openBliSSART

User Manual

Felix Weninger[1]
TU München

Alexander Lehmann[2]
TU München

Björn Schuller[3]
TU München

Version 1.2, May 2010

[1]weninger@tum.de
[2]lehmanna@in.tum.de
[3]schuller@tum.de

Contents

Chapter 1

Overview

openBliSSART is a framework and toolbox for Blind Source Separation for Audio Recognition Tasks. Main features include

- Component separation using non-negative matrix factorization (NMF) [1, 2, 3] and non-negative matrix deconvolution (NMD) [4]
- Component classification:
  - Feature extraction from components
  - Creation of response variables assigning audio components to classes
  - Assembly of audio files for different classes, such as in drum beat separation [5]
- Supervised and unsupervised NMF feature extraction
- Data export (ARFF [6] and HTK [7] formats)

> In many places in this document and the applications, NMF and NMD are used as synonyms. The reason is that mathematically NMF is a special case of NMD.

The remainder of this manual is divided into three chapters. Chapter 2 provides a brief introductory tutorial on how to use openBliSSART for typical blind source separation tasks. Chapter 3 explains the data storage architecture and algorithmic concepts of openBliSSART in detail. The manual is concluded by a detailed description of the openBliSSART toolbox, its command line parameters and configuration options in Chapter 4.

For detailed information about how to use the classes in the openBliSSART framework, please consult the HTML or LaTeX documentation in the `doc` directory of the openBliSSART source distribution, which can be created using the `doxygen` utility.

Chapter 2

Tutorial

This tutorial provides a brief introduction to the main features of openBliSSART. First, we will describe basic source separation that results in an audio file for each component. Second, we will move towards supervised component classification using a data set, separating audio files into signals corresponding to classes, like music and speech.

2.1 Basic Source Separation

In this section, we will explain the basic steps needed for non-negative matrix factorization (NMF)-based source separation. You will need some music files, preferably short segments ($\approx 10\,\mathrm{s}$) in WAV format. A good choice is to use the WAV files from the `demo/wav` directory in the openBliSSART source distribution, for example. Upon completion of this section, you will be able to extract and listen to the components generated by NMF, and synthesize WAV files for them.

In the first step, we will use the "Browser" GUI application which can be found in the `bin` directory of the openBliSSART installation tree.

Upon starting the browser, you will notice a tree view on the left hand side which at the first start contains only four entries (nodes), namely "Classification objects", "Labels", "Processes" and "Responses". For the purpose of this section, only the "Classification objects" will be relevant.

The right hand side of the browser window is used to display and edit the objects you have selected in the tree view.

2.1.1 Separation with the Browser

Probably the easiest way to use NMF is via the "Import audio" dialog of the Browser which can be accessed using the corresponding button on the bottom of the left side panel.

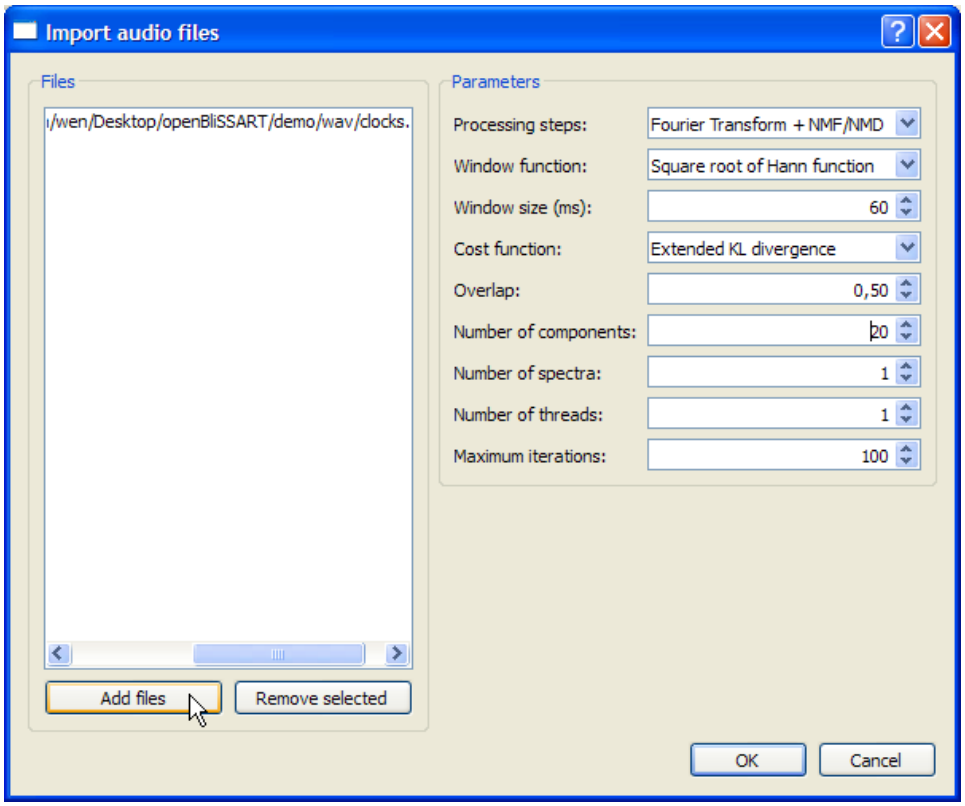

Figure 2.1: "Import audio" dialog

Click "Add files", then select an audio file from the `demo/wav` folder of the openBliSSART source distribution. For once, use the parameters as shown in Figure 2.1. A progress window as shown in Figure 2.3 should appear. The separation process can take several seconds, depending on your hardware.

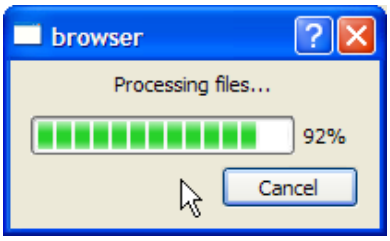

Figure 2.2: Progress display when importing audio

Once the separation process has finished, several items under the "Classification objects" node in the browser tree view should have been generated. Click one of them, and it will be synthesized into an audio signal which you can play back using the buttons in the right part of the window.

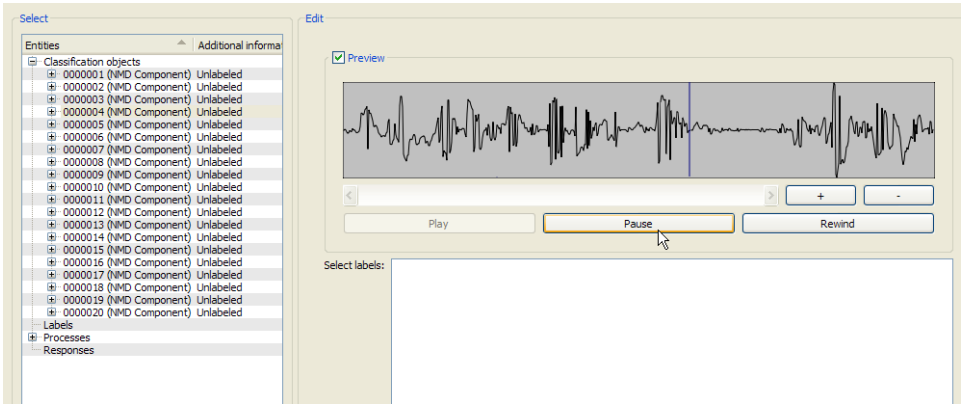

Figure 2.3: Component playback in the browser

You can also export the components as audio signals in the WAV format. To this end, select all of the components (click the first, then Shift-click the last), right-click, and a context menu as in Figure 2.4 will appear.

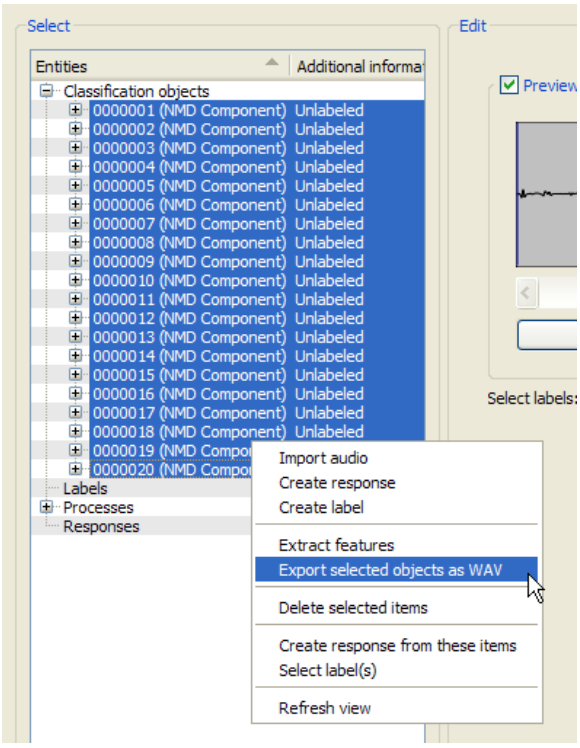

Figure 2.4: Exporting components as WAV files

Select the “Export selected objects as WAV” item, and in the appearing dialog choose a directory where you want to create the WAV files.

The next step of this tutorial will show how you can mix these components together using the free audio editor Audacity, and manually subtract some components. You can skip this part if you do not have, and do not want to install Audacity, and move to the "Command line separation" section below.

2.1.2 Manual Component Mixing

Start Audacity, and select "Import audio" from the "Project" menu. Select all of the WAV files that you exported from the Browser in the previous step. The 20 components should appear as signals below each other in the Audacity window, as shown in Figure 2.5.

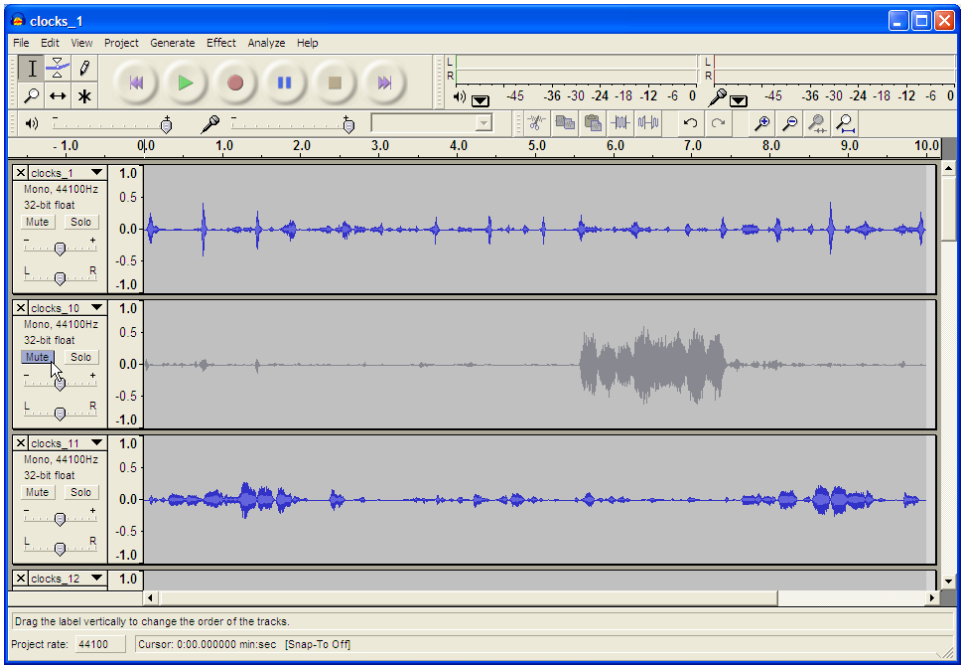

Figure 2.5: Mixing components in Audacity

First, listen to the mix of all components. Depending on the type of music, there are probably hearable artefacts, resulting from the information reduction performed by NMF for separation.

By using the "Mute" and "Solo" buttons, you can mute some of the components, or mute all other components, respectively. Try to identify components which represent drum sounds using the "Solo" button. Normally, this is quite easy as they show a high degree of periodicity. Now mute the identified drum components, and listen to the result.

2.1.3 Command Line Separation

An alternative to the browser is the `septool` (Separation Tool) command line application, which is more flexible, and has more separation features than the browser. The separation process that you performed using the "Import audio" dialog can be realized with `septool` as follows. Open a command line window, change to the `bin` directory within the openBliSSART installation directory, and type

```
septool <file.wav>
```

The default options correspond to the parameters shown in Figure 2.1. After executing this command, open the browser again. There should now be 40 Classification Objects listed (20 from the recent `septool` process, and 20 from the previous separation using the browser). Note that if you left the browser open while running the `septool`, you have to refresh the view using the F5 key.

The `septool` also has the feature to directly save the separated components as WAV files. Open a command line window, change to the openBliSSART installation directory, and type

```
septool -v -p <file.wav>
```

The `-v` option tells the tool not to write to the database (hence the components will not be visible in the browser), and the `-p` option causes the components to be exported as WAV files. Change to the directory where your input WAV file resides. There should now be files named `file_00.wav`, ..., `file_19.wav` corresponding to the 20 components. You can use them for the mixing process as described above.

As an exercise, you can repeat the separation and mixing procedure using different parameters. For once, try the "Squared Euclidean distance" cost function that is available in the "Import audio dialog" (instead of the default "Extended KL divergence"). You can also choose other values for window size, overlap, window function, etc.

