ICA: Independent Component Analysis

December 11, 2015

1 前言

独立成分分析ICA是一个在多领域被应用的基础算法。ICA是一个不定问题，没有确定解，所以存在各种不同先验假定下的求解算法。相比其他技术，ICA的开源代码不是很多，且存在黑魔法–有些步骤并没有在论文里提到，但没有这些步骤是无法得到正确结果的。

本文给出一个ICA最大似然解法的推导，以及FastICA的python实现，限于时间和实际需求，没有对黑魔法部分完全解读，只保证FastICA实现能得到正确结果。

有兴趣的童鞋可以在未来补上相关内容。

2 ICA问题表述

设X是随机向量，且$X \in R^{n\times 1}$，这也就说，X里有n个成员，每个成员是一个随机变量：

$$X = \begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \cdots \\ \boldsymbol{x}_i \\ \cdots \\ \boldsymbol{x}_n \end{pmatrix} \tag{1}$$

其中，$\boldsymbol{x}_i$是一个随机变量。

随机变量有诸多特性，殆由概率论和数理统计教科书详述备尽，在此不一一叙述。

X里的n个随机变量是相互非独立的，在一定的假设下，可以用n个相互独立的随机变量线性组合重新表达X，也就是说：

$$\begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \cdots \\ \boldsymbol{x}_i \\ \cdots \\ \boldsymbol{x}_n \end{pmatrix} = A \begin{pmatrix} \boldsymbol{s}_1 \\ \boldsymbol{s}_2 \\ \cdots \\ \boldsymbol{s}_i \\ \cdots \\ \boldsymbol{s}_n \end{pmatrix} \tag{2}$$

其中，$\boldsymbol{s}_i$是一个随机变量，且两两相互独立，A是满秩矩阵，且$A \in R^{n \times n}$。
令:

$$S = \begin{pmatrix} \boldsymbol{s}_1 \\ \boldsymbol{s}_2 \\ ... \\ \boldsymbol{s}_i \\ ... \\ \boldsymbol{s}_n \end{pmatrix} \tag{3}$$

则：

$$X = AS \tag{4}$$

又有:

$$S = A^{-1}X \tag{5}$$

令：

$$W = A^{-1} \tag{6}$$

则:

$$S = WX \tag{7}$$

其中，$W \in R^{n \times n}$。

记录随机向量X的值m次，则形成数据集：

$$D = \begin{pmatrix} d_{1,1} & d_{1,2} & ... & d_{1,m} \\ d_{2,1} & d_{2,2} & ... & d_{2,m} \\ ... & ... & ... & ... \\ d_{n,1} & d_{n,2} & ... & d_{n,m} \end{pmatrix} \tag{8}$$

其中，$D \in R^{n \times m}$

ICA的目标，就是在只知道D的情况下，估算A，W，S的值。

实例：在一个大厅里，有n个人在随机聊天。在大厅的不同角落，布置n个麦克风记录大厅的声音，每秒一个记录，一共记录m秒。麦克风记录的混合声音，多个麦克风记录不同位置的混合声音。ICA的目标，就是从混声录音中将每个人的声音分离出来。

3 理论推导

由式(7)可知:

$$\boldsymbol{s}_i = \begin{pmatrix} w_{i,1} & w_{i,2} & ... & w_{i,j} & ... & w_{i,n} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ ... \\ \boldsymbol{x}_i \\ ... \\ \boldsymbol{x}_n \end{pmatrix} \tag{9}$$

令:

$$w_i = \begin{pmatrix} w_{i,1} & w_{i,2} & ... & w_{i,j} & ... & w_{i,n} \end{pmatrix} \tag{10}$$

