Paweł GÓRSKI

West Pomeranian University of Technology, Faculty of Computer Science and Information Technology

doi:10.15199/48.2015.02.19

Performance comparison of ICA algorithms for audio blind source separation

***Abstract**. The aim of this paper is to compare five algorithms for Independent Component Analysis. The algorithms are compared with regard to performance for separating three and seven input signals. It also examined how time and number of independent components affect on separation precision. Professional sound recordings and their mixes were used for all tests.*

Streszczenie.** W artykule porównano pięć popularnych algorytmów z rodziny analizy składowych niezależnych. Algorytmy porównywane były pod kątem wydajności dla trzech oraz siedmiu sygnałów wejściowych. Badano również jak czas działania algorytmu oraz zwiększenie liczby składowych wejściowych wpływa na dokładność separacji. Do testów zastosowano profesjonalnie nagrane próbki śpiewu oraz ich mieszanki. **(Porównanie wydajności algorytmów ICA w ślepej separacji sygnałów dźwiękowych).

Keywords: Independent Component Analysis, FastICA, Infomax, KernelICA, Cubica
Słowa kluczowe: Analiza Składowych Niezależnych, FastICA, Infomax, KernelICA, Cubica

Introduction

Party where a lot of conversations are going on at the same time, music concert where the sound production is very dynamic and mixed with fans shouting, cars noises are often artifacts in recorded material. It is not always possible or available to record in studio, where microphones are good enough to pick up high quality sound. Almost all measured signals consist of multiple other signals. it is possible to remove artifacts. There are a lot of way to do it, but a lot of them are not good enough.

Let's imagine the room where two people are talking at the same time, both generate sound waves. In the same room two microphones are recording the same source from different distances. After finish recording, two signals are available. Is it possible to separate sounds from the mentioned mixture of signals[4]? Problem like this is called the cocktail party problem. Principal component analysis and independent components analysis are perhaps the most popular methods for solving the problem of BSS. Both of them can be used as technique to separate independent sources linearly mixed in several sensors, but as was shown [3] the result of PCA can be far from perfect.
A separation problem of acoustic signals can be widely used in many different fields. As an examples of ICA usage can be mentioned: speech recognition, controlling cars by voice, preparation of radio material and many others.

This paper compares the five ICA algorithms, symmetric orthogonal FastICA, deflation-based FastICA, Infomax, Cubica and KernelICA. The algorithms were compared by examining their efficiency and accuracy of the separation. Two types of tests have been used for this purpose, a) separation of three independent components, b) separation of seven independent components.

Hence, three questions are posed in the paper:
1. Which algorithm has the best performance?
2. How the number of independent components relates to performance?
3. How performance relates to separation precision?
The results of the experiments, together with a short discussion, are presented in the paper.

Independent Component Analysis

The problem of a blind source separation (BSS) consist in finding a matrix W such that the linear transformation will allow to recover the source signals from a set of mixed signals [15-16]. The term ‘blind’ means that no prior information about the source signals or the mixing process is available [15].

Independent Component Analysis (ICA) is one of the most popular BSS method. ICA problem can be stated as follows. Let's assume that there are *n* linear mixtures $x_1,...,x_n$ of n independent components. Vector *x* (observed signals) can be written as:

$$x = As \tag{1}$$

where A represents a mixing matrix with the size of n×n, and s is the vector of independent components. The aim of ICA is to find a matrix W (i.e. an inverse of the matrix A) to reverse the mixing effect. Then, after computing the matrix W, we can obtain the independent components by [17-18]:

$$y = wx \cong s \tag{2}$$

Most of the popular ICA algorithms put some constraints on the mixed signals. First of them is a statistical independence between source signals s; second, a non-Gaussian distribution of the source signals and the third - the equality of the number of source signals and the number of mixture signals. While two first constrains are main assumptions utilized by many algorithms, the third one is introduced only to decrease the algorithm complexity (it causes that the mixing matrix is square). Furthermore, it is assumed that each source signal has the unit variance $E\{s_i^2\} = 1$. To hold this assumption, the matrix of the source signals is whitened before the ICA calculation [17-18]. One more assumption, introduced only to simplify the algorithm, is that all mixture signals are centered.
As was mentioned earlier, ICA does not require any prior information about the source signals. Instead, ICA algorithms utilize the concept of statistical independency of the mixed signals. According to the formal definition, the variables a and b are said to be independent if information about the value a does not give any information about the value b and vice versa [16], [18]. Technically, independence can be defined in terms of the probability density function [17]:

$$f(x_1, x_2,..., x_m) = f_1(x_1) f_2(x_2)... f_m(x_m) \tag{3}$$

There are two main approaches to measuring independence: maximization of non-Gaussianity and minimization of mutual information. Most of the existing ICA algorithms are based on one of them. When the first approach is applied, the task for the algorithm is to modify the components in such a way to obtain the source signals of strong non-Gaussian distribution (the assumption is: the stronger non-Gaussianity, the stronger independence [17]). In other words, the distributions of the mixture signals have to be more Gaussian than the source signals. This approach utilizes various measures of non-Gaussianity, like: kurtosis, negentropy, approximations of negentropy and others [18].

Mutual information, utilized in the second approach, informs how much information about the variable a can be gain from the information about the variable b. Since smaller value of mutual information means that more information about a given system is stored in the variables [17], ICA algorithms based on mutual information approach minimize the mutual information of the system outputs [18].

KernelICA

The Kernel ICA algorithm is based on the minimization of a contrast function based on Canonical Correlation Analysis. This approach is based upon the theory of reproducing kernel Hilbert spaces. Denoting by $x_1, x_2, ..., x_m$ data vector and by $K(x_i, x_j)$ kernel, the algorithm can be written as follows:

1. Whitening the input signals by matrix P

2. The contrast function $C(W)$ is minimized with respect to W, in the following way:

a) The centered Gram matrices $K_1, K_2, ..., K_m$ of the estimated sources $\{y_1, y_2, ..., y_m\}$, where $y^i = Wx^i$ are computed.

b) The minimal eigenvalue of the generalized eigenvector equation $\hat{\lambda}_F^K (K_1, ..., K_m)$ is defined as $K_k \alpha = \lambda D_k \alpha$

c) Then

(4) $C(W) = \hat{I}\lambda_F (K_1, ..., Km) = -\frac{1}{2} \log \hat{\lambda}_F^K (K_1, ..., K_m)$

Cubica

The name of algorithm comes from cumulant term (Cumulant-based Independent Component Analysis). It is based on the diagonalization of cumulant tensors[10][11] and takes third- and fourth-order cumulant tensors $(C^{(y)}_{\alpha\beta\gamma}, C^{(y)}_{\alpha\beta\gamma\delta})$ into account simultaneously.[11] Algorithm uses contrast function which can be written as follows[10][11]:

(5) $\overline{\psi}_{34}(y) = \frac{1}{3!} \sum_{\alpha\beta\gamma \neq \alpha\alpha\alpha} (C^{(y)}_{\alpha\beta\gamma})^2 + \frac{1}{4!} \sum_{\alpha\beta\gamma\delta \neq \alpha\alpha\alpha\alpha} (C^{(y)}_{\alpha\beta\gamma\delta})^2$

Independent components are calculated by maximalization of $\overline{\psi}_{34}$ function.

FastICA - Deflation Approach

The FastICA algorithm, proposed by Hyvärinen and Oja, is an iterative method to find local maxima of a defined cost function [17-18], [3]. The purpose of this algorithm is to find the matrix of weights w such that the projection $(w^T x)$ maximizes non-Gaussianity [3], [18]. As a measure for non-Gaussianity, simple estimation of negentropy based on the maximum entropy principle is used [17-18]:

(6) $J(v) \propto [E\{G(y)\} - E\{G(y)\}]^2$

where: y – standardized non-Gaussian random variable, v – standardized random variable with Gaussian distribution, G(.) - any non-quadratic function.

There are two classes of FastICA algorithms, the deflation algorithms (called also one-unit algorithms) and the symmetric algorithms [19]. In the deflation approach, the independent components (ICs) are extracted sequentially, one by one. The algorithm can be summarized as follows [18], [20]:

1. Choose an initial vector w (e.g. random)
2. Do steps 3-6
3. $w^+ = E\{xg(w^+ x)\} - E\{g'(w^+ x)\}w \backslash$
4. $w = \frac{w^+}{\| w^+ \|}$
5. Do the Gram-Schmidt orthogonalization:

$$w_{p+1} = w_{p+1} - \sum_{j=1}^{p} w_{p+1}^T w_j w_j$$

$$w_{p+1} = \frac{w_{p+1}}{\sqrt{w_{p+1}^T w_{p+1}}}$$

6. Stop if not converged

Gram-Schmidt procedure, used in the algorithm, prevents different vectors from matrix w from converging to the same maxima [18]. The order, in which the independent components are extracted, depends on the initial value of w.

