SOURCE SEPARATION FOR AUDIO-APPLICATIONS

Sebastian Ament* and Aarti Bagul †

Abstract. The goal of this project is to successfully implement and test an algorithm for audio source separation. We are going to use the Itakura-Saito divergence and an additional smoothness penalty function to construct a nonnegative matrix factorization (NMF) for a frequency spectrogram of a sound file. Further, we are going to use an online algorithm to reduce the memory and time complexity of calculating the NMF. This will allow us to efficiently test the algorithm on sound files with nearly arbitrary length.

1. Introduction:. Source separation involves estimating the signal produced by an individual source from a mixed signal produced from multiple sources. A classic example of this is the cocktail party problem, where there are a number of people talking simultaneously and a listener is trying to separate their individual voices and follow the discussion. It can also be used to separate the signals produced by individual musical instruments in an audio file. Another interesting application is medical imaging of the brain with magnetoencephalography (MEG)

2. Method:. Nonnegative matrix factorization using the Itakura-Saito divergence has been proven effective for audio source separation. The idea is the following. Given an input sound file, one can calculate frequency spectrograms for small time intervals throughout the length of the file. This leads to a matrix representation $V \in \mathbb{R}_+^{F\times N}$ of the input file, where F is the number of frequency bins and N is the number of spectrograms at different times.

Then, we want to find a factorization of the form

$$V \approx WH$$

where $W \in \mathbb{R}_+^{F\times K}$ and $H \in \mathbb{R}_+^{K\times N}$ are also both positive matrices. K is typically chosen such that $FK + KN << FN$, as to do dimensionality reduction on the data. W is usually called "dictionary" matrix while H is called "activation" matrix. The motivation behind this naming convention is that we are trying to create a "dictionary" of signals $\{\vec{w}_1, ..., \vec{w}_K\}$ such that for each time $t \in \{1, 2, ..., N\}$, we can find activation coefficients $h_{1t}, ..., h_{Kt}$ satisfying

$$\vec{v}_t = \sum_{k=1}^{K} h_{kt}\vec{w}_k$$

Now, as each $\vec{w}$ represents a spectrogram itself, its entries should be nonnegative. Further, the h's should to be nonnegative, because having the negative of a dictionary signal constitute a spectrogram is not consistent with our representation model of the signal. Additionally, allowing negative entries in either or both H and W does not work well in practice. The decomposition $V \approx WH$ is an expression of this idea for all frequency spectrograms simultaneously.

3. Mathematical Problem Formulation:. Above, we introduced the concept of a nonnegative matrix factorization. In order to make the NMF computationally tractable, it is

*New York University, New York, NY. E-mail: sebastian.ament@nyu.edu.
†New York University, New York, NY. E-mail: aarti.bagul@edu.

usually posed as an optimization problem. Most generally, we want to find $W \in \mathbb{R}_+^{F\times K}$ and $H \in \mathbb{R}_+^{K\times N}$ which minimize

$$D(V|WH) = \sum_{f=1}^{F}\sum_{n=1}^{N} d(V_{nf}|\,[WH]_{fn}) \tag{3.1}$$

for some penalty function d. Note that this minimization is subject to nonnegativity constraints of W and H. For source separation of audio files, the Itakura-Saito divergence $d_{IS}(y,x)$, given by

$$d_{IS}(y,x) := \sum_i \left(\frac{y_i}{x_i} - \log\left(\frac{y_i}{x_i}\right) - 1\right)$$

has proven effective. Most importantly, d_{IS} is scale-invariant, that is, $d_{IS}(\alpha x|\alpha y) = d_{IS}(x|y)$. Given an audio file with large dynamic ranges and a penalty function without scale-invariance, the optimization would neglect quiet passages for the sake of optimizing W and H to fit the loudest passages best. With scale-invariance however, we do not have this bias.