则:

$$\boldsymbol{s}_i = w_i X \tag{11}$$

设随机变量$\boldsymbol{s}_i$概率密度函数是$p_{s_i}(s_i)$，其中p的右下角s_i表示随机变量标示，括号中的s_i表示自变量。

由于S的n个成员$\boldsymbol{s}_i$是相互独立的，所以S的概率密度函数为:

$$p_S(s) = \prod_{i=1}^{n} p_{s_i}(s_i) \tag{12}$$

设X的概率密度函数是$p_X(x)$，如何根据$\boldsymbol{s}_i$的概率密度函数求$p_X(x)$呢?这是可以做到的。

设随机向量X的概率分布函数是$F_X(x)$，根据概率分布函数和概率密度函数的关系可知:

$$p_X(x) = F_X^{'}(x) \tag{13}$$

同理，设随机向量S的概率分布函数是$F_S(s)$，则:

$$p_S(s) = F_S^{'}(s) \tag{14}$$

根据概率分布函数的定义，有:

$$F_X(x) = P(X < x) \tag{15}$$

$$F_S(s) = P(S < s) \tag{16}$$

那么:

$$\begin{aligned} p_X(x) &= F_X^{'}(x) \\ &= \left(P(X < x)\right)^{'} \\ &= \left(P(U < u)\right)^{'} \\ &= \left(P(U < s)\right)^{'} \\ &= \left(\|P(U < Wx)\|\right)^{'} \\ &= \left(\|P(S < Wx)\|\right)^{'} \\ &= \left(\|F_S(Wx)\|\right)^{'} \\ &= \|F_S^{'}(Wx)\| \\ &= \|p_S(Wx)(Wx)^{'}\| \\ &= \|p_S(Wx)W\| \\ &= p_S(Wx)\|W\| \\ &= \|W\| \prod_{i=1}^{n} p_{s_i}(w_i x) \end{aligned} \tag{17}$$

其中，上式的第2个等号是概率密度函数的定义，第3个等号是做变量等价代换，以免直接从X变换到S导致思维混乱，第4个等号到第6个等号是逐步将X代换到S，第7个等号是回到S的概率分布函数定义，第8个等号到第10个等号是求导。

从第5个等号开始，对整个等式取行列式运算，因为$p_X(x)$一定是标量，对标量做行列式运算是它自身。那么，到了第10个等号，又因为$p_S(WX)$一定是标量，所以可以从行列式运算拿到外面。这里避免的问题的是，如果不对整个等式取行列式，得到的结果是矩阵W而不是$\|W\|$，这是没有道理的。

注意，在上式中，x是一个向量，且$x \in R^{n\times 1}$，$w_i \in R^{1\times n}$，$p_{s_i}(s_i)$是一个单自变量的函数，$p_X(x)$是一个多自变量函数，它的自变量是x里的多个变量，这样等式左右的每一步就清晰了。

下一步是根据数据集计算W的值，从概率的角度来说，如果数据集已经记录，那么让这个数据集出现概率最大的W就是最优值。

数据集也就是式(8)出现的概率是:

$$L = \prod_{i=1}^{m}(\|W\| \prod_{j=1}^{n} p_{s_j}(w_j d_i)) \tag{18}$$

其中，$\prod$表示连乘，d_i是D的第i列，也就是:

$$d_i = \begin{pmatrix} d_{i,1} \\ d_{i,2} \\ ... \\ d_{i,n} \end{pmatrix} \tag{19}$$

d_i的物理意义，也就是第i次记录随机向量X得到的n个值，这n个值分别对应n个$\boldsymbol{x}_i$随机变量。注意，不要把d_i和$\boldsymbol{x}_i$混淆，前者表示D的一列数据，后者是粗体表示一个随机变量。

式(18)有最大值，当它取最大值时候的W就是最优解。如果以梯度下降法求解，需要计算它对W的偏导，直接求偏导比较复杂，故对它两端取自然对数，则:

$$\begin{aligned} lnL &= \sum_{i=1}^{m}(ln\|W\| + \sum_{j=1}^{n}(lnp_{s_j}(w_j d_i))) \\ &= \sum_{i=1}^{m}\sum_{j=1}^{n} lnp_{s_j}(w_j d_i) + mln\|W\| \end{aligned} \tag{20}$$

