 D。然后 D 将这个信息与来自 E 和 F 的信息聚合起来。这种方法的优点是如果你保存了你在每个节点处发送的消息, 那么将这些消息进行一次前向通过, 然后再进行一次反向通过, 就能让所有节点都得到所有其它节点的信息。然后这个信息可以被用于计算所有的边缘, 这是无法使用变量消除实现的。

(3) 近似推理

对于大型的图模型来说, 进行精准的推理可能极其耗时, 为此很多用于图模型的近似推理算法被开发了出来, 其中大多数都属于下面两类:

(3.1) 基于采样的近似推理

这些算法使用采样来估计希望得到的概率。对于概率图模型领域内的更复杂的算法, 你也可以使用类似的流程。基于采样的算法还可以进一步分为两类。一类中的样本是相互独立的, 比如上面抛硬币的例子。这些算法被称为蒙特卡洛方法。对于有很多变量的问题, 生成高质量的独立样本是很困难的, 因此我们就生成带有依赖关系的样本, 也就是说每个新样本都是随机的, 但邻近上一个样本。这种算法被称为马尔可夫链蒙特卡洛 (MCMC) 方法, 因为这些样本会形成一个马尔可夫链 (Markov chain)。一旦我们得到了样本, 我们就可以将其用于解答各种推理问题。

(3.2) 变分法近似推理

变分法近似推理不是使用采样, 而是试图通过分析的方式来近似所需的分布。假设你写

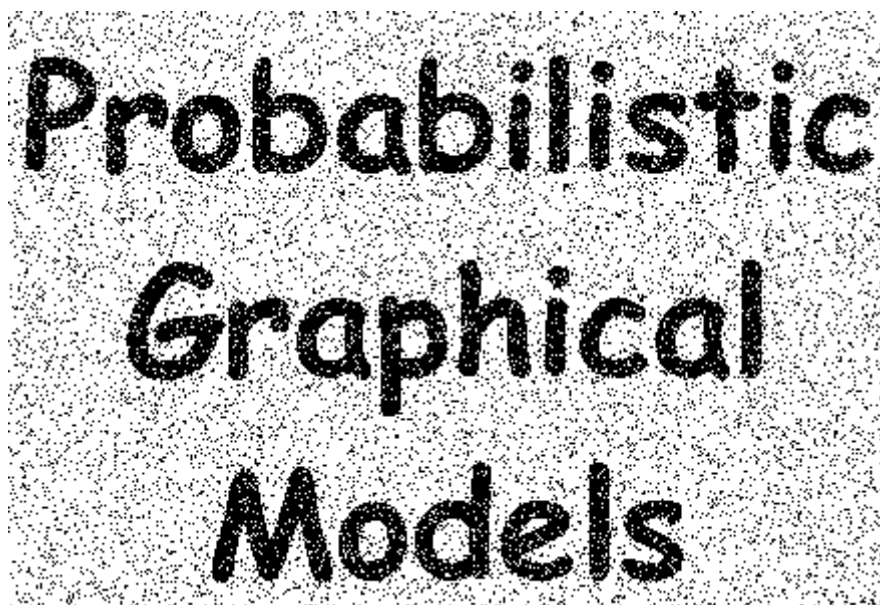
出了计算相关分布的表达式——不管这个分布是边际概率分布还是后验概率分布。通常这些表达式里面有求和或积分，要精确评估是极其消耗计算资源的。要近似这些表达式，一种好方法是求解一个替代表达式，并且通过某种方法使该替代表达式接近原来的表达式。这就是变分法背后的基本思想。具体来说，当我们试图估计一个复杂的概率分布 p_{complex} 时，我们定义另一组更易于操作的概率分布 P_{simple} ，然后基于 P_{simple} 得到最接近于 p_{complex} 的概率分布 p_{approx} 。

三. 概率图模型的应用——以图像消噪为例

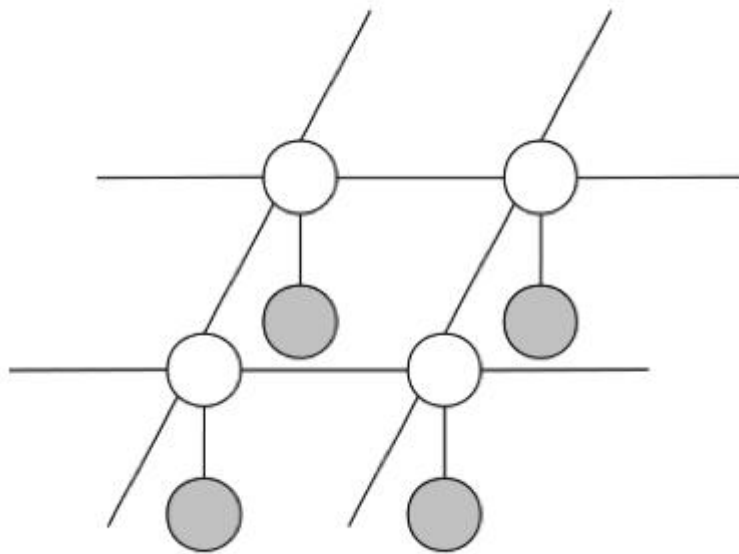
以下图作为原图：

Probabilistic Graphical Models

现在在图形中加入随机噪声（噪声大小为 10.060048647%）：

The image shows the same text 'Probabilistic Graphical Models' as the previous block, but it is heavily corrupted with random black and white noise, making it difficult to read. The noise is distributed across the entire image area, obscuring the original text.