Furthermore, as we are applying the NMF to audio signals, it is reasonable to assume that the activation coefficients $\vec{h}_t$ should change continuously in time. To enforce smoothness of $\vec{h}_t$, we define

$$P_2(H) = \sum_{k=1}^{K}\sum_{t=2}^{N} d_{IS}(h_{k(t-1)}|h_{kt})$$

as a second penalty function. Using the IS divergence, P_2 punishes big changes in h_{kt}. Combining both penalty functions linearly, we get

$$C_2(W,H) = D_{IS}(V|WH) + \lambda P_2(H)$$

where $\lambda \in \mathbb{R}_+$ can be adjusted to give more or less weight to the smoothness of $\vec{h}_t$.

4. Algorithms. The algorithms used are:

Online Algorithm for IS-NMF

Input Training set, $W^{(0)}$, $A^{(0)}$, $B^{(0)}$, ρ, β, η, ϵ
repeat
t ← t+1
draw point v_t from training set.
$h_t \leftarrow \arg\min_h\, d_{IS}(\epsilon + v_t, \epsilon + Wh)$
$a^{(t)} \leftarrow (\frac{\epsilon+v_t}{(\epsilon+Wh_t)^2} h_t^T).W^2$
$b^{(t)} \leftarrow \frac{1}{(\epsilon+Wh_t)^2} h_t^T$
if $t \equiv 0[\beta]$
$A^{(t)} \leftarrow A^{(t-\beta)} + \rho \sum_{s=t-\beta+1}^{t} a^{(s)}$
$B^{(t)} \leftarrow B^{(t-\beta)} + \rho \sum_{s=t-\beta+1}^{t} b^{(s)}$
$W^{(t)} \leftarrow \sqrt{\frac{A^{(t)}}{B^{(t)}}}$
for k = 1...K
$s \leftarrow \sum_f W_{fk}$,
$W_{fk} \leftarrow W_{fk}/s$
$A_{fk} \leftarrow A_{fk}/s$,
$B_{fk} \leftarrow B_{fk}$ x s
end for

end if
until $\left\|W^{(t)} - W^{(t-1)}\right\|_F < \eta$

Smooth IS-NMF Algorithm

Input: nonnegative matrix $\mathbf{V}$
Output: nonnegative matrices $\mathbf{W}$ and $\mathbf{H}$
Initialize $\mathbf{W}$ and $\mathbf{H}$ with nonnegative values
Compute $\hat{\mathbf{V}} = \mathbf{WH}$
for $i = 1 : n_{iter}$ **do**
// Update $\mathbf{H}$
$\mathbf{G}^- = \mathbf{W}^T(\mathbf{V}.\hat{\mathbf{V}}^{-2}); \mathbf{G}^+ = \mathbf{W}^T(\hat{\mathbf{V}}^{-1})$
Update $\mathbf{h}_1$
for $n = 2$: N - 1 **do**
$\mathbf{h}_n^{(i)} = \sqrt{[g_n^-.(\mathbf{h}_n^{(i-1)})^2 + \lambda \mathbf{h}_{n-1}^{(i)}]/[g_n^+ + \lambda/\mathbf{h}_{n+1}^{(i-1)}]}$
end for
Update $\mathbf{h}_N$
Compute $\hat{\mathbf{V}} = \mathbf{WH}$
// Update $\mathbf{W}$
$\mathbf{W} \leftarrow \mathbf{W}[(\hat{\mathbf{V}}^{-2}.\mathbf{V})\mathbf{H}^T]/[\hat{\mathbf{V}}^{-1}\mathbf{H}^T]$
Compute $\hat{\mathbf{V}} = \mathbf{WH}$
Normalize $\mathbf{W}$ and $\mathbf{H}$ together
end for

5. Dataset and Software:. We are planning to generate a dataset by generating multiple recordings of our voices and using Sound eXchange to manipulate these audio files. We are planning to use the pyFASST module (Python implementation of the Flexible Audio Source Spearation Toolbox (FASST)) for the source separation.