The above `septool` command can be adjusted to select squared Euclidean distance as cost function, and to use a window size of 40 ms with the following options:

```
septool --cost-function=ed -s40 -v -p <file.wav>
```

You can also try different numbers of components (in the "Input audio" dialog of the browser as well using the `-c<number>` option of the `septool`). **Congratulations, you have finished the first part of openBliSSART's tutorial!**

2.2 Supervised Component Classification

In this section, we will consider supervised component classification. This is basically the procedure you did above, but instead of manually mixing the tracks, a classifier is used that assigns each component automatically. This is exactly what the openBliSSART demonstrator for drum beat separation does – check it out (in the `demo` subdirectory of the openBliSSART distribution) if you have not yet done so!

In this tutorial, instead of drum beat separation, we will now use the scenario of speech and music discrimination, assuming that you have recordings available that correspond to each of these classes.

In the first step, we will create a data set containing components from speech and music signals. For this purpose, we will again use the "Browser" GUI. Upon completion of this section, you will know what the background of the "Labels" and "Responses" is, and where "Classification objects" got their name.

2.2.1 Importing Audio Files

To start with, we will now import audio files and separate them into components using NMF.

Simply click the "Import audio" button in the lower left corner of the browser window so that the corresponding "Import audio" dialog (figure 2.6) appears.

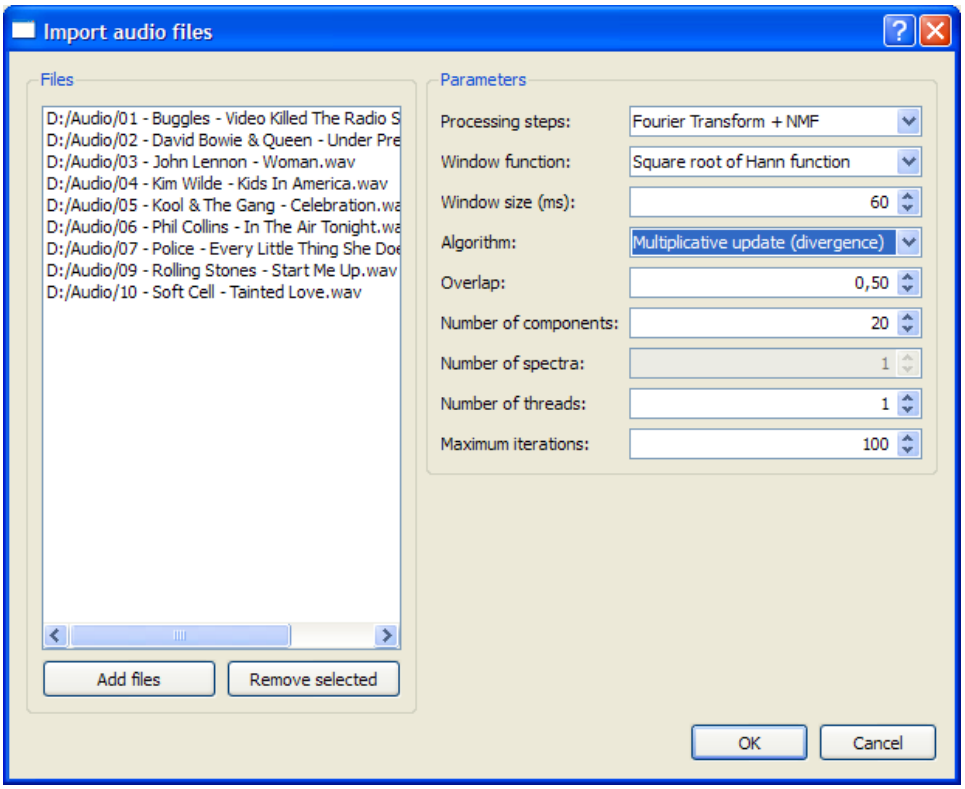

Figure 2.6: "Import audio" dialog

Ensure that the parameters on the right hand side are set exactly as in figure 2.6 and select some audio files (WAV or MP3) containing *music*, preferably around 10-20 seconds long. Then click "Ok" and wait for the process to finish. Depending on the number and length of your audio files, this process may take several minutes as it is computationally intensive. In order to increase performance on multicore systems, you can adapt the "Number of threads" settings to reflect the number of available cores before actually starting the process.

Once the process has completed, you can expand the "Classification objects" node in the tree view so as to examine the entries reflecting the separated components. The second column states that they are still "Unlabeled" – we will take care of that in the next step.

However, at first please repeat the above procedure while this time selecting audio files containing *speech*. Make sure to remember how many audio files of each class (speech and music) you have imported as this will simplify the next step.

2.2.2 Defining Classes

Having imported the neccessary audio files, we will now define the two classes "Speech" and "Music" by creating two corresponding labels. Click the "Create label" button in the lower left corner of the browser window. A new label entry will be inserted under the "Labels" node of the tree view with its text defaulting to the current date and time. Use the textfield on the right hand side to change the text to something more meaningful (like "Music"), then hit the "Save" button. Repeat this step for the "Speech" label. The "Labels" node should now look like in figure 2.7.

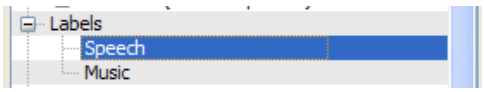

Figure 2.7: Two defined labels

Next, we assign these labels to the separated components which we just have created. Try and select a component in the tree view, and a preview as well as a list of our labels will then appear on the right hand side (see figure 2.8).

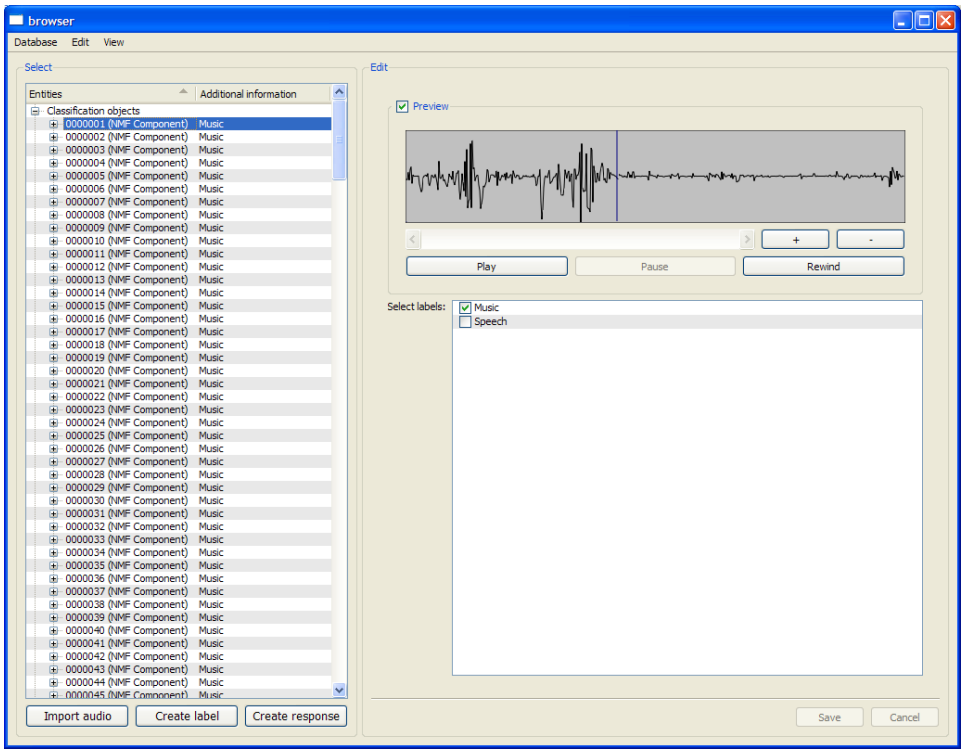

Figure 2.8: View of a classification object (NMF component)

Should you want to listen to the selected component, for example to inspect the results of the NMF procedure, make sure that the "Preview" checkbox is enabled. Once the preview is available, you can listen to the component, move around, and zoom in and out within the respective signal data by using the corresponding buttons inside the preview area.

While it is possible to assign labels to each component individually using the checkboxes on the right hand side, for our scenario it is much more convenient to select all components that were created from music files (remember how many it were?), then right-click to open the context menu and use the "Select label(s)" item (see figure 2.9).

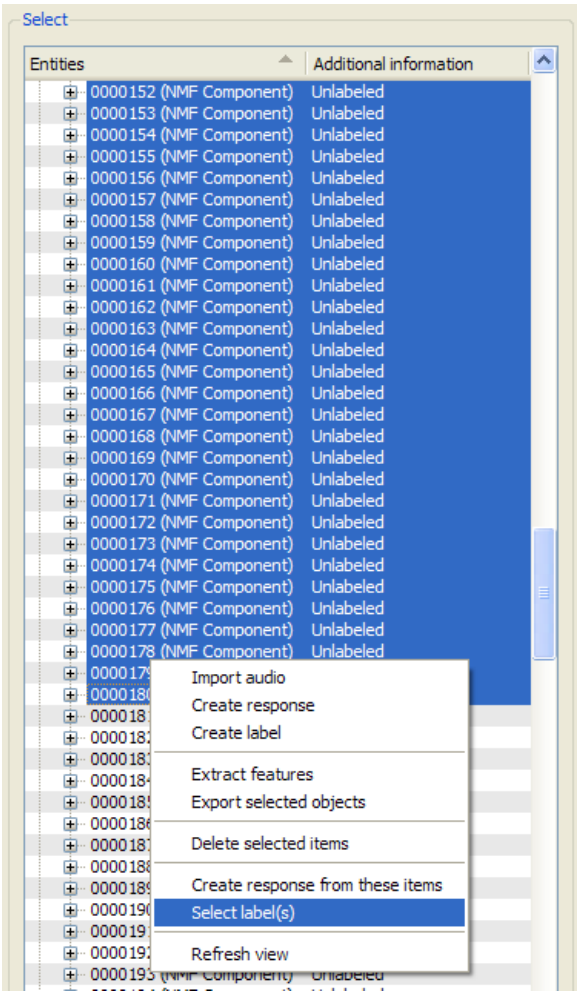

Figure 2.9: Activating the context menu for components

A dialog will appear that allows to add one or more labels to all selected components at the same time. Select “Music” and click “Ok”, then wait for the operation to finish. Upon completion, all selected components should show the label “Music” instead of “Unlabeled” in the second column. By the way, you can always refresh the tree view by either pressing F5 or selecting “Refresh view” from the application’s “View” menu.

Repeat the above procedure for the remaining components, yet this time assign the label “Speech”.

2.2.3 Feature Extraction

The next step towards creating a data set is to extract features from the created components. Again, this is very simple: Just select “Extract features from all data descriptors” from the application’s “Database” menu. Another dialog will appear, prompting you for the number of feature extraction tasks to start. Remember that if you have a multicore system, you might want to set this number to the number of cores for maximum performance, but usually feature extraction is done quite fast anyway.

After the feature extraction has completed, expand one of the classification object nodes and in turn also the “Data descriptors” node inside. Three entries will appear: “Gains”, “Phase Matrix” and “Spectrum”. The phase matrix is only used for conversion of components to wave files, features are extracted from either one of the other two elements. Open the nodes “Gains” or “Spectrum”. A list of features with values will be displayed (see figure 2.10). The numbers inside the parentheses are feature parameters (such as the MFCC index). The meanings of the features are discussed in [8].

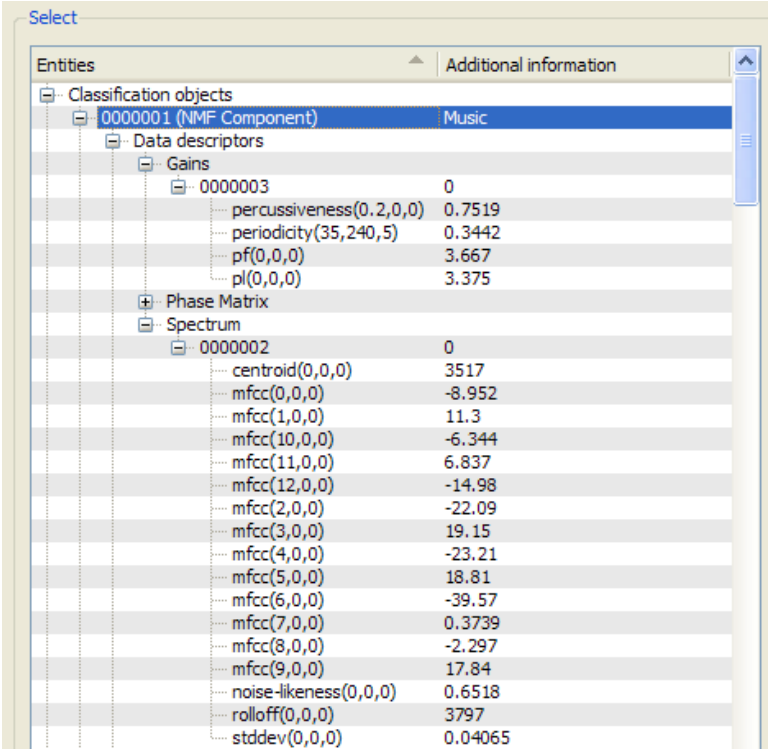

Figure 2.10: Feature subtree of a classification object

2.2.4 Defining a Response

Eventually we will have to feed the extracted features to a support vector machine (SVM). To this end, we create a response variable from all components we have in the database.