FastICA - Symmetric Approach

The only difference between deflation approach and symmetric approach is the procedure of weights calculation. While in deflation approach vectors of weights are calculated one by one, in symmetric approach the estimation of all components (all weights vectors) proceeds in parallel [18-19]. Instead of Gram-Schmidt procedure, the following formula is used in the orthogonalization step:

(6) $w = (ww^T)^{-1/2} w$

where w is the matrix of weights vectors $(w_1, ..., w_n)^T$.

The square root of ww^T is obtained from the eigenvalue decomposition of $ww^T = QDQ^T$ as [21]:

(7) $(ww^T)^{-1/2} = QD^{-1/2}Q^T$

where Q is the matrix of eigenvectors and D is the diagonal matrix of eigenvalues.

The algorithm is performed until the stop condition (e.g. given by 7 [19]) is met:

(8) $1 - \min(abs(diag(w^T w_{old}))) < \varepsilon$

where ε is a chosen constant.

Infomax

Infomax algorithm is based on the general optimization principle for neural networks and other processing systems described by Linsker in 1987 [22]. In general this principle says that a function that maps a set of input values a to a set of output values b should be chosen or learned so as to maximize the average Shannon mutual information between a and b. The ICA algorithm utilizing this principle was first proposed in 1995 by Bell and Sejnowski [23] and then in 1997 optimized by Amari [18], [20].

Infomax algorithm for calculating independent components is based on the maximization of the output entropy of a neural network with non-linear outputs [18]. The most essential parameter of this algorithm is a learning rate which does not need to be constant over time and which should give a good compromise between speed of learning and estimation precision [18], [24]. The weights of this neural network are updated according to the following formula [17], [20], [25]:

(9) $w_{k+1=}w_k + \mu_k [I + 2g(y_k) y_k^T] w_k$

where: $y-$ matrix of source estimation ($y = Wx$); k – number of iteration; l – the identity matrix; μ_k – learning rate which may depend on k; $g(.)$ – a nonlinear function.

Mostly a classic logistic function is used as a nonlinear function g [25]:

(10) $$g(y) = \frac{1}{1+e^{-y}}$$

however, sometimes also its extended version is applied:

(11) $$g(y) = y \pm \tanh(y)$$

Using (9), the Infomax algorithm can be summarized as follows [20]:

1. $x = perm(sources)$ = perm(sources);
2. $y = w \times x$
3. $g = \frac{1}{1+e^{-yk}}$
4. $gu = g \times y^T$
5. $gu = l - 2 \times gu$
6. $w_{k+1=}w_k + \mu_k \times gu \times w_k$

where $perm$ is random permutation.

Experimental Settings

In all experiments the same data set was used - a data set belongs to West Pomeranian University of Technology.[14] The data set is divided into twenty directories. The name of the each directory entry contains a number of singer and the character gender an 'f' for female or a 'm' for male. The next levels of directories contains again a number of signer and the character gender but the name contains also a number of exercise as a e01, e03 and e05. All exercises have been made in legato technique. The instructions for each exercise are as follows:

1. $e01$ - $sing : a, e, i, o, u$
2. $e03$ - $sing : do, la, a, do, la, a, do$
3. $e05$ - $sing : a, e, i, o, y.$

The sound file name contains the same information as subdirectories name. In addition it has some useful informations:

- pN – where N is phrase number
- oN – where N is octave number
- nN – where N is sound number in octave in the ascending direction, including semitones
- $d0$ or $d1$ – 0 means the ascending direction, 1 descending
- v or p – v for vocal or p for piano

The samples length depends on the exercise. It is not regular. Bitrate for each sample is 768kb/s while sampling rate is 48000 Hz. All files are monophonic with *.wav extension. The symbol od used coded is PCM S16 LE. All data set contains 2277 files.