6. Baseline:. First, we made a recording of a female speaker counting from one to ten in English, and a recording of a male speaker counting from one to ten in German. We then mixed both recordings together to get one 14 second mono .wav file, with a sampling rate of 44,100. We then down-sampled the recording by a factor of 2. This step was not crucial for the performance of our baseline programs, as the input file was small already. However, it may be of higher importance for longer recordings which we are going to deal with later. Next, we created a power spectrogram of the down-sampled recording. We chose a window size of 100 samples (4.5 ms) for the short time Fourier transform and computed a spectrogram with 129 frequency bins and 5291 time frames. The power spectrogram of the recording can be seen in Figure 6.1

To do the source separation, we used the nnmf method in Matlab with a multiplicative update algorithm, and with the number of iterations set to 10,000 and $K = 10$ (K as defined in 2.). We repeated this 100 times with random matrix initializations. We got a non-negative matrix factorization W,H with root mean squared error of 0.19.
We tried another approach using the FastICA module in scikit- learn.
We used the same audio file from before and used the python wave module to get information about the parameters of the audio signal. We used this to plot the audio signal and generate a spectrogram. We then used the FastICA module to separate the components of the signal and generate graphs of these components. The results were not promising and the graphs of the two components had substantial overlap. Then, we used PCA to first reduce the dimensionality and then applied ICA. This gave us better results than using just ICA.

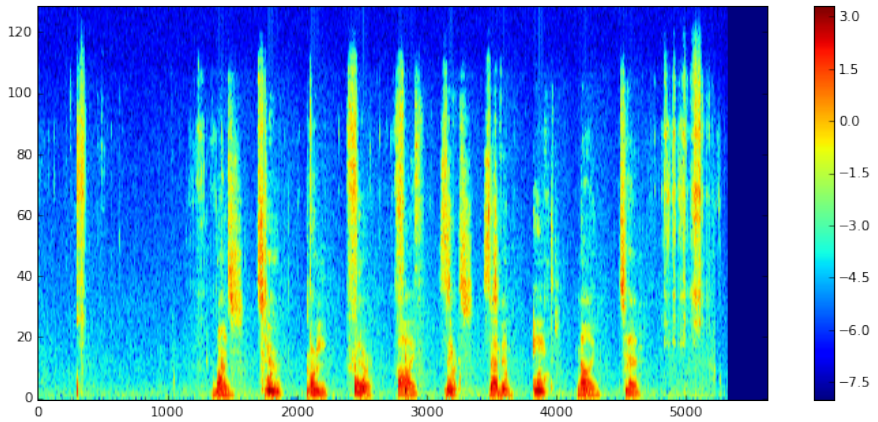

Figure 6.1. *Power spectrogram of recording.*

7. Homework 4: Project. First, we would like to mention that, as we are going to work on blind source separation, this exercise might seem to be of limited direct use. However, an interesting idea for separation of instruments would be to precompute a centroids in a k-means clustering which represent indicative signals for different instruments, and use it to detect what instruments are present. For this exercise, we computed a "vocabulary" for our two voices using two training recordings. Subsequently, we used it to recognize our voices on test recordings. In the following, we describe the specific procedure.

Like in our baseline, we computed power spectrograms for our input files, where the frequency bins represent the features of the samples. We tested the optimal window size for the short-time Fourier transform using a grid search with dyadic increments. That is, our grid for the window size was 128, 256, 512, 1024, and 2048. However, it is critical to note that the length of the feature vector grows with the grid size, as a longer Fourier expansion has to be used to accurately portray the signal in each window. Additionally, we know that k-means does not perform well in very high dimensions. That is, the performance of k-means, especially in the worst case, is negatively affected by increasing the window size. On the other hand, decreasing the window size has a negative effect on the amount of detail that can be described by the resulting feature vector. As a consequence, we chose 256 as our optimal window size.

Regarding the optimal number of clusters for the k-means part of the approach, we made an interesting observation. As we only considered our two voices, the number of clusters to necessary to distinguish between our voices was relatively low. With only about 20 clusters, the difference between the histograms of our voices were already significant enough. However, this does not mean that the audio signal was represented well by these clusters. Indeed, the total distortion of the representation of the audio signals by only 20 clusters was expectedly large. Therefore, to generalize this approach, we will have to employ k-means with far larger number of clusters. For future investigations, we are considering using this approach to detect different instruments in a sound file. Once detected, we could use this information to initialize the matrix in the NMF algorithm using k-means representations of the present algorithms. We will report on that later on.