Click the "Create response" button in the lower left corner of the browser window. A response entry will be created under the "Responses" node of the tree view. Like it was the case for labels, the name of the new response defaults to the current date and time. Use the textfield on the right hand side to change it into something more meaningful, like for example "Speech vs. music" (see igure 2.11).

Then click the "Add CLOs by label" button and select both labels ("Music" and "Speech") in the corresponding dialog. After clicking "Ok", your response should look like in figure 2.11.

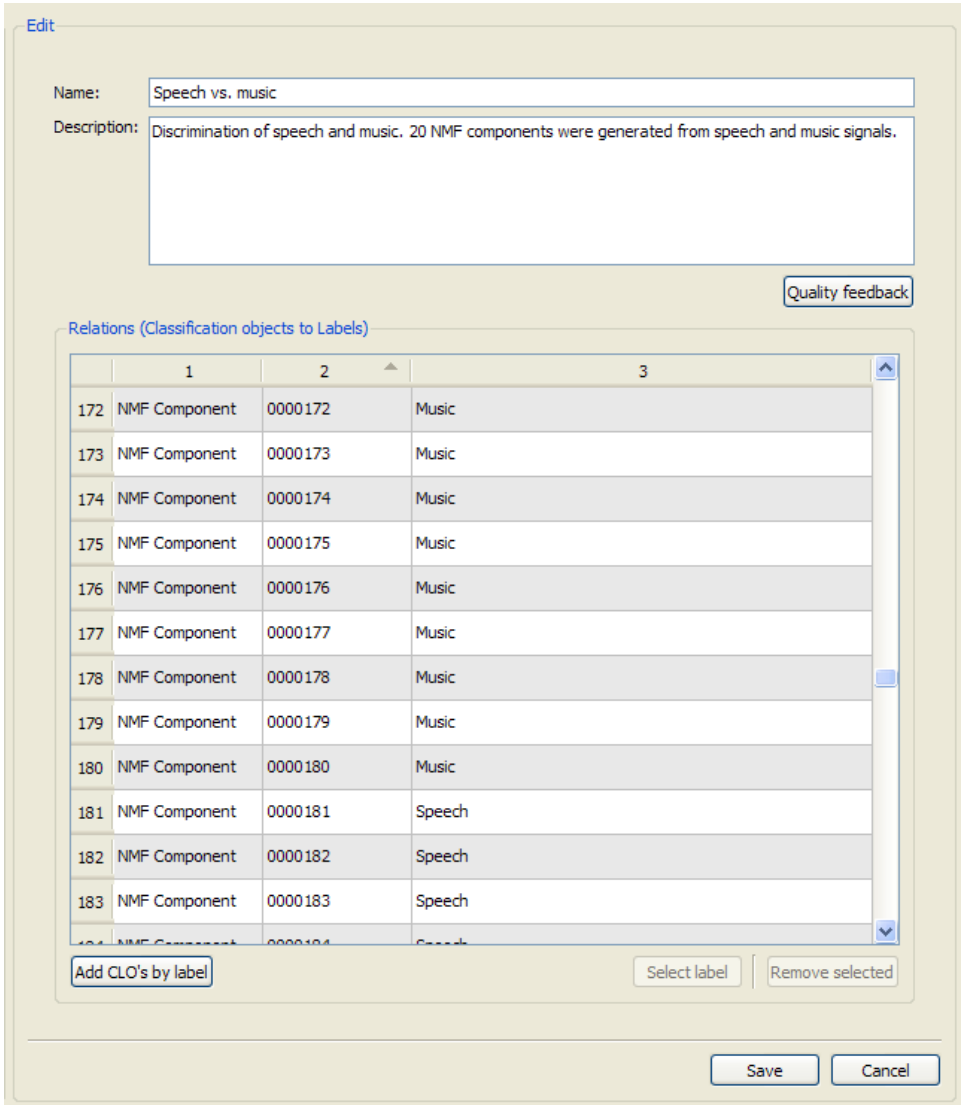

Figure 2.11: Editing a response

2.2.5 Cross-Validation

To assess the quality of the response we have just defined, we might perform a stratified 10-fold cross validation. Currently this function is not accessible from the browser, but is available through a separate tool (`cvtool`).

Open a shell (or Windows command prompt), change to the `bin` directory of the openBliSSART installation tree and type

```
cvtool -r1
```

assuming the response has the ID 1, which is the case if it is the first response you created – otherwise, check the number appearing before the respective response's name in the tree view.
The cross-validation tool should output something like this:

```
Validated 320 samples with 10-fold cross validation.

Confusion matrix

       predicted
real   Music  Speech
Music  179    1
Speech 0      140

Accuracy = 0.996875
Recalls:
Music  0.994444
Speech 1
Mean recall: 0.997222
```

2.2.6 Using a Response for Blind Source Separation

Finally, we are now able to separate audio files into their music and speech parts by means of the response that we have created.

For this purpose, we also use a command-line tool (`septool`). We want the separation tool to perform NMF into 20 components using a window size of 60 ms, then classify the components by a support vector machine trained on the response we have defined in the previous steps, and eventually create audio files by summing up all components for each class and transforming them back into the time domain, i.e. re-synthesizing the results into an appropriate number of files depending on the number of distinct classes that the given response uses. Thus the command line for an arbitrary input file `file.wav` is as follows:

```
septool -c20 -s60 -l1 -v file.wav
```

Again, it is assumed that our response has ID 1. The `-v` ("volatile") option has been added here because we do not want to store additional components from the given input file `file.wav` into the database.

The result of this procedure will be two wave files, namely `file_Speech.wav` and `file_Music.wav`. Of course, you can replace `file.wav` by any suitable WAV or MP3 file. Try mixing speech and music

together and then separating them using the separation tool like described above.

Congratulations, you have just finished openBliSSART's introductory tutorial. For an in-depth discussion of openBliSSART's features and toolbox, move on to the next sections.

Chapter 3

openBliSSART Internals

3.1 Data Organization

openBliSSART's data storage consists of a SQLite database [9] (in the `db` directory of the installation tree) in conjunction with an archive of binary files (in the `storage` directory). The database stores information about the available objects (such as components generated by the NMF), their features and class labels, while the object data itself is externalized to binary files.

Generally[1], when processing audio files, e.g. by FFT and/or NMF, openBliSSART saves information about the separation process, such as the name of the input file, the number of components, the STFT parameters etc. in a respective *process* entity. Furthermore, the computed objects (such as NMF components) are saved as *classification objects*. Each classification object consists of one or more *data descriptors* which describe data like spectral vectors or phase matrices.

Classification Objects openBliSSART currently creates and handles the following types of classification objects:

NMD component	generated by applying STFT and NMD or NMF to an audio file
Spectrogram	generated by applying STFT to an audio file

Data Descriptors The following types of data descriptors are used:

[1] The separation process can also be run in a "volatile" mode that does not store anything. This is useful for example if the result of a NMF separation should be output as WAV files. See section 4.1 for details.

Magnitude matrix	the magnitude spectrogram of an audio file
Phase matrix	the phase spectrogram of an audio file
Spectrum	a magnitude spectrum, generated by NMF or NMD from an audio file; a vector in case of NMF, or a matrix in case of NMD
Gains	a gains vector, generated by NMF or NMD from an audio file

Note that it is perfectly valid for a data descriptor to occur in relation to more than one classification object. For example, each classification object generated by a NMF process contains a reference to the phase matrix of the original signal so as to be able to re-synthesize wave files from one or more components. The phase matrix, however, is stored only once.

Each data descriptor is associated with a separation process with a unique ID. These IDs can for instance be found out by looking at the process listing in the browser application, and are needed for component feature extraction as well as data export.

Features, Responses and Labels Data descriptors relate to *features* which are used during classification. A *response* assigns classification objects to *labels*. Classification is done using features from the data descriptors that make up the classification objects in the response.

The browser (4.7) can be used to conveniently explore the database structure.

3.1.1 Database entities

A graphical overview over database entities and their relations (entity-relationship diagram) is given by figure 3.1.

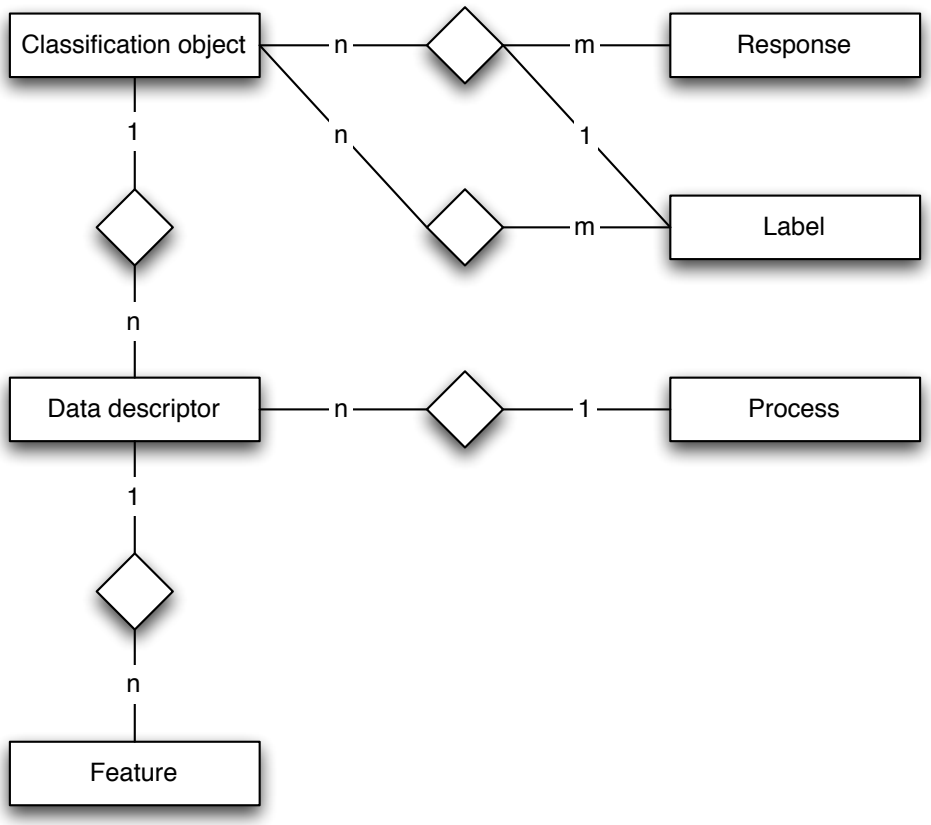

Figure 3.1: Entity-relationship diagram of our database scheme

Processes A *process* creates objects by processing an audio file. It has the following attributes:

- Process ID
- Name
- Input file name
- Sample frequency of the input file
- Time at which the process was started

Furthermore, each process can have an arbitrary number of named parameters, where the parameter value can be of any data type.

Data descriptors A *data descriptor* contains information ("meta-data") about a data object, such as a vector or a matrix, which is stored as a file. The data descriptor entity has the following attributes:

- Data descriptor ID
- ID of the process that created the data object
- Type annotation, in our system one of "Gains vector", "Spectral vector" or "Phase matrix"
- Index (in our NMF case, the component index for gains and spectral vectors, zero for phase matrices)
- Data availability flag. Functions that need the binary data ignore data descriptors whose data availability flag is false. This makes it possible to migrate the database to another computer without copying all the externalized binary data, in a consistent way.

Besides the data descriptor ID, the triple (process ID, type annotation, index) uniquely identifies a data descriptor.

Classification objects A *classification object* consists of several data objects described by data descriptors. For example, in our application, we want to classify components generated by a NMF process, which consist of a gains vector and a spectral vector.

A classification object has a unique ID and a type annotation (in our case, the only possible annotation is "NMF component"); and furthermore, a list of IDs of data descriptors that make up the classification object.

Finally, for each classification object a preselection of possible class labels is stored. For example, a drum component could be labelled with "Drum" or, more specifically, with "Snare drum".

Classification objects are subject to the following constraints:

- All data descriptors that make up the object must be created by the same process.
- Every type of data descriptor (determined by the "type annotation" attribute) may occur at most once.

Features A *feature* is a named value assigned to a data object, for example, a cepstral coefficient of a spectral vector. Thus, the following attributes are required:

- ID of the data descriptor describing the data object
- Feature name (e.g. "MFCC")
- Feature parameter (e.g. the coefficient index in the MFCC case)
- Feature value

Every feature of a data object can be uniquely identified by feature name and parameter.

Labels A *label* is a textual class label that can be assigned to classification objects. In our case, we could define the labels "Drum", "Harmonic" or more specific labels like "Guitar" or "Snare drum".

Responses A *response* is an assignment of classification objects to labels. Additionally, every response has a response ID, a name (e.g. "Drum vs. Harmonic") and a textual description.

3.1.2 Storage of binary files

Binary files corresponding to data objects, i. e. vectors and matrices, are stored in a directory layout such that the file name can be uniquely determined by the attributes of the corresponding data descriptor.

