Independent Component Analysis for Blind Source Separation

Gerald Schuller, Oleg Golokolenko

Ilmenau University of Technology
and Fraunhofer Institute for Digital Media Technology (IDMT)

June 26, 2017

Introduction

- Goal: Separate sources with multiple microphones
- Different microphones pick up sound with different amplitudes and delays
- For simplicity start with panning (different amplitudes) and no delays
- With programming exampl in Python
- This is for easier understandability,
- to test if and how algorithms work,
- and for reproducibility of results, to make algorithms testable and useful for other researchers.

Introduction

- Goal: Separate sources with multiple microphones

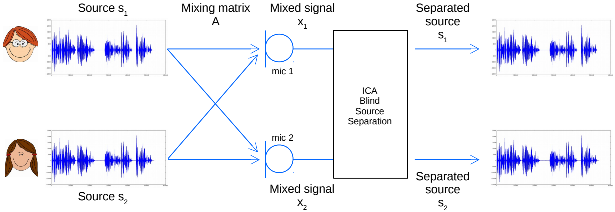

Figure: A Mixing/Unmixing Visualization

Assumptions

- The original sources are statistically independent,
- and have a non-Gaussian probability distribution.
- To visualize statistical dependencies we can use the scatter plot
- in Python: plt.scatter(sources[:,0],sources[:,1])

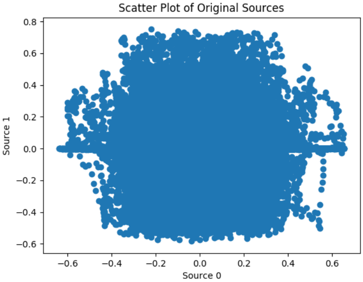

Figure: A scatter plot of the original sources. Observe that there are no diagonal structures, just vertical and horizontal

The Mix at the Microphones

- The microphone signals have strong statistical dependencies, because each picks up each source, but with slightly different amplitudes.
- Hence the scatter plot now has mostly diagonal structures,

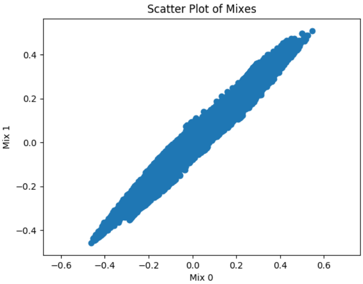

Figure: A scatter plot of the the mixes at the microphone. Observe that there are mostly diagonal structures, indicating dependencies

Approach

- Independent Component Analysis (ICA)
- Books:
- "Independent Component Analysis", A. Hyvärinen, J. Karhunen, E. Oja, 2001 John Wiley & Sons.
- "The Elements of Statistical Learning" Book by Jerome H. Friedman, Robert Tibshirani, and Trevor Hastie Originally published: 2001.

Independet Component Analysis

ICA consists of several steps

1. DC (mean) removal
2. Decorrelation using the Karhounen-Loew Transform or Principal Component Analysis
3. "Whitening", normalizing the power of the decorrelated components
4. Apply rotation matrix to the set of components, rotate until the components have a minimum similarity using an entropy based similarity measure,like Kullback-Leibler Divergence.

ICA, Step 1

- DC (mean) removal
- In Python:
- The 2 audio mixes are in array `X` (tall matrix)
- DC removal: `X`= X- X.mean(axis=0)

ICA, Step 2

- Decorrelation using the Karhounen-Loew Transform or Principal Component Analysis
- In Python, first compute the correlation matrix, divided by the signal length,
- `Axx = np.dot(X.T, X)/X.shape[0]`
- Then compute its Eigenvalues and Eigenvectors (the KLT matrix T),
- `Lambda, T = LA.eig(Axx)`
- Apply the transform T to obtain de-correlated signals,
- `X=np.dot(X,T)`

ICA, Step 2

- To visualize the correlation before and after the transform, we can use a scatter plot,
- plt.scatter(X[:,0],X[:,1])
- We can now see no **diagonal structures anymore**. But maybe they are just **hidden** because of the unequal energies of the components after KLT!
- This is confirmed when **listening** to the signals, they are still mixed!

ICA, Step 2

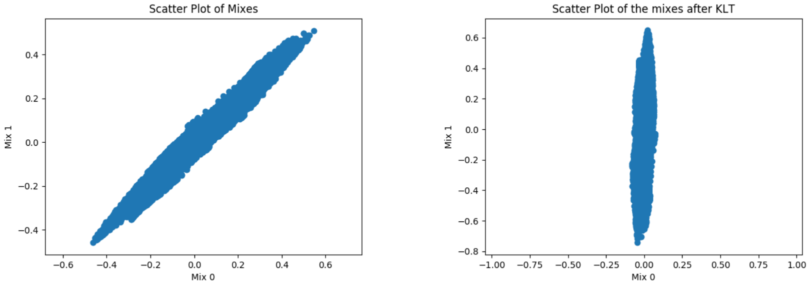

Figure: The effect of the KLT on the scatter plots, left before, right after KLT. Observe there are no diagonal structures visible anymore.