8. Hottopixx algorithm. We implemented the Hottopixx algorithm and tested it on the power spectrogram of two Jazz pieces, each about a minute long. The spectrogram had 129 frequency bins, and about $4 \cdot 10^4$ temporal bins. In order to speed up our experimental computations, we chose a uniform sample of about 10^3 columns of the spectrogram. We then ran several experiments with 10^3 epochs of the incremental gradient descent with primal stepsize $s_p = 10^{-1}$ and dual stepsize $s_d = 10^{-2}$, with varying input parameter r. Each run took approximately three minutes.

Interestingly, if we choose a small r, relative to $f = 129$, the algorithm does not converge to a solution to the NMF problem. The empirically smallest value of r that seems to work on the songs is 70. This suggests that the spectrogram data exhibits a high degree of linear independence. As a consequence, the data cannot be efficiently written as a linear combination of a small number of rows of the spectrogram. It is notable that the authors of the paper ran their benchmarks on synthetic datasets, which were created by a linear combination of no more than three random vectors (see paragraph 3 on page 8).

To visualize the results, we plotted rows of the activation matrix W and columns of the library matrix F in the accompanying notebook.

We also tried to run the author's implementation of hottopixx on our dataset, but failed to get the data in the right format to be accepted by the algorithm.

9. Implementing the IS-divergence .

$$d_{IS}(x, y) = \sum_i \frac{x_i}{y_i} - \log \frac{x_i}{y_i} - 1$$

so that

$$\nabla_y d_{IS}(x, y) = \sum_i \frac{1}{y_i} \left(1 - \frac{x_i}{y_i}\right) e_i$$

and

$$\nabla_x d_{IS}(x, y) = \sum_i \left(\frac{1}{x_i} - \frac{1}{y_i}\right) e_i$$

where e_i is the ith canonical basis vector. As one can see above, the expression for ∇_y can be more efficiently evaluated on a computer than ∇_x. Indeed, ∇_y needs less flops, and hence less time, to be evaluated than ∇_x. We ran tests to quantify the speed difference. In particular, we computed ∇_x and ∇_y of the objective function 10^5 times with random 128-dimensional input vectors x and y. ∇_y took 1.05 s while ∇_x took 1.15 seconds. As a consequence, one could shave off approximately 10% of the execution time of the algorithm by rewriting the optimization problem.

For this reason, we experimented with exchanging the problem of minimizing $d_{IS}(v_t, Wh)$ over all positive h, with $\min_h d_{IS}(Wh, v_t)$ in line 3 of the online algorithm. Even though the two optimization problems are not equivalent, as d_{IS} is not symmetric, there is a priori no reason to expect a differing quality of the resulting solutions. However, we noticed that the rest of the algorithm is not compatible with this change. This is why we did not end up changing the optimization problem, as mentioned above.

For the minimization procedure, we implemented a gradient backtracking algorithm. The gradient step of this problem is given by

$$\nabla_h d_{IS}(v_t, Wh) = \frac{1}{Wh} \left(\vec{1} - \frac{1}{Wh}\right) \cdot W$$

where the division is meant to be element-wise.

Another idea of improvement we experimented with was adding a smoothness term to this minimization, similar to the one given in [2] for batch NMF. In particular, we define

$$S_{\lambda,t}(h) = \lambda \left[d_{IS}(H_{\cdot(t-1)}, h) + d_{IS}(H_{\cdot(t+1)}, h)\right]$$

where $H_{\cdot(t\pm1)}$ is the column before, and after h_t in H, respectively. The motivation for this term is to force a continuous change of the columns of H. This is desirable specifically for audio applications, as shown by [2].