Seven files have been chosen for test. Chosen set has sings with melody as well as single piano. Piano sounds have been replicated and saved in single file. Full names of chosen files are:

- s18m_e03_p26_o2_n10_d0_v.wav
- s20f_e05_p29_o4_n03_d0_v.wav
- s18m_e01_p22_o3_n11_d0_v.wav
- s20f_e05_p34_o3_n10_d0_v.wav
- s12m_e03_p12_o3_n12_d0_p.wav
- s06f_e05_p28_o3_n08_d0_p.wav
- s06f_e05_p23_o4_n01_d0_v.wav

New files are named with the convention of wave#, where # starts at 1 and increments automatically with each new file. All of the files differed in length and so it was necessary to standardize to the length of the shortest file. Then performed, two types of tests. The First one separates mixing of three independent components, second one of seven. Signals for first test are obtained from below formulas:

- $mix1 = wave1 - 2.14 * wave2$
- $mix2 = 1.9 * wave1 + 4.13 * wave2;$
- $mix3 = 2.03 * wave3 + 1.12 * wave2;$

Formulas for second test signals are:

- $mix1 = 0.89 * wave1 - 2.14 * wave2 + 0.8 * wave5 - 0.4 * wave7;$
- $mix2 = 0.7 * wave3 + 4.13 * wave4 - 0.4 * wave6 + 5.6 * wave5;$
- $mix3 = 0.73 * wave3 + 3.12 * wave2 + 0.4 * wave4;$
- $mix4 = wave1 - 2.14 * wave2 + 1.54 * wave3 + 1.1 * wave4;$
- $mix5 = 1.9 * wave4 + 4.13 * wave5 + 2.4 * wave6;$
- $mix6 = 1.3 * wave7 + 0.62 * wave6 + 0.2 * wave1 - 1.6 * wave4;$
- $mix7 = 2.03 * wave5 + 1.12 * wave7 + 4.3 * wave6;$

The resulting file are named with the convention of mix#, where # starts at 1 and increments automatically. Implementations of KernelICA and Cubica algorithms can be found from the Internet. The first one is available at: [17], the second one at: [18]. Both of them where started with default parameters. Two FastICA and Infomax algorithms were implemented by author. The parameters for FastICA are:

- $epsilon = 0.001$ (for stop condition)
- $maxNumIterations = 1000$, the iterations limit
- the initial vector is random.

Parameters for Infomax are:

- $lrate = 0.001$ - learning rate,
- $g(y) = \frac{1}{1+e^{-y}}$ - optimization function,
- $epsilon = 0.0001$ for stop condition.

The running time of algorithms is measured by counting the elapsed CPU clock. Results from intercorrelation tests provided evidence of how good separation is achieved with the all ICA's algorithms. Cross-correlation is a measure of similarity of two signals $x(t)$ in time t and $y(t)$ in time $t+T$ and can be written as follows[12]:

(12) $$R_{XY}(\tau) = \lim_{T->\infty} \frac{1}{T} \int_0^T x(t) * y(t+\tau) * dt$$

where: $x(t)$ and $y(t)$ - input data, $R_{XY}(\tau)$ - cross correlation function for signals $x(t)$ and $y(t)$, t - delay between signals, T - period.

The result set returned by the function are corelation values. The cross correlation function is the normalized cross-covariance function. The normalized cross-correlation function known as the cross-correlation function coefficient (normalized cross-covariance function) is defined as:

$$\rho[x(t), y(t)] = \frac{\operatorname{cov}[x(t), y(t)]}{\sqrt{\sigma_x^2 * \sigma_y^2}} \quad (13)$$

where: $\sqrt{\sigma_x^2}$ - standard deviation of x, $\sqrt{\sigma_y^2}$ - standard deviation of y

The cross-correlation function coefficient was used to define a measure of similarity between signals. These cross-correlation function coefficient values are in the range -1.0 for ideal negative correlation to 1.0 for ideal positive correlation. Closer to 0 value will result worst similarity.[15].