All multi-byte values are saved in little-endian order.

Our binary file format for vectors consists of the following elements:

- Orientation header (0 = row vector, 1 = column vector) (as 32 bit `unsigned int`)
- Vector dimension (32 bit `unsigned int`)
- Array of components (64 bit `double`)

Our binary file format for matrices consists of the following elements:

- Matrix header = 2 (as 32 bit `unsigned int`)
- Number of rows (32 bit `unsigned int`)
- Number of columns (32 bit `unsigned int`)
- Array of matrix entries (64 bit `double`) in column-major order, i.e. entry $a_{i,j}$ of a matrix with m rows is stored at position $j * m + i$

3.2 Source separation by NMF

Non-Negative Matrix Factorization (NMF) is an algorithm originally proposed for image decomposition [1]. As a method of information reduction, its most promiment feature is the usage of non-negativity constraints: unlike other methods such as Principal Components Analysis, it achieves a parts-based representation where only additive – never subtractive – combinations of the Given a matrix $\mathbf{V} \in \mathbb{R}_+^{m \times n}$ and a constant $r \in \mathbb{N}$, non-negative matrix factorization (NMF) computes two matrices $\mathbf{W} \in \mathbb{R}_+^{m \times r}$ and $\mathbf{H} \in \mathbb{R}_+^{r \times n}$, such that

$$\mathbf{V} \approx \mathbf{WH} \tag{3.1}$$

openBliSSART applies NMF in the frequency domain, by factorizing magnitude spectrogram matrices obtained by short-time Fourier transformation (STFT). Thereby the signal is split into overlapping frames of constant size. In speech processing, it is common to use a frame size of 25 ms and an overlap of 60 %, corresponding to a *frame rate* of 10 ms. Each frame is multiplied by a window function and transformed to the frequency domain using Discrete Fourier Transformation (DFT), with transformation size equal to the number of samples in each frame.

First, openBliSSART provides the Hamming function for windowing

$$h(k) = 0.54 - 0.46 \cos\left(\frac{2\pi k}{T-1}\right) \tag{3.2}$$

where T is the frame size and $k \in \{0, \ldots, T\}$.

Other window functions are the Hann(ing) function:

$$h(k) = 0.5 - 0.5 \cos\left(\frac{2\pi k}{T-1}\right) \tag{3.3}$$

and its square root, which can be used for reducing artefacts resulting from the transformation.

Only the magnitudes of the DFT coefficients are retained, and the frame spectra are put in the columns of a matrix. Denoting the number of frames by n and the frame size by T, and considering the symmetry of the coefficients, this yields a $(\lfloor T/2 \rfloor + 1) \times n$ real matrix.

The crucial idea behind NMF-based blind source separation is to assume a *linear signal model*. Note that Eq. 3.1 can be written as follows (the subscripts $:, t$ and $:, j$ denotes the t^{th} and j^{th} matrix columns, respectively):

$$\mathbf{V}_{:,t} \approx \sum_{j=1}^{r} \mathbf{H}_{j,t} \mathbf{W}_{:,j}, \quad 1 \leq t \leq n \tag{3.4}$$

Thus, if $\mathbf{V}$ is the magnitude spectrogram of a signal (with short-time spectra in columns), the factorization from Eq. 3.1 represents each short-time spectrum $\mathbf{V}_{:,t}$ as a linear combination of spectral basis vectors $\mathbf{W}_{:,j}$ with non-negative coefficients $\mathbf{H}_{j,t}$ $(1 \leq j \leq r)$.

When there is no prior knowledge about the number of spectra that can describe the source signal, the number of components r has to be chosen empirically, depending on the application.

3.2.1 Basic NMF Algorithms

A factorization according to Eq. 3.1 is usually achieved by iterative minimization of a cost function $c(\mathbf{W}, \mathbf{H})$:

$$(\mathbf{W}, \mathbf{H}) = \arg\min_{\mathbf{W},\mathbf{H}} c(\mathbf{W}, \mathbf{H}) \tag{3.5}$$

In fact, many variants of NMF only differ by their choice of a particular cost function. The core of these functions is a measurement of the reconstruction error between the original matrix and the product of the NMF factors. Thus, a basic cost function is the squared Euclidean distance between $\mathbf{V}$ and $\mathbf{WH}$:

$$c_e(\mathbf{W},\mathbf{H}) = ||\mathbf{V}-\mathbf{WH}||_F = \sum_{i=1}^{n}\sum_{j=1}^{m}(\mathbf{V}-\mathbf{WH})_{i,j}^2, \tag{3.6}$$

where $||.||_F$ denotes the Frobenius norm.

Another cost function consists of a modified version of Kullback-Leibler (KL) divergence:

$$c_d(\mathbf{W},\mathbf{H}) = \sum_{i=1}^{m}\sum_{t=1}^{n}\left(\mathbf{V}_{i,t}\log\frac{\mathbf{V}_{i,t}}{(\mathbf{WH})_{i,t}} - (\mathbf{V}-\mathbf{WH})_{i,t}\right) \tag{3.7}$$

For minimization of either cost function, openBliSSART implements the two algorithms by Lee and Seung [3], which iteratively modify $\mathbf{W}$ and $\mathbf{H}$ using 'multiplicative update' rules. It can be shown that c_e is non-increasing under the update rules

$$\mathbf{H}_{j,t} \leftarrow \mathbf{H}_{j,t}\frac{(\mathbf{W}^T\mathbf{V})_{j,t}}{((\mathbf{W}^T\mathbf{W})\mathbf{H})_{j,t}} \qquad j=1,\ldots,r; t=1,\ldots,n \tag{3.8}$$

$$\mathbf{W}_{i,j} \leftarrow \mathbf{W}_{i,j}\frac{(\mathbf{V}\mathbf{H}^T)_{i,j}}{(\mathbf{W}(\mathbf{H}\mathbf{H}^T))_{i,j}} \qquad i=1,\ldots,m; j=1,\ldots,r \tag{3.9}$$

and that c_d is non-increasing under the update rules

$$\mathbf{H}_{j,t} \leftarrow \mathbf{H}_{j,t}\frac{(\mathbf{W}^T(\mathbf{V}./\mathbf{WH}))_{j,t}}{(\mathbf{W}^T\mathbf{1})_{j,t}} \qquad j=1,\ldots,r; t=1,\ldots,n \tag{3.10}$$

$$\mathbf{W}_{i,j} \leftarrow \mathbf{W}_{i,j}\frac{((\mathbf{V}./(\mathbf{WH}))\mathbf{H}^T)_{i,j}}{(\mathbf{1}\mathbf{H}^T)_{i,j}} \qquad i=1,\ldots,m; j=1,\ldots,r. \tag{3.11}$$

where $\mathbf{1}$ is an all-unity matrix and ./ indicates elementwise division. The above matrix formulation has been shown to yield better performance than the scalar product formulations in [3] when using fast implementations of matrix multiplication, as openBliSSART does.

Thereby the denominators are floored to a very small positive constant (such as 10^{-10}) to avoid divisions by zero. Note that these rules are applied alternatingly, with each $\mathbf{W}$ update using the new value of $\mathbf{H}$ that was calculated in the previous $\mathbf{H}$ update and vice versa. Note that the order of calculation, indicated by the parentheses in Eq. 3.8 and Eq. 3.9 can have a great effect on performance due to the different matrix dimensions.

3.2.2 Initialization and Termination

For conventional, i. e. unsupervised NMF, $\mathbf{W}$ and $\mathbf{H}$ can be initialized with the absolute values of random numbers drawn from a Gaussian distribution with $\mu = 0$ and $\sigma = 1$, or from a uniform distribution on the interval $]0, 1]$.

openBliSSART uses the following stopping criterion for NMF:

$$\frac{||\mathbf{W}^{q+1}\mathbf{H}^{q+1} - \mathbf{W}^{q}\mathbf{H}^{q}||_F}{||\mathbf{W}^{q}\mathbf{H}^{q}||_F} < \zeta, \tag{3.12}$$

with $\mathbf{W}^q$ and $\mathbf{H}^q$ denoting the values of $\mathbf{W}$ and $\mathbf{H}$ at iteration q, respectively, and ζ being a small constant. However, evaluation of the criterion 3.12 is costly, as the matrix product $\mathbf{WH}$ has to be computed, and the previous values of $\mathbf{W}$ and $\mathbf{H}$ (or the previous value of their product) have to be stored. Thus, to reduce computational cost, it is preferred to perform a *fixed number* of iterations. Experience shows that 100–200 iterations ensure a small reconstruction error which is not significantly reduced by further iterations.

3.2.3 Supervised Component Classification

In scenarios like speaker separation or drum accompaniment reduction, sources (speakers, drums) can often not be modelled by a single spectrum. NMF-based approaches in this area thus have to use a number r of components which is larger than the number of sources. Consequently, an assignment of the components to sources has to be made.

For the following discussion, we formally define the j^{th} component of the signal to be the pair $(\mathbf{w}_j, \mathbf{h}_j)$ of a spectrum $\mathbf{w}_j := \mathbf{W}_{:,j}$ along with its time-varying gains $\mathbf{h}_j := \mathbf{H}_{j,:}$ (the subscript $j,:$ denotes the j^{th} matrix row).

openBliSSART uses the following approach to decide which components belong to which source. First, a Support Vector Machine (SVM) classifier is trained from the features in a *response variable* according to Section 3.1. After classification, a magnitude spectrogram $\mathbf{V}_{s_i}$ for each source s_i can be computed: let

$$J_{s_i} = \{j : (\mathbf{w}_j, \mathbf{h}_j) \text{ assigned to } s_i\} \tag{3.13}$$

be the set of indices of components assigned to source s_i. Then,

$$\mathbf{V}_{s_i} = \sum_{j \in J_{s_i}} \mathbf{w}_j \mathbf{h}_j. \tag{3.14}$$

$\mathbf{V}_{s_i}$ is transferred back to the time domain using a column-wise inverse IDFT, using the phase matrix from the original signal. Finally, time signals for each source are obtained by adding up the time frames respecting their overlap. Multiplication of the time frames with the square root of the Hann function can reduce the artifacts resulting from the transformation [5].

3.2.4 Source Separation by Supervised NMF

Supervised NMF means that Thereby $\mathbf{W}$ is set to a predefined matrix where each column contains a spectrum corresponding to one of the sources. For example, in speaker separation these spectra can be computed from phonemes uttered by a certain speaker [10].

Then $\mathbf{W}$ is kept constant throughout the iteration whereas $\mathbf{H}$ is initialized randomly and updated iteratively. Time signals for each source can be obtained using the procedure which was mentioned above, setting J_{s_i} (Eq. 3.13) to the indices of columns of $\mathbf{W}$ that were initialized with spectra from source s_i. This paradigm has led to notable results in speech denoising [11, 12] and speaker spearation [10, 13].

3.2.5 Sparse NMF

The aforementioned cost functions measure the reconstruction error c_r. However, for overcomplete bases (i. e. $r > m, n$) sparse NMF [14, 15, 16, 10, 17] can be valuable, whereby a term is added that increases the value of the cost function for each non-zero entry in $\mathbf{H}$, hence 'dense' matrices are penalized. The resulting cost function $c(\mathbf{W}, \mathbf{H})$ is

$$c(\mathbf{W}, \mathbf{H}) = c_r(\mathbf{W}, \mathbf{H}) + \lambda c_s(\mathbf{H}) \tag{3.15}$$

where $c_r(\mathbf{W}, \mathbf{H})$ can – for example – be set to squared Euclidean distance (Eq. 3.6) or modified KL divergence (Eq. 3.7).

First, openBliSSART supports a straightforward approach introduced by [17]:

$$c_s(\mathbf{H}) = \sum_{j=1}^{r} \frac{1}{\sigma_j} \sum_{t=1}^{n} \mathbf{H}_{j,t} \tag{3.16}$$

To prevent the scaling from affecting the value of the cost function, it normalizes the activations of each component j, e. g. by their standard deviation estimates σ_j [17]. The multiplicative update rules for $\mathbf{H}$ minimizing the cost function 3.15 are derived as follows.