In practice, it does not make sense to include this term in the very beginning of the algorithm. Indeed, we have no reason to expect the initialization of H to be close to the solution and the inclusion of S would enforce smoothness of the updated columns to the previous, most likely inaccurate columns. As a consequence, this would only slow down the convergence rate of the algorithm. Rather, we run the algorithm for a couple of epochs, until the values of H stabilize, and then include S to enforce the smoothness of the columns of H. To do this, we replace step 3 of the online algorithm with the minimization problem

$$h_t = \arg\min_h [d_{IS}(v_t, Wh) + S_{\lambda,t}(h)]$$

The gradient of S is given by

$$\nabla_h S_{\lambda,t}(h) = \lambda \left[\frac{1}{H_{\cdot(t-1)}} + \frac{1}{H_{\cdot(t+1)}} - 2\frac{1}{h} \right]$$

again where the division is understood to be element-wise. In our implementation, we follow the tip in Lefevre and minimize $d_{IS}(\epsilon + v_t, \epsilon + Wh)$ for some small ϵ to avoid the singularity of d_{IS} at $(0, y)$.

10. K-means initialization.. We implemented the mini-batch k-means algorithm (described below)[3] and ran it on the power spectrogram to get initialize W and get a lower value for the initial IS divergence.

Mini-batch k-Means algorithm

Input number of clusters k, mini-batch size b, iterations t, data set X
Initialize each $\mathbf{c} \in C$ with an $\mathbf{x}$ picked randomly from X
$\mathbf{v} \leftarrow 0$
for $i = 1$ to t **do**
  $M \leftarrow$ b examples picked randomly from X
  for $\mathbf{x} \in M$ **do**:
    $\mathbf{d}[\mathbf{x}] \leftarrow f(C, \mathbf{x})$ //Cache the center nearest to $\mathbf{x}$
  end for
  for $\mathbf{x} \in M$ **do**
    $\mathbf{c} \leftarrow \mathbf{d}[\mathbf{x}]$ //Get cached center for this $\mathbf{x}$
      $\mathbf{v}[\mathbf{c}] \leftarrow \mathbf{v}[\mathbf{c}] + 1$ // Update per-center counts
      $\eta \leftarrow \frac{1}{\mathbf{v}[\mathbf{c}]}$ //Get per-center learning rate
      $\mathbf{c} \leftarrow (1 - \eta)\mathbf{c} + \eta\mathbf{x}$ //Take gradient step
  end for
end for

We modified the algorithm to use k-means++ to initialize the centers from X (instead of picking k random centers).

11. Experimental Results.

11.1. Final Dataset. In addition to the audio file we used for our baseline, we experimented with two other audio files. We used a 5 second piano recording of "Mary had a little lamb". We chose a window size of 512 samples and overlap of 256 samples for the short time Fourier transform and computed the power spectrogram shown below.

We also experimented with 2 Jazz recordings with multiple instruments, both of which were about 2 minutes long.

11.2. Gradient Descent. For the minimization of $d_{IS}(v_t, Wh)$ in the online algorithm, we wrote a gradient descent algorithm with backtracking. Importantly, this algorithm had to enforce the non-negativity of the solution. In order to achieve this, we adjusted the initial step size η by scaling it with a factor β until the step leads to a non-negative vector. This vector then serves as the input to a regular backtracking algorithm with the same back step factor β. The algorithm terminates when a maximum number of backtracking steps m is reached. As a result, the algorithm will perform better the smaller m is. Yet, it has to be large enough to allow for sufficient backtracking. This can be captured with the following relation

$$\epsilon = \beta^m \eta$$

where m is the maximum number of backtracking steps, η is the initial step size, β is the backtracking factor, and $\epsilon \sim 10^{-16}$ is double precision. This yields

$$m(\beta) = \frac{\log \epsilon - \log \eta}{\log \beta}$$

which is how we determined m as a function of β. The following is a table of results of the performance of the gradient descent with different choices of β. η was always chosen to be the magnitude of the gradient step before the backtracking. Here s is computed backtracking constant, $\mathbb{E}(\cdot)$ is the empirical mean of "$\cdot$".