当上式取最大值的时候，L也同时取最大值，所以求L的最大值等价于求上式的最大值。

用梯度下降法求解上式，需要计算$\frac{\partial lnL}{\partial W}$。这是一个复杂的过程，先从计

算$\frac{\partial L}{\partial w_{u,v}}$开始，它表示$W$的第$u$行第$v$列的一个成员：

$$\begin{aligned}\frac{\partial lnL}{\partial w_{u,v}} &= \sum_{i=1}^{m}\sum_{j=1}^{n}\frac{1}{p_{s_j}(w_j d_i)}\frac{\partial p_{s_j}(w_j d_i)}{\partial w_{u,v}} + \frac{m}{\|W\|}\frac{\partial \|W\|}{\partial w_{u,v}} \\ &= \sum_{i=1}^{m}\sum_{j=1}^{n}\frac{1}{p_{s_j}(w_j d_i)}\frac{\partial p_{s_j}(w_j d_i)}{\partial w_{u,v}} + \frac{m}{\|W\|}(-1)^{u+v}M_{uv} \\ &= \sum_{i=1}^{m}\frac{1}{p_{s_u}(w_u d_i)}\frac{\partial p_{s_u}(w_u d_i)}{\partial w_{u,v}} + \frac{m}{\|W\|}(-1)^{u+v}M_{uv}\end{aligned} \tag{21}$$

其中，$(-1)^{u+v}M_{uv}$是$w_{u,v}$的代数余子式，$\frac{\partial p_{s_u}(w_u x_i)}{\partial w_{u,v}}$的值要根据$p_{s_i}(s_i)$的具体形式求解。

对于$p_{s_i}(s_i)$，如果在没有任何先验信息的情况下，是无法求解的。如果要求解上式，需要对它做一定的假设，在合理的假设下，可以达到相当不错的近似结果。

设随机变量$\boldsymbol{x}_i$的概率分布函数是sigmoid函数，因为它是递增，可微，且最大值不超过1，也就是说：

$$F_{s_i}(s_i) = \frac{1}{1+e^{-s_i}} \tag{22}$$

那么，概率密度函数就是:

$$p_{s_i}(s_i) = F'_{s_i}(s_i) = \frac{e^{s_i}}{(1+e^{s_i})^2} \tag{23}$$

所以有：

$$p_{s_u}(w_u d_i) = \frac{e^{w_u d_i}}{(1+e^{w_u d_i})^2} = e^{w_u d_i}(1+e^{w_u d_i})^{-2} \tag{24}$$

故:

$$\begin{aligned}\frac{\partial p_{s_u}(w_u d_i)}{\partial w_{u,v}} &= e^{w_u d_i} d_{i,v}(1+e^{w_u d_i})^{-2} - 2e^{w_u d_i}(1+e^{w_u d_i})^{-3}e^{w_u d_i}d_{i,v} \\ &= \frac{d_{i,v}e^{w_u d_i}}{(1+e^{w_u d_i})^2}(1 - 2\frac{e^{w_u d_i}}{1+e^{w_u d_i}}) \\ &= d_{i,v}p_{s_u}(w_u d_i)\frac{1-e^{w_u d_i}}{1+e^{w_u d_i}}\end{aligned} \tag{25}$$

其中，$d_{i,v}$是d_i的第v行的一个成员。

因此:

$$\begin{aligned}\frac{\partial lnL}{\partial w_{u,v}} &= \sum_{i=1}^{m}\frac{1}{p_{s_u}(w_u d_i)}\frac{\partial p_{s_u}(w_u d_i)}{\partial w_{u,v}} + \frac{m}{\|W\|}(-1)^{u+v}M_{uv} \\ &= \sum_{i=1}^{m}\frac{1}{p_{s_u}(w_u d_i)}d_{i,v}p_{s_u}(w_u d_i)\frac{1-e^{w_u d_i}}{1+e^{w_u d_i}} + \frac{m}{\|W\|}(-1)^{u+v}M_{uv} \\ &= \sum_{i=1}^{m} d_{i,v}\frac{1-e^{w_u d_i}}{1+e^{w_u d_i}} + \frac{m}{\|W\|}(-1)^{u+v}M_{uv}\end{aligned} \tag{26}$$