ICA, Step 3

- "Whitening", normalizing the power of the decorrelated components
- This makes hidden dependencies "visible".
- Their powers are in the Eigenvalues, hence we just need to divide our signals by the square roots of the Eigenvalues to obtain normalized powers.
- in Python:
- `X= np.dot(X,np.diag(1.0/np.sqrt(Lambda)))`

ICA, Step 3

- The scatter plot now reveals hidden dependencies as diagonal structures.

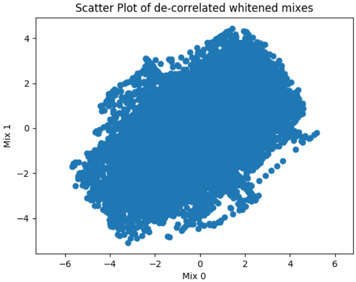

Figure: A scatter plot of the whitened signals. Observe the diagonal structures.

ICA, Step 4, Rotation

- Apply rotation matrix to the set of components, rotate until the components have a minimum similarity using an entropy based similarity measure,like Kullback-Leibler Divergence.
- In Python, we construct a rotation matrix R for angle α in degrees as
- ```
  theta = np.radians(alpha)
  c, s = np.cos(theta), np.sin(theta)
  R = np.array([[c, -s], [s, c]])
  ```
- We then rotate the signals with
- `X_prime = np.dot(X,R)`
  ```

ICA, Step 4, Similarity Measure

- We need to find the rotation angle which gives us the vertical and horizontal structure back.
- This is an indication of independence.
- But visual measures don't work in an numerical approach.
- Hence we choose a measure for a statistical similarity
- For instance the Kullback-Leibler Divergence of 2 distributions P and Q, defined as
- $D_{KL}(P, Q) = \sum_{n=i} P(i) \log \frac{P(i)}{Q_1(i)}$
- i runs over the (discrete) distributions
- Observe that is is non-symmetric, $D_{KL}(P, Q) \neq D_{KL}(Q, P)$

ICA, Step 4, Similarity Measure

- In Python we have the function scipy.stats.entropy.
- It computes the Kullback-Leibler Divergence wen we supply it with 2 distributions.
- We compute the distributions using the np.histogram function.
- Since we are looking for a minimum, we take the minus sign, and we add a small ϵ to avoid infinities at zeros:
- hist0, bins =np.histogram(X_prime[:,0],bins=1000)
  hist1, bins =np.histogram(X_prime[:,1],bins=1000)
  similarity= -stats.entropy(hist0+1e-6,hist1+1e-6)
- We turn all this into a function:
- entropysimilarity(alpha,X)

ICA, Step 4, Similarity Measure

- We can now plot the resulting similarity measure over the rotation angles, in Python:
-
```
similarity=np.zeros(100)
for n in np.arange(100):
   similarity[n]=entropysimilarity(180.0/100*n,X)
plt.plot(np.arange(100)*180.0/100 , similarity)
```

ICA, Step 4, Similarity Measure

The resulting plot is

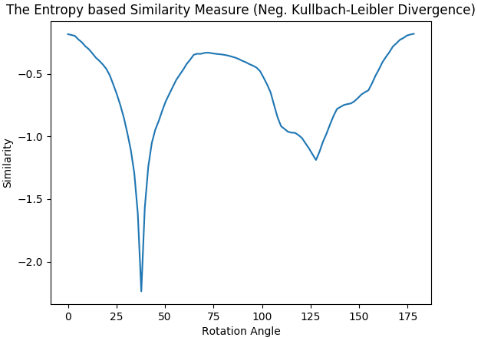

Figure: Our similarity measure of the signals over the rotation angle. Observe the minima.

ICA, Step 4, Minimization

- Searching the rotation angles for the minimum similarity corresponds to a "brute force" minimization.
- But Python has powerful optimization functions which we can apply here.
- We use scipy.optimize.fminbound
- and apply it as
- alpha_minimized = opt.fminbound(entropysimilarity, 0.0, 180.0, args=(X,), xtol=1e-05, maxfun=500)

ICA, Step 4, Minimization

- We can now apply the rotation with the optimum angle to our signal.
- To check if we got the right rotation angle, we plot the resulting scatter plot, to see if we have no diagonal structure:

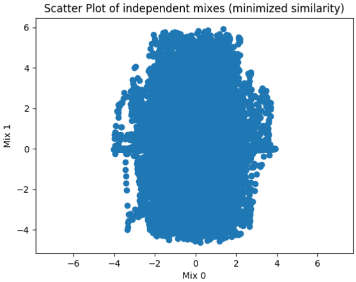

Figure: A scatter plot of the signals after ICA. Observe the non-diagonal structures.

ICA, Step 4, Minimization

- Listening to the resulting signals confirms that they are really separated!
- Python demo program:
- python ICAseparation.py

Conclusions

- We saw that we can use ICA to effectively separate sources out of a mix.
- Python can be use for a simple implementation
- Next step: apply it to signal mixes including delays.