β	m	t	$\mathbb{E}(backsteps)$	$\mathbb{E}(s)$	$\max s$
.1	16	3.5×10^{-4}	9	1.0×10^{-10}	1.0
.2	23	2.1×10^{-4}	11.5	9.1×10^{-10}	.5
.5	53	4.9×10^{-4}	26.5	1.4×10^{-9}	.2
.8	165	1.4×10^{-3}	82.5	1.9×10^{-9}	.125

Table 11.1

Impact of β parameter on performance of Gradient Descent.

11.3. Online NMF Algorithm. We experimented with different values of β (the number of iterations after which W is updated) to determine its effect on the performance of the Online NMF Algorithm. In table 11.2, t is the time until the stopping criterion is met, $d_{IS}(V, WH)$ the IS-divergence of the last iteration. We can see that as the value of β increases, the W is updated more frequently and the time t increases but the IS divergence decreases. We decided to use $beta = 2^8$ as it is a preferable trade off between performance and time.

β	$t[sec]$	$d_{IS}(V, WH)$
2^6	3.9×10^0	3.8×10^9
2^7	7.6×10^0	7.4×10^8
2^8	1.4×10^1	2.5×10^6
2^9	2.0×10^1	2.4×10^6
2^{10}	3.9×10^1	2.2×10^6
2^{11}	9.1×10^1	2.3×10^6

Table 11.2

Impact of β parameter on performance of NMF.

Smoothing: When we tested the effects of the smoothing term, introduced in section we found that no smoothing in the traditional sense occurred. This can be seen in figure 11.1.

While unfortunate, this is not a big surprise, as the term as defined above only considers the immediately adjacent columns of the column to be updated. For a real smoothing effect

to occur, one would have to include more terms to smooth the columns H over a bigger range. Still, figure 11.1 shows that the values of a row of H that are very big in magnitude decrease

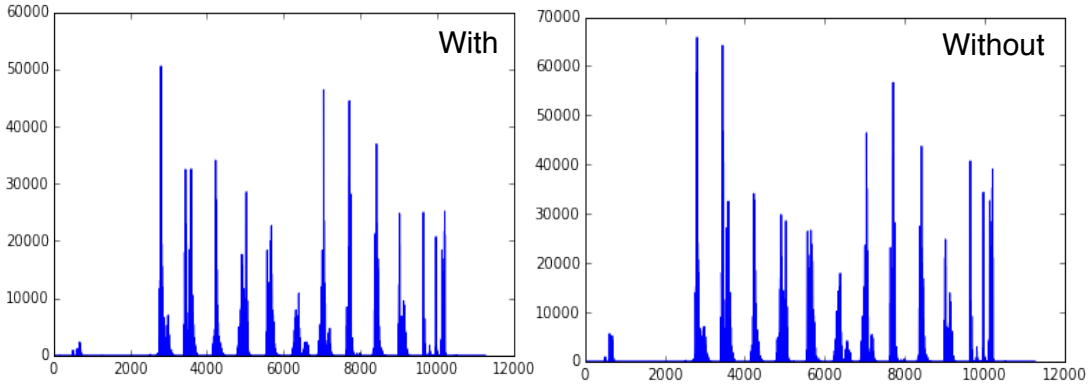

Figure 11.1. *A row of matrix H with and without smoothing.*

in magnitude once the smoothing term is applied. This exemplifies the effect of this term and is a proof of concept. For more effective smoothing, more columns of H would have to be taken into consideration.

11.4. Initializating W. K Means Initialization: We experimented with different values of batch size b to determine its effect on the K-Means objective function and the time taken. As the batch size increases, the distortion goes down but the time taken goes up. This is expected because as the batch size increases up to the size of the test dataset, the algorithm becomes a simple k-means algorithm. We chose a batch-size of 100 for our k-means initialization.