现在对上式进行矩阵化，令：

$$K = WD \tag{27}$$

其中，$K \in R^{n\times m}$，$W \in R^{n\times n}$，$D \in R^{n\times m}$，那么，$k_{u,i}$就是K的第u行的第i列的一个成员，令：

$$g(x) = \frac{1-e^x}{1+e^x} \tag{28}$$

令:

$$Z = g(K) = \begin{pmatrix} g(k_{1,1}) & g(k_{1,2}) & ... & g(k_{1,m}) \\ g(k_{2,1}) & g(k_{2,2}) & ... & g(k_{2,m}) \\ ... & & & \\ g(k_{n,1}) & g(k_{n,2}) & ... & g(k_{n,m}) \end{pmatrix} \tag{29}$$

那么，就得到:

$$\frac{\partial lnL}{\partial w_{u,v}} = z_u^T d_v + \frac{m}{\|W\|}(-1)^{u+v} M_{uv} \tag{30}$$

其中，z_u是Z的第u行，d_v是D的第v列。

于是，对W而言，则有：

$$\frac{\partial lnL}{\partial W} = Z^T D + \frac{m}{\|W\|}(W^*)^T \tag{31}$$

其中，W^*是W的伴随矩阵，$(W^*)^T$是W^*的转置，它的第i行第j列的元素是$w_{i,j}$的代数余子式，也就是$(-1)^{i+j}M_{i,j}$。

根据矩阵和它的伴随阵的性质可知:

$$WW^* = \|W\|I \tag{32}$$

其中，I是单位矩阵。根据上两式可知:

$$\begin{aligned} \frac{\partial lnL}{\partial W} &= Z^T D + \frac{m}{\|W\|}(W^*)^T \\ &= Z^T D + \frac{m}{\|W\|}(\|W\|W^{-1})^T \\ &= Z^T D + m(W^{-1})^T \end{aligned} \tag{33}$$

那么，在梯度下降法求解W的时候，更新公式是:

$$W = W + \alpha(Z^T D + m(W^{-1})^T) \tag{34}$$

其中，α是学习速率。

最后的结论简洁且美，Verweile doch, du bist so schön。然并卵，按照这个结果实现代码，计算结果是不合理的，无法恢复原始信号。于是，在实现FastICA之后，可以认为本推导缺少一些黑魔法，至于到底缺少什么并不知道，限于时间关系和实际需求，不再继续研究下去。

4 FastICA

FastICA计算性能更好。《Indepdent Componet analysis》一书在第8章给出了FastICA的算法流程，如下:

1. Center the data to make its mean zero.
2. Whiten the data to give $\mathbf{z}$.
3. Choose m, the number of independent components to estimate.
4. Choose initial values for the $\mathbf{w}_i, i = 1, ..., m$, each of unit norm. Orthogonalize the matrix $\mathbf{W}$ as in step 6 below.
5. For every $i = 1, ..., m$, let $\mathbf{w}_i \leftarrow E\{\mathbf{z}g(\mathbf{w}_i^T\mathbf{z})\} - E\{g'(\mathbf{w}_i^T\mathbf{z})\}\mathbf{w}$, where g is defined, e.g., as in (8.31)–(8.33).
6. Do a symmetric orthogonalization of the matrix $\mathbf{W} = (\mathbf{w}_1, ..., \mathbf{w}_m)^T$ by
$$\mathbf{W} \leftarrow (\mathbf{W}\mathbf{W}^T)^{-1/2}\mathbf{W}, \tag{8.51}$$
or by the iterative algorithm in Sec. 8.4.3.
7. If not converged, go back to step 5.

Table 8.4 The FastICA algorithm for estimating several ICs, with *symmetric* orthogonalization. The expectations are estimated in practice as sample averages.