The gradient of the cost function is written as a subtraction $\nabla c(\mathbf{W}, \mathbf{H}) = \nabla c^+(\mathbf{W}, \mathbf{H}) - \nabla c^-(\mathbf{W}, \mathbf{H})$ of element-wise nonnegative terms $\nabla c^+(\mathbf{W}, \mathbf{H}) = \nabla c_r^+(\mathbf{W}, \mathbf{H}) + \lambda \nabla c_s^+(\mathbf{H})$ and $\nabla c^-(\mathbf{W}, \mathbf{H}) = \nabla c_r^-(\mathbf{W}, \mathbf{H}) + \lambda \nabla c_s^-(\mathbf{H})$. For Euclidean distance, we have

$$\nabla c_r^+(\mathbf{W}, \mathbf{H}) = \mathbf{W}^T \mathbf{W} \mathbf{H} \tag{3.17}$$

and

$$\nabla c_r^-(\mathbf{W}, \mathbf{H}) = \mathbf{W}^T \mathbf{V} \tag{3.18}$$

For KL divergence, we have:

$$\nabla c_r^+(\mathbf{W}, \mathbf{H}) = \mathbf{W}^T \mathbf{1} \tag{3.19}$$

and

$$\nabla c_r^-(\mathbf{W},\mathbf{H}) = \mathbf{W}^T(\mathbf{V}./.(\mathbf{WH})) \tag{3.20}$$

For the sparseness term, we have:

$$[\nabla c_s^+(\mathbf{H})]_{j,t} = \frac{1/\sqrt{n}}{\sqrt{\sum_{k=1}^n \mathbf{H}_{j,k}^2}} \tag{3.21}$$

and

$$[\nabla c_s^-(\mathbf{H})]_{j,t} = \mathbf{H}_{j,t}\frac{\sqrt{n}\sum_{k=1}^n \mathbf{H}_{j,k}}{(\sum_{k=1}^n \mathbf{H}_{j,k}^2)^{3/2}} \tag{3.22}$$

The final multiplicative update rule is:

$$\mathbf{H}_{j,t} \leftarrow \mathbf{H}_{j,t}\frac{\nabla_c^-(\mathbf{W},\mathbf{H})}{\nabla_c^+(\mathbf{W},\mathbf{H})} \tag{3.23}$$

As a second approach to sparse NMF, openBliSSART implements the algorithm from [16] which is based on a cost function resembling Euclidean distance with a column-wise normalized $\mathbf{W}$ matrix. openBliSSART reformulates the multiplicative update rules for enhanced performance:

$$\mathbf{H}_{j,t} \leftarrow \mathbf{H}_{j,t}\frac{(\mathbf{W}^T\mathbf{V})_{j,t}}{((\mathbf{W}^T\mathbf{W})\mathbf{H})_{j,t} + \lambda} \tag{3.24}$$

where λ is the sparseness weight, and

$$\mathbf{W}_{i,j} \leftarrow \mathbf{W}_{i,j}\frac{(\mathbf{V}\mathbf{H}^T)_{i,j} + ((\mathbf{H}\mathbf{H}^T)(\mathbf{W}^T\mathbf{W}))_{j,j}\hat{\mathbf{W}}_{i,j}}{(\mathbf{W}(\mathbf{H}\mathbf{H}^T))_{i,j} + (\mathbf{H}\mathbf{V}^T\mathbf{W})_{j,j}\hat{\mathbf{W}}_{i,j}} \tag{3.25}$$

where $\hat{\mathbf{W}}$ is the column-wise normalized matrix $\mathbf{W}$ (Euclidean norm).

3.2.6 Convolutive NMF

Convolutive variants of NMF consider spectra that evolve over time. In other words, the acoustic events that build the signal are no longer instantaneous, but rather sequences of observations. In speech processing, these sequences can correspond to phonemes [18] or even whole words [4].

First, openBliSSART supports Non-Negative Matrix Deconvolution, which is based on a convolutive signal model:

$$\mathbf{V} \approx \mathbf{\Lambda} = \sum_{p=0}^{P-1} \mathbf{W}(p)\overrightarrow{\mathbf{H}}^p \tag{3.26}$$

where $\mathbf{W}(p), p \in \{0,\ldots,P-1\}$ is a set of P matrices and $\overrightarrow{(\cdot)}^p$ is a matrix operator that shifts the columns of its argument by p spots to the right, filling the leftmost p columns with zeros. Analogously to Eq. 3.4, this equation can be rewritten as

$$\mathbf{V}_{:,t} \approx \sum_{j=1}^{r} \sum_{p=0}^{\min\{P-1,t\}} \mathbf{H}_{j,t-p} \mathbf{W}(p)_{:,j}, \quad 1 \le t \le n \tag{3.27}$$

where again r is the number of components and n is the number of columns of $\mathbf{V}, n \ge P$. (Note that the inner sum now resembles a convolution.)

It is straightforward to extend the cost functions c_e (Euclidean distance, Eq. 3.6) and c_d (modified KL divergence, Eq. 3.7) to the convolutive signal model:

$$c'_e = ||\mathbf{V} - \mathbf{\Lambda}||_F = \sum_{i=1}^{n} \sum_{j=1}^{m} (\mathbf{V} - \mathbf{\Lambda})_{i,j}^2 \tag{3.28}$$

$$c'_d = \sum_{i=1}^{m} \sum_{t=1}^{n} \left(\mathbf{V}_{i,t} \log \frac{\mathbf{V}_{i,t}}{\mathbf{\Lambda}_{i,t}} - (\mathbf{V} - \mathbf{\Lambda})_{i,t} \right) \tag{3.29}$$

A multiplicative update algorithm can be derived for either cost function [4, 19]. Note that there are now $P+1$ updates in each iteration: one for each matrix $\mathbf{W}(p), p = 0, \ldots, P-1$ and one for $\mathbf{H}$. In detail, the update rules for minimization of c'_e (Eq. 3.28) are given by

$$\mathbf{W}(p)_{i,j} \leftarrow \mathbf{W}(p)_{i,j} \frac{(\mathbf{V} \overset{p\rightarrow}{(\mathbf{H})}^T)_{i,j}}{(\mathbf{\Lambda} \overset{p\rightarrow}{(\mathbf{H})}^T)_{i,j}} \qquad i = 1, \ldots, m; j = 1, \ldots, r \tag{3.30}$$

$$\mathbf{H}_{j,t} \leftarrow \mathbf{H}_{j,t} \frac{1}{P} \sum_{p=0}^{P-1} \frac{(\mathbf{W}(p)^T \overset{\leftarrow p}{\mathbf{V}})_{j,t}}{(\mathbf{W}(p)^T \overset{\leftarrow p}{\mathbf{\Lambda}})_{j,t}} \qquad j = 1, \ldots, r; t = 1, \ldots, n \tag{3.31}$$

while c'_d (Eq. 3.29) is minimized by

$$\mathbf{W}(p)_{i,j} \leftarrow \mathbf{W}(p)_{i,j} \frac{\sum_{t=1}^{n} (\overset{p\rightarrow}{\mathbf{H}})_{j,t} \tilde{\mathbf{V}}_{i,t}}{\sum_{t=1}^{n} (\overset{p\rightarrow}{\mathbf{H}})_{j,t}} \qquad i = 1, \ldots, m; j = 1, \ldots, r \tag{3.32}$$

$$\mathbf{H}_{j,t} \leftarrow \mathbf{H}_{j,t} \frac{1}{P} \sum_{p=0}^{P-1} \frac{\sum_{i=1}^{m} \mathbf{W}(p)_{i,j} (\overset{\leftarrow p}{\tilde{\mathbf{V}}})_{i,t}}{\sum_{i=1}^{m} \mathbf{W}(p)_{i,j}} \qquad j = 1, \ldots, r; t = 1, \ldots, n. \tag{3.33}$$

Thereby $\tilde{\mathbf{V}}$ is the element-wise division of $\mathbf{V}$ and $\mathbf{\Lambda}$, and the $\overset{\leftarrow p}{(\cdot)}$ operator shifts the columns of its argument by p spots to the left, introducing zeros in the rightmost p columns. Furthermore the denominators are floored to a very small positive constant (such as 10^{-10}) to avoid divisions by zero.

Notice that the update rules for $\mathbf{H}$ were both obtained by first deriving an $\mathbf{H}$ update rule that takes into account only one $\mathbf{W}(p)$, then taking the average of these updates for all $p \in \{0, \ldots, P-1\}$.

The value of the approximation $\mathbf{\Lambda}$ must be updated after execution of each update rule, but openBliSSART reduces the computational cost for this step by the formulation introduced in [19]:

$$\mathbf{\Lambda} \leftarrow \mathbf{\Lambda} - \hat{\mathbf{W}}(p) \overset{p\rightarrow}{\mathbf{H}} + \mathbf{W}(p) \overset{p\rightarrow}{\mathbf{H}} \tag{3.34}$$

after update of each $\mathbf{W}(p)$, where $\hat{\mathbf{W}}(p)$ denotes the value of $\mathbf{W}(p)$ before the update.

NMF can be regarded as a special case of NMD: by setting $P = 1$, the convolutive signal model as well as the NMD update rules reduce to the linear signal model and NMF update rules, respectively.

Besides NMD, a 'sliding window' NMF variant [20] is supported by openBliSSART. Here, simply a matrix $\mathbf{V}' \in \mathbb{R}_+^{m \times (Tn)}$ is created from $\mathbf{V}$ by concatenating T subsequent columns of $\mathbf{V}$ into one column of the larger matrix $\mathbf{V}'$. Compared to NMD, this method has the advantage that no special update rules are needed, hence any algorithm for NMF can be immediately exploited.

3.3 Source Separation by ICA

ICA approaches the problem of blind source separation based on the assumption that observed signals can be regarded as linear combinations of independent sources. Hence, the basic ICA model can be expressed in matrix notation as

$$\mathbf{X} = \mathbf{A} \cdot \mathbf{S} \tag{3.35}$$

where $\mathbf{X}$ denotes the observed signals, $\mathbf{A}$ is considered as the *mixing*-matrix and the $\mathbf{S}$ contains the signal sources.
Since both $\mathbf{A}$ and $\mathbf{S}$ are unknown, ICA provides a solution by considering the signals as *independent* random variables and, consequently, the values of the signals at time t as random samples of these variables.

ICA makes use of the Central Limit Theorem in terms of assuming that due to the fact that $\mathbf{X}$ is a linear combination of the sources, $\mathbf{X}$ eventually has a more Gaussian distribution than the original random variables in $\mathbf{S}$. Vice versa, $\mathbf{A}^{-1}$ has to be determined such that it *maximizes the non-gaussianity* of the original random variables in $\mathbf{S}$ in order to retrieve the independent source signals.

The FastICA algorithm implemented by openBliSSART [21] constitutes a good compromise between the properties of both *kurtosis* and *negentropy*. It uses a fast fixed-point algorithm for the following cost function:

$$C(\mathbf{x}) = \frac{1}{a} \log \cosh(a \cdot \mathbf{w}^T \mathbf{x})$$

where a is a real constant within $[1, 2]$ and w is the current *weight*-vector which maximizes projected data's non-gaussianity and hence is constantly updated throughout the FastICA iterations.

Chapter 4

Toolbox

4.1 Separation Tool

The separation tool (`septool`) is the central command-line application of openBliSSART. It takes one or more audio files and separates them into components by using non-negative matrix factorization. Components can be stored and/or classified using an existing response variable.
In the former case, each component is saved to the database as classification object. Also, the parameters of the separation process are saved.
In the case of classification, an audio file is generated for each class.

An arbitrary number of files to be processed (≥ 1) can be given as arguments. WAV, OGG, and FLAC formats are supported.[1] Furthermore, the process can be controlled via a variety of parameters, are listed below.

4.1.1 General

- `-h`, `--help` – display information about command line parameters and exit.
- `-A`, `--echo` – print the base name of the application binary and its named command line options in long format, with their parameters if given, before executing.
- `-C`, `--config=<filename>` – use the specified configuration file (properties format) instead of the default one. See section 4.9 for details.
- `-n<number>`, `--num-threads=<number>` – the number of concurrent threads to use for separation and classification. Should be set to the number of CPUs (cores) present in the computer for maximum performance.

[1] Generally speaking, all audio file formats supported by the SDL_sound library can be read.

- `-S`, `--scripted` – run in "scripted" mode, i.e. assume that the input files contain file names of audio files separated by newlines. This option can be useful if lots of files should be processed, and to ensure compatibility with systems that limit the number of command-line options.

4.1.2 Audio Preprocessing

Transformation options given on the command line override the corresponding configuration options (see 4.9.2).

- `-r<function>`, `--reduce-mids` – subtract right from left channel when converting from a stereo to a mono signal.
- `-k<k>`, `--preemphasis=<k>` – preemphasizes the signal with a factor of k such that for all $t > 0$, $s'_t = s_t - ks_{t-1}$, where s_t and s'_t are the sample values at position t in the original and preemphasized signal, respectively.
- `-d`, `--remove-dc` – subtracts the mean (DC component) from each frame before transformation.

4.1.3 Transformation

Transformation options given on the command line override the corresponding configuration options (see 4.9.3).

- `-w<function>`, `--window-function=<function>` – the window function to use in short-time Fourier transformation. Must be one of "hann" (Hann function), "sqhann" (Square root of the Hann function), "hamming" (Hamming function) or "rectangle" (rectangle function). The default is "sqhann".
- `-o<overlap>`, `--overlap=<overlap>` – overlap of windows, given as a number from the interval [0,1). The default is 0.5.
- `-s<size>`, `--windowSize=<size>` – window size in milliseconds. Default is 25.
- `-z`, `--zero-padding` – perform zero-padding before FFT, such that the transformation size is a power of 2.

4.1.4 Separation

- `-m<method>`, `--method=<method>` – The method to be used for component separation. As of the time of writing, this option exists only for extensibility reasons and has no effect.