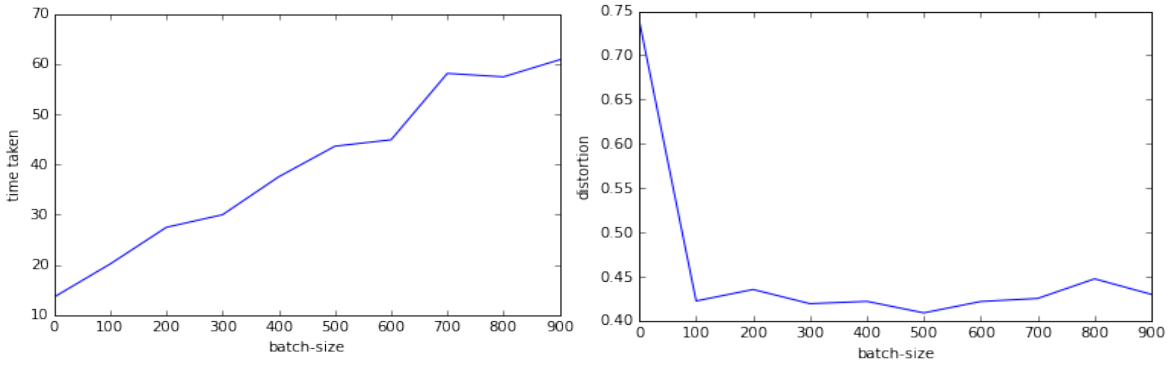

Figure 11.2. *Batch size v/s time and batch size v/s k means objective function.*

As we can see in Figure 11.3, even though the K-Means initialization of W did not show drastic improvements in the initial value of IS divergence, it helped the algorithm to converge faster and improved the performance of the algorithm overall.

Choosing columns of Power Spectrogram: We also experimented with choosing random columns of the power spectrogram to initialize W. The original power spectrogram (of "Mary had a little lamb") and the spectrogram obtained after multiplying W and H matrices (output from online NMF algorithm) can be seen in figure 11.4.

The approximation of the spectrograms was successful but the individual components did not lead to a meaningful source separation (as seen in figure 11.5).

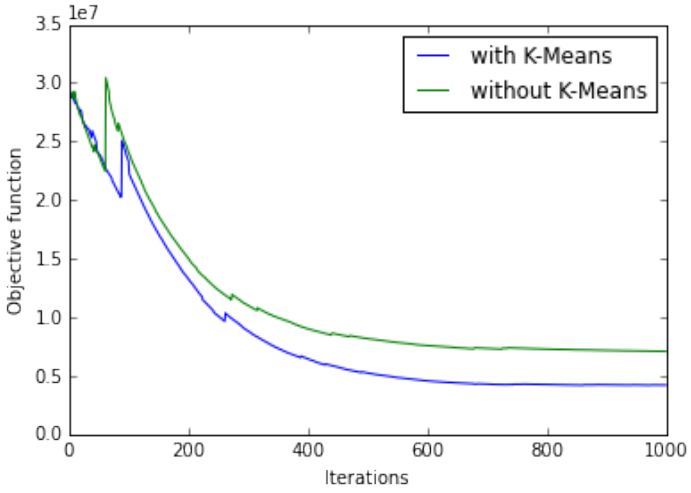

Figure 11.3. *IS Divergence with K-Means initialization.*

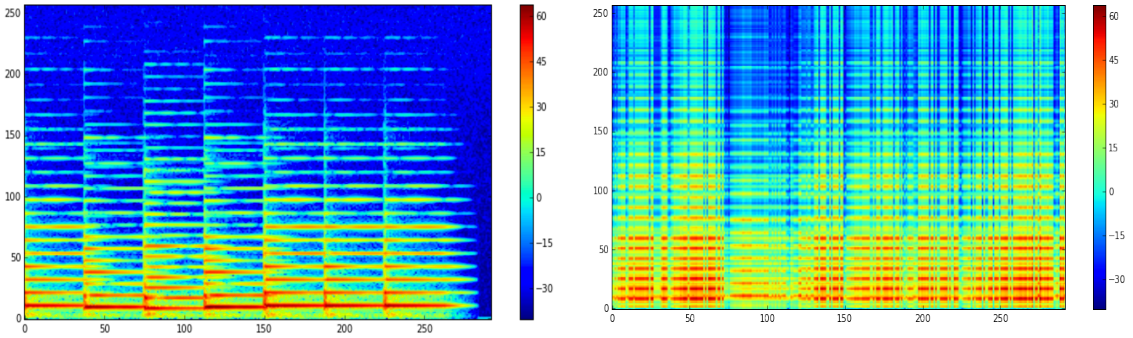

Figure 11.4. *Original power spectrogram and spectrogram obtained after running online NMF.*