Results

The experiments were performed with the application prepared for the Matlab 7.12.0 environment. The tests were performed on a computer with Processor type: Intel(R) Core(TM) i3-2350 @ 2.30 GHz, Total Physical Memory: 10.0 GB and Total Disk Space: 500 GB. Operating system was Windows 7 and all the updated patches installed.
The main aim of the experiments described in the paper was to find out what is the performance and how it is related to the accuracy of the five ICA algorithms. The running times of the algorithms are shown in Table 1:

Table 1. The running times of the ICA algorithms.

	The average running times of the algorithms:	
Algorithm	For three components:	For seven components:
KernelICA	557.95	7126.6
Cubica	0.62	4.34
FastICA-Symmetric	2.08	11.6
FastICA-Deflation	2.22	10.83
Infomax	22.70	89.23

The second shortest time is obtained by the Cubica algorithm. For three independent components it was shorter than 1 second, for seven was about 4 seconds. Both FastICA algorithms get similar times, 2 seconds for three IC and 11 for seven. Separation by Infomax was calculated in 22 and 89 seconds. The KernelICA algorithm requires longest times.

As was mentioned, the cross-correlation function coefficient can be used as a reliable measure of the separation quality. Since standard ICA comes out with an unordered set of sources, each input signals has be compared with each out. Outputs are named with as a U#, where # starts at 1 and increments automatically. The example of is shown in Table 2:

Table 2. Example of the accuracy of the separation.

KernelICA			
ρ[x(t),y(t)]	U1	U2	U3
S1	1.00	0.01	0.37
S2	0.00	0.99	1.00
S3	0.00	0.00	0.00

All output signals were compared in the way schematically shown in Table 2. All average results are grouped according to the ICA algorithms. Tables 3 and 4 presents a summary of separation accuracy results for 3 and 7 independent components. Tables contain only results for matching output signals to corresponding unmixed wave file.

Table 3. Separation accuracy of ICA for three independent components.

	KernelICA	Cubica	FastICA – sym.	FastICA – defl.	Infomax
S1	1	1	1	1	0.79
S2	1	1	0.99	0.99	0.82
S3	1	1	1	1	0.99
Average:	1	1	0.99	0.99	0.84

Table 3 shows that mixing of three waves files are correctly separated by almost all algorithms. Only Infomax not correctly solved the separation. Outputs from KernelICA and Cubica are perfect - cross-correlation function coefficient were 1 for them.

Table 4. Separation accuracy of ICA for seven independent components.

	KernelICA	Cubica	FastICA – sym.	FastICA – defl.	Infomax
S1	0.99	0.99	0.99	0.99	0.83
S2	0.99	0.99	0.99	0.99	0.86
S3	0.88	0.70	0.73	0.99	0.73
S4	0.88	0.71	0.73	0.99	0.68
S5	0.99	0.99	0.99	0.99	0.99
S6	0.99	0.99	0.99	0.99	0.99
S7	0.99	0.99	0.99	0.99	0.97
Average:	0.96	0.91	0.92	0.99	0.87

The average separation values that are presented in Table 4 shows how the independent components number influence the ICA precision. The average precision of Infomax increased to 0.87, however this is still the worst value. Deflation FastICA keeps good result (0.99 of average precision). The rest of algorithms get worse results than for previous test.

Conclusion

In Section 1 three questions regarding the efficiency of ICA transformation were posed:
1. Which algorithm has the best performance?
2. How the number of independent components relates to performance?
3. How performance relates to separation precision?

Table 1 can used to answer the first question. The shortest time for 3 and 7 independent components was obtained by Cubica. The second shortest time was almost 4 times longer for 3 signals and twice time longer for seven. This shows how efficiency Cubica is to the goals of these separations. Unfortunately its perfect precision is lost. The best precision is achieved by FastICA deflation algorithm. The average value of it is 0.99 - this means that it is almost perfect. This is also the only algorithm that keeps previous precision for seven independent components. Time is increased to four times more than for three inputs, but it was still good enough. Good precision was also obtained by KernelICA, but the calculation time was hundreds of times longer than for other ICA's methods. Unfortunately this property is not suitable for normal tasks. The Infomax yields average efficiency results, and worst separation precision. The reason for this may be in wrong input parameter of the Infomax algorithm.

Authors
M.Sc., Paweł Górski, West Pomeranian University of Technology, Faculty of Computer Science and Information Technology, Żołnierska 49, 71-210 Szczecin, Poland, email: pagorski@wi.zut.edu.pl.