- `-c<number>`, `--components=<number>` – The number of components which should be separated. Default is 20.
- `-T<number>`, `--spectra=<number>` – The number of spectra which should be computed per component. If the number of spectra is ¿ 1, NMD is performed. Default is 1.
- `-f<name>`, `--cost-function=<name>` – The cost function for NMF/NMD. The following strings are valid: "ed" (Euclidean distance), "kl" (Kullback-Leibler divergence) [3], "eds" (Euclidean distance with a sparsity constraint), "eds" (Euclidean distance with a sparsity constraint), "kls" (KL divergence with a sparsity constraint) [17] and finally "edsn" (Euclidean distance with a sparsity constraint, measured using normalized basis vectors as in [16]). Default is "kl". Note that NMD (i. e. ¿ 1 spectrum per component) can only be performed using the "ed" and "kl" cost functions.
- `-y<number>`, `--sparsity=<number>` – The sparsity parameter for the NMF cost function. Only has an effect if either "eds", "edsn" or "kls" are selected as cost function.
- `-N`, `--normalize-matrices` – Normalize NMF/NMD matrices such that the second factor has unity Frobenius norm.
- `-g`, `--generator=<func>` – Sets the generator function for initialization of the matrices ("gaussian" for absolute Gaussian noise, "uniform" for values uniformly distributed on the interval $[0.01, 0.02)$, or "unity" for every value equal to 1). Default is "gaussian". The "unity" generator makes the separation process deterministic and can hence be used for debugging purposes.
- `-e<number>`, `--precision=<number>` – The desired precision (relative error in terms of Frobenius norm) of the result. If set to zero, the maximum number of iteration steps is performed in any case. Default is 0.
- `-i<number>`, `--max-iter=<number>` – The maximum number of iteration steps. Default is 100.
- `-I<range>`, `--init=<range>` – Pre-initializes the separation using the spectra of several classification objects, specified as a range of classification object IDs. "range" is a string of the form "min..max" where "min" and "max" are IDs of classification objects. This option can be repeated to specify multiple ranges. If the number of initialization objects is smaller than the number of components, randomized spectra are added. The option can be repeated to give multiple ranges of objects for initialization.

- `-P`, `--preserve` – preserves the initialization, i.e. do not update it during iteration. Nevertheless, if the number of initialization objects is smaller than the number of components, the additional randomized are updated in any case.

4.1.5 Component Processing

- `-v`, `--volatile` – run in "volatile" mode, i.e. components are thrown away after the tool terminates. This only makes sense when either the `classify` or one of the "export" options are activated. If the `--volatile` option is *not* specified, components are stored for later use.

- `--export-prefix=<prefix>` – sets the filename prefix for export of components (as WAV files) or matrices.

- `-p<prefix`, `--export-components` – exports the separated components as WAV files with the given prefix.

- `--export-matrices=<name>` – Export the separation matrices. `<name>` can be one of "W" (spectra, first factor), "H" (gains, second factor) or "WH" (both factors, not the product!) The export format is controlled by the `blissart.separation.export.format` configuration option (see Section 4.9).

- `-l<response>`, `--classify=<response>` – performs feature extraction on the separated components, classifies them using training data from the given response, and generates audio files for each class which are named like `<input_file_name>_<class_name>.wav`.

- `-L<label>`, `--preset-label=<label>` – during classification, assigns the label with the given ID to the components which have been initialized by the `-I` option, instead of the class label predicted by the classifier.

4.1.6 Usage Examples

- `septool file.wav`
  Separates `file.wav` into 20 components using the default NMF settings, and saves the components.

- `septool -c30 -s60 -l7 test.wav`
  Separates `test.wav` into 30 components, using a window size of 60 ms, saves the components and classifies them using the response with the ID 7. Assuming this response contains classes "Class1" and "Class2", files named `test_Class1.wav` and `test_Class2.wav` are generated.

- `septool -v -c30 -s60 -l7 test.wav`
  Like the above, except that separated components are not stored.
- `septool -n4 file1.wav file2.wav file3.wav file4.wav file5.wav`
  Separates the files `file1.wav` to `file5.wav` using default settings and saves the components, using at most 4 concurrent threads.
- `septool -v -T5 -c40 -l7 -I11..30 -P -L3 test.wav`
  Separates `test.wav` by means of NMD into 40 components (`-c 40`), each consisting of 5 spectra (`-T5`). Thereby the first 20 are initialized using the classification objects with IDs 11 to 30, which must in turn be NMD components (`-I11..30`). Spectra are not updated during the iteration (`-P`). Classification is done using the response with ID 7 (`-l7`), where the first 20 components are assigned the label with ID 3 regardless of the classifier's decision (`-L3`). Nothing is written to the database (`-v`).
- `septool -v -T20 -c10 -ptestcomp test.wav`
  Separates `test.wav` by means of NMD into 10 components, consisting of 20 spectra each. The components are exported as WAV files with the prefix `testcomp`.
- `septool -v -I1..20 -P -c20 --export-matrices=H test.wav`
  Separates `test.wav` into 20 components whose spectra are all predefined in the classification objects with IDs 1 to 20. The gains matrix (H) is exported to a file.
- `septool --cost-function=kls -y0.5 test.wav`
  Like the first example, but using sparse NMF, setting the sparsity parameter to 0.5.

4.1.7 Multithreading vs. Multiple Processes

> It is important to note that while there is an option to run multiple threads simultaneously from one single instance of the separation tool (in this case, only one *user process* is created by the operating system), starting multiple concurrent instances of the separation tool (and hence multiple user processes) can lead to errors, as the integrated SQLite database can only be written by one user process at a time.

4.2 Feature Extraction Tool

The feature extraction tool (`fextool`) extracts features from stored components and saves them into the database.
It can be controlled via the following command line options:

- `-h`, `--help` – only display information about command line parameters.
- `-A`, `--echo` – print the base name of the application binary and its named command line options in long format, with their parameters if given, before executing.
- `-C`, `--config=<filename>` – use the specified configuration file (properties format) instead of the default one. See section 4.9 for details.
- `-a`, `--all` – performs feature extraction for all components whose data is available.
- `-p<id>`, `--process=<id>` – performs feature extraction on the components that have been generated by the separation process with the given ID.
- `-n<number>`, `--num-threads=<number>` – the number of concurrent threads to use for separation and classification. Should be set to the number of CPUs (cores) present in the computer for maximum performance.

The feature extraction process itself can be influenced by a great variety of configuration options, which are all listed in section 4.9.5.

The same note about multithreading and multiple instances of the tool applies as for the separation tool (4.1.7).

4.3 Cross-Validation Tool

The cross-validation tool (`cvtool`) performs stratified cross-validation of a data set given by a response.
The following options can be specified on the command line:

- `-h`, `--help` – displays information about command line parameters.
- `-A`, `--echo` – print the base name of the application binary and its named command line options in long format, with their parameters if given, before executing.
- `-C`, `--config=<filename>` – use the specified configuration file (properties format) instead of the default one. See section 4.9 for details.
- `-r<id>`, `--response=<id>` – gives a response ID. All classification objects that are assigned a label in this response are validated.
- `-f<n>`, `--fold=<n>` – gives the number of folds. If 0 is given, leave-one-out cross-validation is performed. The default value is 10.

- `-t<id>`, `--train=<id>` – gives a response ID for a training set instead of performing $n-$fold cross-validation.
- `-s`, `--shuffle` – shuffles the data set before validation, i.e. randomly reorders the classification objects within the data set. Of course, this does not make sense for leave-one-out cross-validation.
- `--fs=<algorithm>` – enables automatic feature selection. If `algorithm` is `anova`, features are rated by their t-test score (only available for responses with two classes). Otherwise, if `algorithm` is `correlation`, features are rated by their correlation with their class label.
- `-m<number>`, `--max-features=<number>` – gives the maximum number of features that automatic feature selection should select. The default value is 10.
- `-v`, `--verbose` – enables verbose output (see below).
- `-p`, `--prob` – estimates probabilities for SVM classification. If this option is given, verbose output is automatically enabled.
- `--dump[=<prefix>]` – for each fold, write the training and test data to an ARFF data file with the given prefix (default prefix: `fold`). These files can be used to manually reproduce the cross-validation result with the Weka [6] software.

If a response ID was specified, the tool outputs the number of classification objects that were validated, the recalls for each class, the mean recall, as well as the overall accuracy. Finally, a confusion matrix for all classes in the response is printed.

If verbose output is enabled, additionally a list of misclassified objects, their ID, their class label, their predicted class label and, if the corresponding option is given, their prediction probabilities is printed.

Unless automatic feature selection is enabled, the features to be used for classification are read from the configuration file (see section 4.9.5).

4.3.1 Usage Examples

- `cvtool -r7 -f3`
  Validates the response with ID 7 using stratified 3-fold cross validation and the feature set given by the configuration file.
- `cvtool -r7 -t8`
  Validates each object in the response with ID 7, using the response with ID 8 as training set.

- `cvtool -r7 --fs=anova -m10 -p`
  Validates the response with ID 7 using stratified 10-fold cross validation, using the 10 features which score best in a t test, and outputs all objects which have been misclassified along with the classification probability.

4.4 Export Tool

The export tool (`export`) exports objects (usually NMF components) in the storage to a file. HTK or Gnuplot output format can be selected.
The following options can be specified on the command line:

- `-h`, `--help` – displays usage information.
- `-a`, `--all` – exports the data from all data descriptors of the given type (`-t`) in the database.
- `-p<list>`, `--process=<list>` – exports data descriptors associated with the given process IDs. Single process IDs or ranges (`x..y`) can be given and must be separated with commata.
- `-f<format>`, `--format=<format>` – selects an output format. Must be one of "htk" or "gnuplot". Default is "htk".
- `-c`, `--concat` – concatenates data descriptors of the same type, so that only one output file per type is generated. The type of concatenation (column- or row-wise) depends on the type of data descriptor: Spectra are considered column vectors, hence concatenated column-wisely; conversely, gains are considered row vectors, hence concatenated row-wisely. Magnitude and phase matrices are concatenated column-wisely.
- `-t<type>`, `--type=<type>` – selects the type of data descriptor to export. Available types are: Spectrum ("spect"), Gains ("gains"), Magnitude Matrix ("mmatr"), and Phase Matrix ("phase").
- `--strip-prefix=<path>` – when selecting the output file name, the default is to use the full path name of the corresponding input file is used. This option can be used to strip a certain path prefix, to create relative file names.
- `--target-dir=<path>` – sets the target directory for output. Output files are placed in this directory, and relative path names are interpreted with respect to this directory.
- `-T`, `--add-type` – adds a string giving the type of data to the file names, e. g. "spect".

4.4.1 Usage Example

`export -p17 -fgnuplot -tgains -c`
Exports the gains vectors created in process 17 and concatenates them. The output (in this case a gains matrix) is written to a file in Gnuplot format.

4.5 Audio Export Tool

In contrast to the export tool, the audio export tool (`exportaudio`) exports objects (usually NMF components) in the storage to an audio file. The following options can be specified on the command line:

- `-h`, `--help` – displays usage information.
- `-o<id1>[..<id2>]`, `--object-id=<id1>[..<id2>]` – selects the objects to export. Single IDs or ranges of IDs can be given. This option can be repeated to export multiple objects or ranges.

4.6 Cleanup Tool

Because openBliSSART stores binary data in a filesystem directory which is physically independent of the database, there exist some cases where 'orphaned' binary files remain in the storage directory, without a data descriptor referencing them.

The purpose of the cleanup tool (`cleanup`) is to purge the storage directory of these files. After execution, it displays the number of files that have been deleted.

The `-s` or `--simulate` option can be used if no deletions should be performed, but just the number of "orphaned" files should be printed.

4.7 Browser

The main purpose of the browser application is to facilitate the creation of data sets (responses) which can be used for classification of NMF components in blind source separation. It also supports playback of components, displays component features and allows export of selected data sets to Weka [6] for a more detailed assessment of suitability.

The user interface has been designed with simplicity in mind, i.e. having everything at hand where it might be needed or helpful. Thus, the database entities are displayed in a tree-like view on the main window's left-hand side. Further information related to any entity can be displayed by simply expanding the corresponding subtree. For an example, refer to figure 4.1.

Also, when selecting a database entity, edit and/or preview facilities will be provided on the user interface's right-hand side, the so-called *edit area*. Furthermore, almost every item provides a context-sensitive menu that shows up when the user presses the right mouse button on an item.

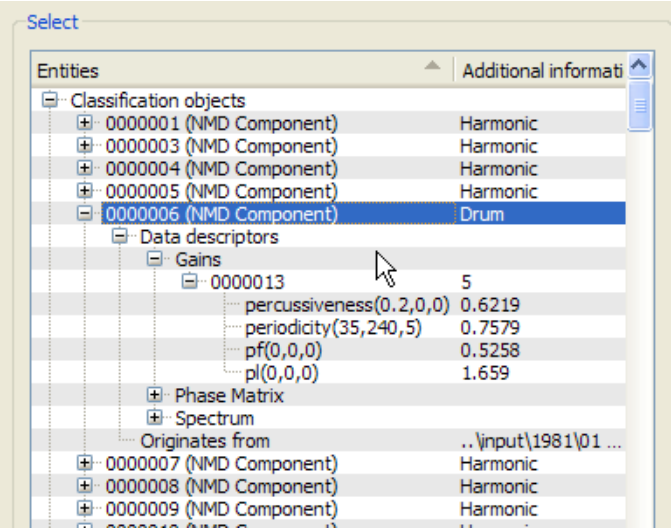

Figure 4.1: Example subtree expansion

4.7.1 Typical Workflow

The typical workflow for supervised component classification in blind source separation includes

- the import of audio files (separated into components),
- the extraction of the related features,
- the creation of various labels with arbitrary precision,
- the assignment of one or more labels to selected classification objects,
- the creation of one or more responses, and finally
- the assignment of classification objects to one or more responses.