Figure 1: FastICA

5 白化

FastICA需要对数据做白化处理。设x是一个随机变量，存在一个线性变换V将它变换成z:

$$z = Vx \tag{35}$$

且:

$$E\{zz^T\} = I \tag{36}$$

那么，V就是白化变换矩阵。

x的协方差阵是$C_x = E\{xx^T\}$，$C_x = PDP^T$，P是C_x的单位特征向量，D是C_x的特征值组成的对角阵。那么，V的值就是:

$$V = D^{-\frac{1}{2}}P^T \tag{37}$$

证明如下：根据相关性质，有$P^T = P^{-1}$，由于D对角阵，则$(D^{-\frac{1}{2}})^T = D^{-\frac{1}{2}}$，那么：

$$\begin{aligned} E\{Vx(Vx)^T\} &= E\{Vxx^TV^T\} \\ &= E\{VPDP^TV^T\} \\ &= E\{VPDP^TV^T\} \\ &= E\{D^{-\frac{1}{2}}P^TPDP^TP(D^{-\frac{1}{2}})^T\} \\ &= E\{D^{-\frac{1}{2}}DD^{-\frac{1}{2}}\} \\ &= E\{I\} \\ &= I \end{aligned} \tag{38}$$

6 代码实现

基于python2.7，matplotlib，numpy实现ICA，主要参考sklean的FastICA实现。

```
#!/usr/bin/env python

#FastICA from ICA book, table 8.4

import math
import random
import matplotlib.pyplot as plt
from numpy import *

n_components = 2

def f1(x, period = 4):
    return 0.5*(x-math.floor(x/period)*period)

def create_data():
    #data number
    n = 500
    #data time
    T = [0.1*xi for xi in range(0, n)]
    #source
    S = array([[sin(xi)  for xi in T], [f1(xi) for xi in T]],
                                    float32)
    #mix matrix
    A = array([[0.8, 0.2], [-0.3, -0.7]], float32)
    return T, S, dot(A, S)
```

```
def whiten(X):
    #zero mean
    X_mean = X.mean(axis=-1)
    X -= X_mean[:, newaxis]
    #whiten
    A = dot(X, X.transpose())
    D , E = linalg.eig(A)
    D2 = linalg.inv(array([[D[0], 0.0], [0.0, D[1]]], float32))
    D2[0,0] = sqrt(D2[0,0]); D2[1,1] = sqrt(D2[1,1])
    V = dot(D2, E.transpose())
    return dot(V, X), V

def _logcosh(x, fun_args=None, alpha = 1):
    gx = tanh(alpha * x, x); g_x = gx ** 2
    g_x -= 1.; g_x *= -alpha
    return gx, g_x.mean(axis=-1)

def do_decorrelation(W):
    #black magic
    s, u = linalg.eigh(dot(W, W.T))
    return dot(dot(u * (1. / sqrt(s)), u.T), W)

def do_fastica(X):
    n, m = X.shape; p = float(m); g = _logcosh
    #black magic
    X *= sqrt(X.shape[1])
    #create w
    W = ones((n,n), float32)
    for i in range(n):
        for j in range(i):
            W[i,j] = random.random()

    #compute W
    maxIter = 200
    for ii in range(maxIter):
        gwtx, g_wtx = g(dot(W, X))
        W1 = do_decorrelation(dot(gwtx, X.T) / p - g_wtx[:,
                                              newaxis] * W)
        lim = max( abs(abs(diag(dot(W1, W.T))) - 1) )
        W = W1
        if lim < 0.0001:
            break
    return W
```

```
def show_data(T, S):
    plt.plot(T, [S[0,i] for i in range(S.shape[1])], marker="*"
                                        )
    plt.plot(T, [S[1,i] for i in range(S.shape[1])], marker="o"
                                        )
    plt.show()

def main():
    T, S, D = create_data()
    Dwhiten, K = whiten(D)
    W = do_fastica(Dwhiten)
    #Sr: reconstructed source
    Sr = dot(dot(W, K), D)
    show_data(T, D)
    show_data(T, S)
    show_data(T, Sr)

if __name__ == "__main__":
    main()
```