现在目标是恢复原图，建立以下的图结构：



其中，白色节点表示隐变量 Y_{ij} ，灰色节点表示观测变量 x_{ij} ，每个隐变量与周围四个隐变量存在关系（临近的像素通常有一样的值），每个隐变量与其观测值存在关系（90%的像素真实值与观测值相同）。

因为图像的像素间并不存在因果关系，所以采用无方向的网络（贝叶斯网络不适合）

最大后验（MAP）推理问题可以写成

$$\begin{aligned}
 Y^* &= \arg \max_Y P(Y|X) \\
 &= \arg \max_Y \log P(Y|X) \\
 &= \arg \max_Y [\log P(X, Y) - \log P(X)] \\
 &= \arg \max_Y \log P(X, Y)
 \end{aligned}$$

联合分布函数可以写成：

$$p(X, Y) = \frac{1}{Z} \prod_{ij} \phi(X_{ij}, Y_{ij}) \prod_{(ij, kl)} \phi(Y_{ij}, Y_{kl})$$

将分布函数带入 MAP 推理式中可以得到：

$$Y^* = \arg \max_Y \log p(X, Y) = \arg \max_Y \sum_{ij} w_e X_{ij} Y_{ij} + \sum_{(ij,kl)} w_s Y_{ij} Y_{kl}$$

其中参数 w_e 和 w_s 可以通过梯度下降估计得到

我们这里为了简化推理的难度，采用了蒙特卡洛的思路来近似推理：在每个步骤选择一个节点 Y_{ij} ，然后检查 $Y_{ij}=-1$ 和 $Y_{ij}=1$ 时 MAP 推理表达式的值，然后选择值更高的那个。

重复这个过程一定的迭代次数或直到收敛，通常就能得到相当好的结果。

通过该算法去噪后图像：

Probabilistic Graphical Models

去噪后误差仅有 0.664335664336%，结果收敛，为局部最优。

具体例子实现代码见下：（Python）

```
from __future__ import print_function
import numpy as np
from PIL import Image
def compute_log_prob_helper(Y, i, j):
    try:
        return Y[i][j]
    except IndexError:
        return 0
def compute_log_prob(X, Y, i, j, w_e, w_s, y_val):
    result = w_e * X[i][j] * y_val
    result += w_s * y_val * compute_log_prob_helper(Y, i-1, j)
    result += w_s * y_val * compute_log_prob_helper(Y, i+1, j)
    result += w_s * y_val * compute_log_prob_helper(Y, i, j-1)
    result += w_s * y_val * compute_log_prob_helper(Y, i, j+1)
    return result
def denoise_image(X, w_e, w_s):
    m, n = np.shape(X)
```

```

# initialize Y same as X
Y = np.copy(X)
# optimization
max_iter = 10*m*n
for iter in range(max_iter):
    # randomly pick a location
    i = np.random.randint(m)
    j = np.random.randint(n)
    # compute the log probabilities of both values of Y_ij
    log_p_neg = compute_log_prob(X, Y, i, j, w_e, w_s, -1)
    log_p_pos = compute_log_prob(X, Y, i, j, w_e, w_s, 1)
    # assign Y_ij to the value with higher log probability
    if log_p_neg > log_p_pos:
        Y[i][j] = -1
    else:
        Y[i][j] = 1
    if iter % 100000 == 0:
        print ('Completed', iter, 'iterations out of', max_iter)
return Y

def read_image_and_binarize(image_file):
    im = Image.open(image_file).convert("L")
    A = np.asarray(im).astype(int)
    A.flags.writeable = True
    A[A<128] = -1
    A[A>=128] = 1
    return A

def add_noise(orig):
    A = np.copy(orig)
    for i in range(np.size(A, 0)):
        for j in range(np.size(A, 1)):
            r = np.random.rand()
            if r < 0.1:
                A[i][j] = -A[i][j]
    return A

def convert_from_matrix_and_save(M, filename, display=False):
    M[M==-1] = 0
    M[M==1] = 255
    im = Image.fromarray(np.uint8(M))
    if display:
        im.show()
    im.save(filename)

def get_mismatched_percentage(orig_image, denoised_image):
    diff = abs(orig_image - denoised_image) / 2
    return (100.0 * np.sum(diff)) / np.size(orig_image)

```

```
orig_image=read_image_and_binarize(r'C:\Users\xiangjiang.hu\Desktop\12321.png')
noisy_image = add_noise(orig_image)
denoised_image = denoise_image(noisy_image, 15, 10)
print ('Percentage of mismatched pixels: ', get_mismatched_percentage(orig_image,
denoised_image),'%',sep='')
print ('Percentage of mismatched pixels: ', get_mismatched_percentage(orig_image,
noisy_image),'%',sep='')
convert_from_matrix_and_save(orig_image, 'orig_image.png', display=True)
convert_from_matrix_and_save(noisy_image, 'noisy_image.png', display=True)
convert_from_matrix_and_save(denoised_image,
r'C:\Users\xiangjiang.hu\Desktop\denoised_image.png', display=True)
```