4.7.2 Import of Audio Files

Figure 4.2 shows an example of the "Import audio" dialog. This dialog can be displayed either by pressing the respective button or by selecting the corresponding entry from the application's main menu, or alternatively the context menu of the tree view.

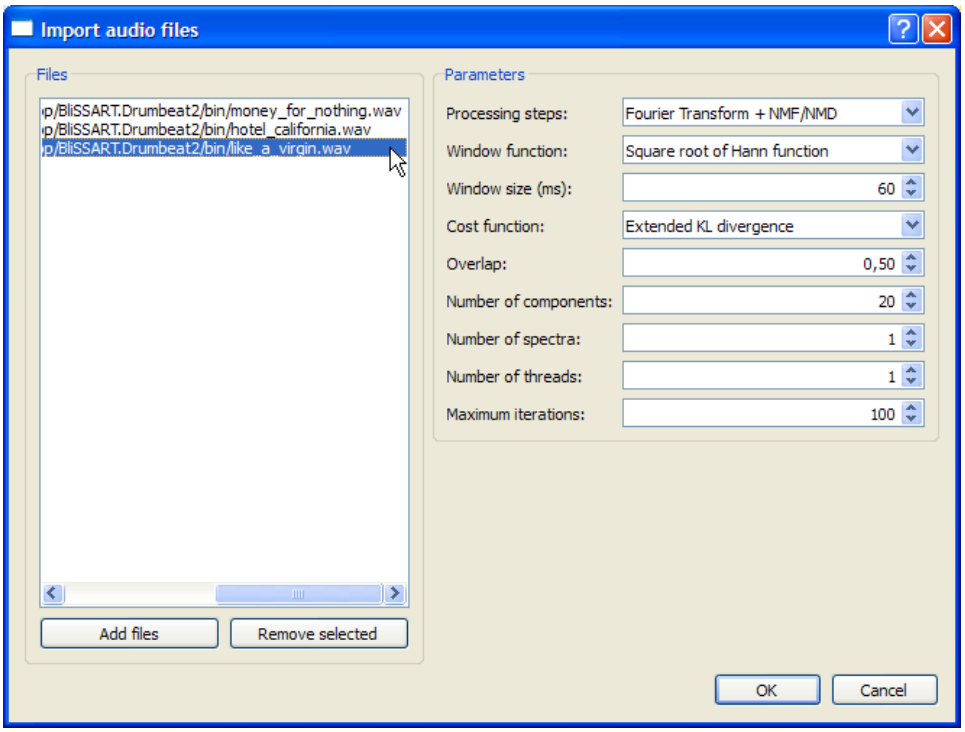

Figure 4.2: Example audio import dialog

While an arbitrary number of input files can be specified on the left-hand side, the right-hand side allows the selection of the intended parameters for the separation process. Currently only a subset of the parameters (e. g. cost functions) of the separation tool is offered by the browser. The user can choose whether to perform a separation process, or whether to only load the file's spectrogram into the database.

Note that increasing the number of threads is only useful when working with multiple files because they will be distributed individually among the available worker-threads. Also, the number of threads should not exceed the number of available processors as there are only few disk operations but rather heavy computational costs involved in the separation process.

The components of the chosen audio files will appear in the "Classification objects" tree on the left hand side of the Browser main window.

4.7.3 Feature Extraction

While it is possible to extract the features of individual classification object (see figure 4.3) via their context menu, the features of all classification objects can be extracted in one step as well by selecting the "Database" / "Extract features from all data descriptors" item in the application's menu.

Again, the number of threads can be specified when extracting all features at once and significantly reduces the processing time on multiprocessor machines.

> If you change the configuration options for feature extraction (see 4.9.5), you have to restart the browser for changes to take effect.

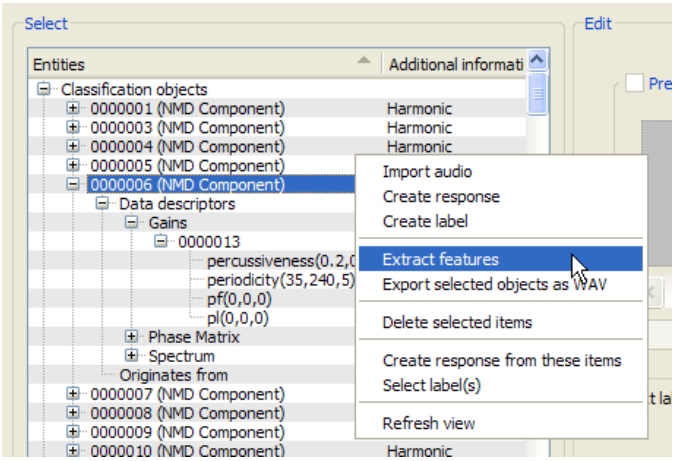

Figure 4.3: Feature extraction

4.7.4 Label Creation

Labels can be created either by pressing the corresponding "Create label" button located at the user interface's lower left or by selecting the appropriate item from the application's or context menu. Creating a label automatically inserts the new label into the tree view, selects it and allows editing of the label's properties inside the edit area.

4.7.5 Assignment of Labels to Classification Objects

After a suitable set of labels has been created, these labels have to be assigned to classification objects wherever appropriate. Selecting a classification object shows a list of all available labels inside the edit area. One or more labels can be assigend by checking the corresponding checkbox and then saving this selection. Figure 4.4 shows the selection of multiple labels for a particular classification object. In order to determine which of the available labels satisfy the needs of a particular classification object, one can use the application's preview feature so as to visually explore the samples or else playing them back. Depending on the applications preferences, the "Preview" checkbox is checked automatically. If not, either manually check that box to be able to explore the samples or select the corresponding option in the preferences dialog.

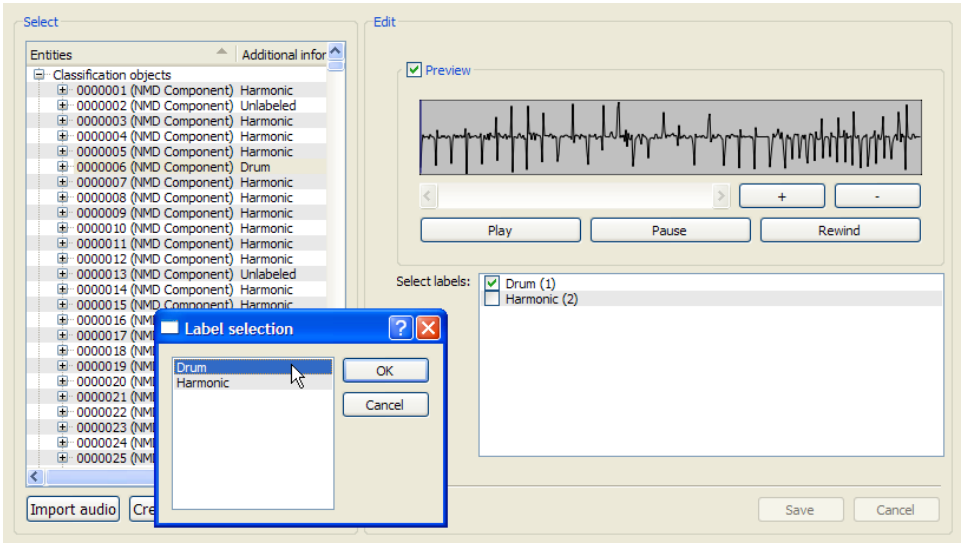

Figure 4.4: Assignment of labels to classification objects

It is also possible to select one or more labels for multiple classification objects at once by means of the "Select label" item in their context menu. In this case, a dialog is shown which allows the selection of one or more labels. The selected labels are assigned to each selected classification object. Existing labels are not removed.

4.7.6 Response Creation

To create an empty response, either press the "Create response" button located at the user interface's lower left or select the corresponding item from the main menu or the context menu of the tree view. The newly created response is automatically inserted into the entities tree while the response's properties (name, description and assigned classification objects) can be modified inside the edit area.

To create a response that contains a set of classification objects, simply select the desired classification objects in the tree view and click "Create response from these items" in the context menu.

4.7.7 Adding Classification Objects to Responses

Currently the only way to assign classification objects to an existing response is via the "Add CLO's by label" button located inside a response's edit area. Pressing this button pops up a dialog that allows the selection of the desired label. Thereupon all classification objects related to this label will be assigned to the current response.

Since multiple labels can be assigned to a classification object, one might wish to change the label in-use. In order to do that, simply select the corresponding classification object from the list inside the response's edit area and press "Select label". Note that this button will be enabled as soon as *more than one* label is linked to the selected classification object.

Classification objects can be removed from the assignment list by selecting them followed by pressing the "Remove selected" button.

As with all of the browser's edit options, the newly made assignments are not automatically stored. Instead, they have to be saved explicitly.

Figure 4.5 shows the described features.

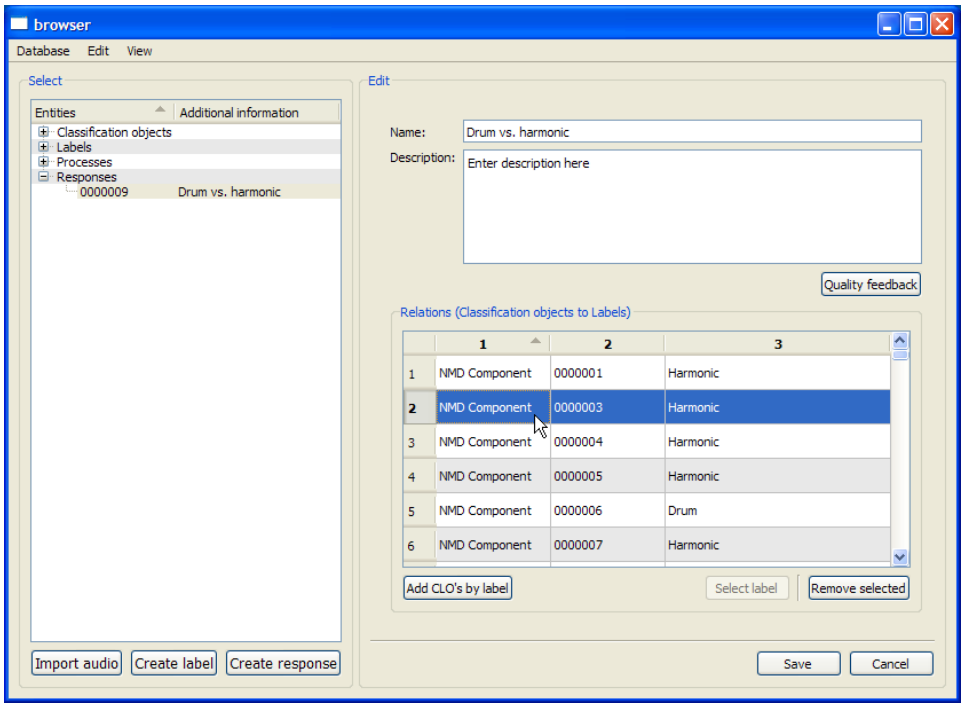

Figure 4.5: Assignment of classification objects to a response

4.7.8 Exporting Selected Objects

If a selection of classification objects should be exported as audio files, one can simply select the desired objects and choose "Export selected objects as WAV" via the corresponding objects' context menu item. When selecting this item, a directory selection dialog shows up and allows selecting the destination directory for the exported files.

4.7.9 Browser Preferences

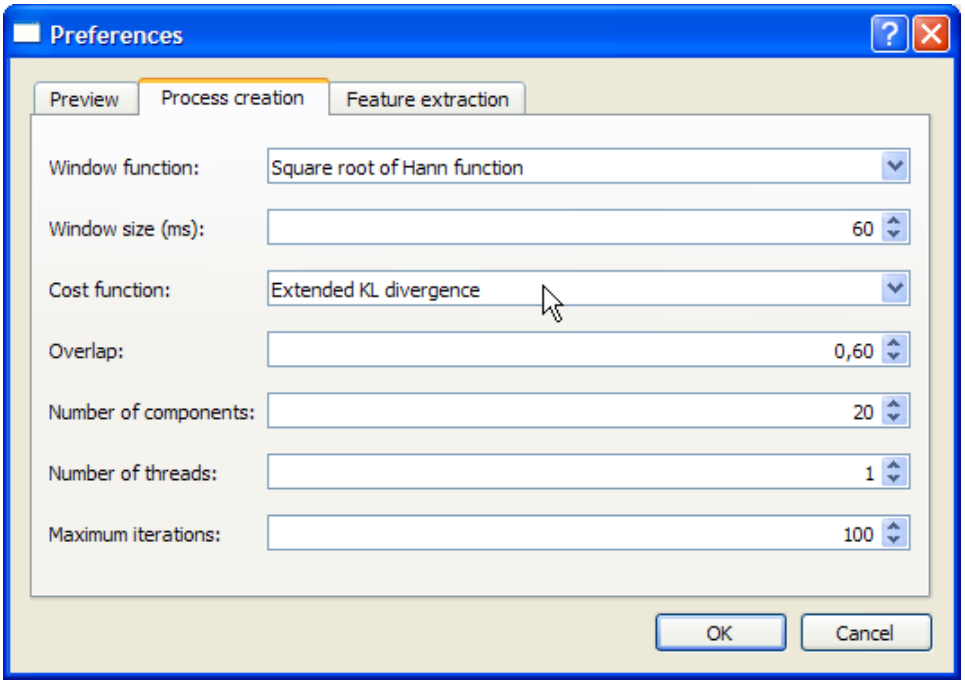

Figure 4.6: Browser Preferences Dialog

Figure 4.6 shows the preferences dialog of the Browser, which allows the user to choose options for the audio preview, select the default parameters for creating separation processes from the Browser, and set the default number of threads to use for feature extraction.