12. Conclusions. First, our implementation of the online IS-NMF exhibits higher performance compared to batch IS-NMF, as measured by the time it takes to reach a certain objective function value (in Online NMF). However, while similar divergence values were achieved, meaningful separation was not observed with the online algorithm alone. As a result, we gave the return values W and H of the online algorithm as input to a batch implementation, given by [2], and ran it for 100 iterations. Given our initialization, the divergence values stabilized quickly, so further iterations did not lead to improvements.

In the case of the Jazz recordings, this proved to be more successful than the online algorithm alone. However, for the recording of both our voices, the code of the batch implementation failed because the singular behavior of the IS-divergence at $(x, 0)$. Following the practical considerations of [1], we changed the code of [2] to make the objective function well defined everywhere. After this fix, the quality of the output is improved. Some output files still remain silent, which is indicative of other numerical instabilities of the algorithm.

Secondly, we observed that the initialization of W in the online NMF with the centroids of a precomputed k-means clustering is an improvement over random initialization of W. In addition, we proposed a new objective function for the online IS-NMF algorithm, which includes a locally acting smoothness term. The motivation behind this is that the smoothness of the columns of H improves the quality of the resulting solutions greatly, as shown by [2].

However, as smoothness is by definition not a point-wise property, we plan on further improving on this idea by letting the smoothness term take more columns of H into consideration. To make this idea efficient as well, updating multiple adjacent columns of H at the same time is preferable. Therefore, a logical extension of this online algorithm is a mini-batch NMF algorithm. This would make the inclusion of a smoothness term, analogous to the one

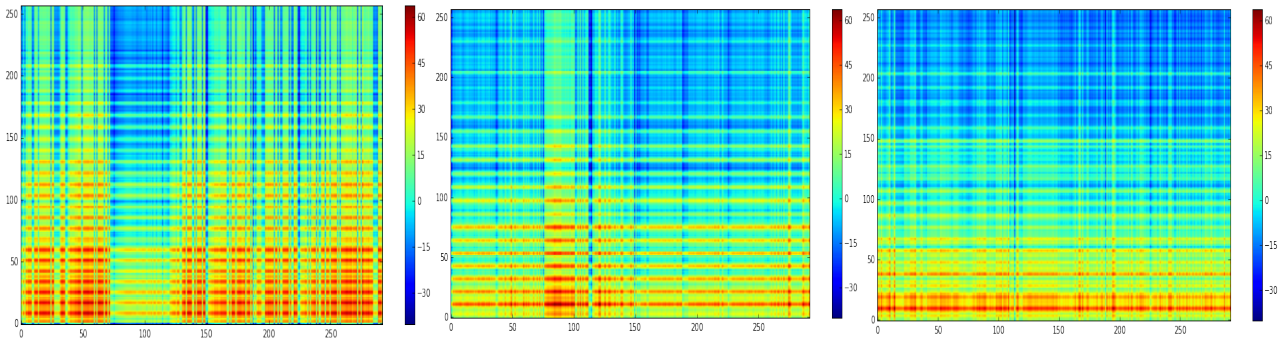

Figure 11.5. *Individual components of power spectrogram*

presented here, more effective.

REFERENCES

[1] Augustin Lefevre, Francis Bach, Cedric Fevotte, *Online algorithms for Nonnegative Matrix Factorization with the Itakura-Saito divergence*,2011..

[2] Cedric Fevotte, *Majorization-Minimization Algorithm for Smooth Itakura-Saito Nonnegative Matrix Factorization*, CNRS LTCI; Telecom ParisTech Paris, France.

[3] D. Sculley, *Web-Scale K-Means Clustering*,Google, Inc. Pittsburgh. PA USA.

[4] Sound eXchange http://sox.sourceforge.net/

[5] pyFASST https://pypi.python.org/pypi/pyFASST