在这个实现中，创建了两个数据源，一个是正弦函数，一个是线性周期函数，它们的图形是Figure 2。

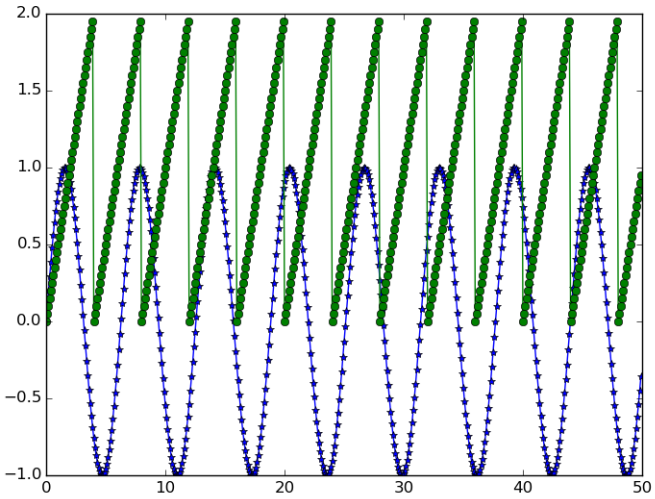

Figure 2: Sources

将这两个数据源混合成两个新数据源，也就是“可观测”的数据，它们的图像是Figure 3。

经过FastICA处理后，重建数据源Figure 4。注意，此时的数据源在图形形状上跟初始数据源具有相似性，但幅度是不一样的，且可能会发生翻转，这是因为ICA是一个不定问题，有多个解符合假设，不是唯一解。

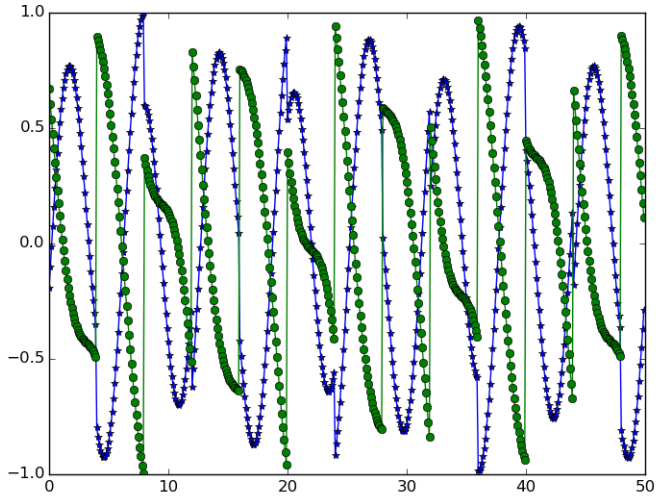

Figure 3: X

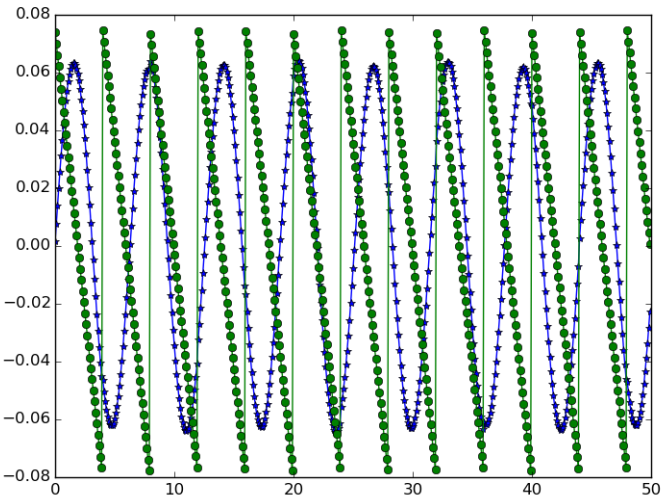

Figure 4: Reconstructed Sources