4.8 ICA Tool

The ICA tool (`icatool`) performs blind source separation on multiple audio input files by applying independent component analysis to the corresponding time signals. Possible choices for the output of the results are either WAVE audio files or Weka ARFF format.

The format of the input files may differ[2], yet all of them must have the same sampling rate and equal number of samples. Should the latter vary, the corresponding signal can be expanded by using the expected value of its time signal.

[2]Generally speaking, all audio file formats supported by the SDL_sound library can be read.

Whenever an input file contains more than one channel, only the first one will be used for computation.

If the number of sources to be separated is smaller than the number of input files, the corresponding number of signals with the greatest variance and thus most information will be selected from all available signals. Since principal component analysis is a preprocessing step for ICA anyway, this yields no particular further computational effort.

Readers should note that this is a stand-alone application that makes no further use of the framework's storage- and/or classification components.

General

- `--help` – display information about command line parameters and exit.
- `--as-wave` – output the results as WAVE audio files, which is also the default.
- `--as-arff` – output the results as Weka ARFF files.
- `--prefix=<prefix>` – the prefix to be used for the output files. The filenames will be comprised of `<prefix><nr>.<format>`, where `<nr>` equals the number of each separated source and `format` resembles the chosen output file format.

Separation

- `--nsources=<x>` – the number of sources to be separated. Must be greater one and less than or equal to the number of input files.
- `--force` – in case of varying lengths of the input signals, extends shorter input signals by their expected values instead of aborting.
- `--prec=<x>` – the desired precision for the projection of the components. Must be a real value greater than 10^{-20}. Defaults to 10^{-10}.
- `--max-iter=<x>` – the maximum number of iterations per component for FastICA. Applies only if the desired precision has not been achieved before reaching this limit.

4.8.1 Usage Examples

- `icatool --prefix=foo mix31.wav mix32.wav mix33.wav`
  Performs ICA on the given input files and outputs the results as WAVE audio files with the names `foo1.wav`, `foo2.wav` and `foo3.wav`.

- `icatool --as-arff --prefix=baz mix4[1-4].wav`
  Performs ICA on the four given input files `mix41.wav` to `mix44.wav` and outputs the results in Weka ARFF format with the names `baz1.arff` to `baz4.arff`.

- `icatool --prefix=ext --force shorter.mp3 longer[1-2].mp3`
  Performs ICA on the three given input files, one of which has less samples than the others. The time signal of the "delinquent" is expanded by its expected value. Output will be as WAVE audio files with the names `ext1.wav` to `ext3.wav`.

- `icatool --prefix=reduced --nsources=2 mix5[1-5].ogg`
  Performs ICA on the five given input files `mix51.ogg` to `mix55.ogg` and output the results as WAVE audio files. Before the actual application of ICA, however, the two principal signals, i.e. the signals with the greatest variance and thus most information, are selected amongst all available signals.

4.9 Configuration Files

Audio processing, feature extraction, classification and browser behavior can be fine-tuned by means of configuration files in the Java properties file format. Basically, files in this format may contain option lines of the form

`<option-name>: <option-value>`

as well as comment lines starting with `#`, which are ignored. Boolean values can be notated as `0`, `false` or `1`, `true`, respectively.

The configuration files reside in the `etc` directory of the installation tree.

4.9.1 Global Options

- `blissart.global.mfcc.count` (positive integer): The number of Mel frequency cepstral coefficients (MFCCs) to compute. Default is 13.

- `blissart.global.mfcc.mfcc0` (boolean): Whether the first MFCC should be computed. Default is `true`. If this option is set to `false` and `blissart.global.mfcc.count` is set to N, MFCCs 1 through $N - 1$ are computed.

- `blissart.global.mfcc.lifter` (double): The parameter for MFCC liftering. Liftering with parameter L means that the ith coefficient is multiplied with $1 + L/\sin(2\pi i/L)$, i.e. if $L = 0$ this procedure has no effect. More information can be found in the HTK book [7].

- `blissart.global.mel_filter.high_freq` (double): the upper limit frequency of the Mel filter bank. If this is 0 (default), the Nyquist frequency is assumed. If this is larger than the Nyquist frequency, an error is raised.
- `blissart.global.mel_filter.low_freq` (double): the lower limit frequency of the Mel filter bank. Default 0.
- `blissart.global.mel_bands` (positive integer): the number of Mel frequency bands to use for Mel filtering (e.g. in MFCC computation).
- `blissart.global.deltaregression.theta` (positive integer): The parameter θ for the regression procedure which is used to compute delta- and delta-delta MFCCs. More information can be found in the HTK book [7].

4.9.2 Audio Preprocessing

Audio preprocessing options can be specified in the configuration file `blissart.properties`. These are valid for the browser as well as the separation tool, but can be overridden by passing the corresponding command line parameters to the separation tool.

- `blissart.audio.remove_dc` (boolean): See the `--remove-dc` option of the separation tool.
- `blissart.audio.preemphasis`: See the `--preemphasis` option of the separation tool.
- `blissart.audio.reduce_mids`: See the `--reduce-mids` option of the separation tool.

4.9.3 Transformation

Options for the short-time Fourier transformation can be specified in the configuration file `blissart.properties`. Some of these can be overridden in the "Import audio" dialog of the browser, as well as by passing the corresponding command line parameters to the separation tool. In addition, the short-time Fourier spectrograms can be transformed in various ways, as will be explained below.

- `blissart.fft.windowfunction` (string): See the `--window-function` option of the separation tool.
- `blissart.fft.windowsize` (positive integer): See the `--window-size` option of the separation tool.

- `blissart.fft.overlap` (double): See the `--overlap` option of the separation tool.
- `blissart.fft.zeropadding` (boolean): See the `--zero-padding` option of the separation tool.
- `blissart.fft.transformations.powerSpectrum`: If set to `true`, converts the spectrum to the power spectrum (default: square).
- `blissart.fft.transformations.powerSpectrum.gamma`: The exponent for the power spectrum (default 2.0).
- `blissart.fft.transformations.melFilter`: If set to `true`, applies a Mel filterbank to the spectrogram. The number of Mel bands is controlled by the `blissart.global.mel_bands` global option.
- `blissart.fft.transformations.slidingWindow`: If set to `true`, applies a 'sliding window' to the spectrogram, i. e. multiple columns (frames) are concatenated into a single column.
- `blissart.fft.transformations.slidingWindow.frameSize`: The 'frame size' for the sliding window transformation, i. e. the number of columns to concatenate for each output column. Default is 10.
- `blissart.fft.transformations.slidingWindow.frameRate`: The 'frame rate' for the sliding window transformation, i. e. the number of columns to skip between subsequent concatenations.

4.9.4 Separation

Options for the separation process can be specified in the configuration file `blissart.properties`.

- `blissart.separation.notificationSteps`: The number of iteration steps after which a notification is generated, i. e. the progress bar is updated in the septool and Browser applications. Default ist 25. Setting this number to a low value may result in performance loss for small input files, whereas raising it to a high value prevents any progress begin seen over long periods of time.
- `blissart.separation.export.format`: One of "bin", "htk" or "gnu" for BliSSART binary matrix format, HTK format or Gnuplot format, respectively. This option has an effect on the separation tool with the `--export-matrices` option enabled.
- `blissart.separation.storage.phasematrix`: `true` (default) if the separation tool should store the phase matrix of the original signal, `false` otherwise.

- `blissart.separation.storage.magnitudematrix`: `true` if the separation tool should store the magnitude matrix of the original signal, `false` otherwise (default). Usually this option should be disabled.

4.9.5 Feature Extraction

Feature extraction options can be found in the configuration file `blissart.properties`. Unless stated otherwise, these options are boolean values which include/exclude certain features in the feature set. The available features, and the default set by data descriptor type, is shown in Table 4.1.

Data descriptor type	**Feature**	**Default**
Magnitude matrix	(Sampled) MFCCs	x
	$\delta + \delta\delta$ coefficients	x
	Mean and standard deviation of $\delta + \delta\delta$ coefficients	x
Spectrum	(Mean) MFCCs 0-12	x
	(Sampled) MFCCs	
	$\delta + \delta\delta$ coefficients	
	Mean and standard deviation of $\delta + \delta\delta$ coefficients	
	Standard deviation	
	Spectral centroid	
	Spectral rolloff	
	Noise-likeness	
	Dissonance	
	Flatness	
Gains	Standard deviation	
	Skewness	
	Kurtosis	
	Periodicity	
	Peak length	
	Peak fluctuation	
	Percussiveness	

Table 4.1: Available Audio Features

Note that for NMD, "spectra" are actually spectrograms, hence functionals of MFCCs and the other features are computed (mean, standard deviation), and sampled values of MFCCs can be computed.

The following options control feature extraction from magnitude matrices:

- `blissart.features.magnitudematrix.mfcc`: Whether to compute MFCCs. MFCCs are sampled at a given number of equidistant frames which can be modified by the `blissart.features.magnitudematrix.mfcc.frame_count` option (default 5).
- `blissart.features.magnitudematrix.mfccD`: Whether to compute delta coefficients (using the regression procedure described in the HTK book [7]).
- `blissart.features.magnitudematrix.mfccA`: Whether to compute delta-delta (**A**cceleration) coefficients (using the regression procedure described in the HTK book [7]).
- `blissart.features.magnitudematrix.mean_mfcc`: Whether to compute the mean of each MFCC (and possibly its regression coefficients) over the whole signal.
- `blissart.features.magnitudematrix.stddev_mfcc`: Whether to compute the standard deviation of each MFCC (and possibly its regression coefficients) over the whole signal.

The following options control feature extraction from spectra:

- `blissart.features.spectrum.mean_mfcc`: For NMF, these are simply the MFCCs. For NMD, this option indicates whether to compute the mean of each MFCC (and possibly its regression coefficients) over the whole signal.
- `blissart.features.spectrum.stddev`: Whether to compute standard deviation.
- `blissart.features.spectrum.centroid`: Whether to compute the spectral centroid.
- `blissart.features.spectrum.rolloff`: Whether to compute spectral rolloff.
- `blissart.features.spectrum.noiselikeness`: Whether to compute noise-likeness ([22]).
- `blissart.features.spectrum.noiselikeness.sigma`: The *sigma* (standard deviation) parameter for the calculation of noise-likeness ([22]).
- `blissart.features.spectrum.dissonance`: Whether to compute spectral dissonance ([22]). Be aware that this operation can be time-consuming, as its time complexity is quadratic in the length of the spectra.

- `blissart.features.spectrum.flatness`: Whether to compute spectral flatness [22].

Furthermore, the `blissart.features.spectrum.mfcc`, `blissart.features.spectrum.mfccD`, `blissart.features.spectrum.mfccA`, and `blissart.features.spectrum.stddev_mfcc` configuration options are available for spectra, but make only sense for NMD where each component is described by a spectrogram.
The default is to only compute the (mean) MFCCs.

The following options control feature extraction from gains vectors:

- `blissart.features.gains.stddev`: Whether to compute standard deviation.
- `blissart.features.gains.pl`: Whether to compute peak length [5].
- `blissart.features.gains.pf`: Whether to compute peak fluctuation [5].
- `blissart.features.gains.percussiveness`: Whether to compute percussiveness [22].
- `blissart.features.gains.percussivness.length` (double): The length (in seconds) of the percussive impulse to use for computation of percussiveness.
- `blissart.features.gains.periodicity`: Whether to compute periodicity of gains [5].
- `blissart.features.gains.periodicity.bpm_min` (positive integer): The minimum bpm (beats per minute) value to consider for periodicity.
- `blissart.features.gains.periodicity.bpm_max` (positive integer): The maximum bpm (beats per minute) value to consider for periodicity.
- `blissart.features.gains.periodicity.bpm_step` (positive integer): The distance between the bpm values to consider for periodicity.

4.9.6 Classification

Classification options control SVM parameters and scaling. They can be specified in the configuration file `blissart.properties`.

The type of kernel function that is used to build the SVM is given by the `blissart.classification.svm.kernel` option. Possible values include `linear` for linear functions, `poly` for polynomials of higher degree, `rbf` for radial basis functions and `sigmoid` for sigmoid functions. Default is `linear`. The polynomial degree can be given by the `blissart.classification.svm.degree` option, which defaults to 3.

The precision of the training procedure is controlled by the `blissart.classification.svm.epsilon` option (default: $1e-3$).

"Bias" components (i.e. one component that is always 1) can be added by settings the `blissart.classification.addBias` to `true`.

Scaling is controlled by the `blissart.classification.scaling` family of options:

- `blissart.classification.scaling.method`
  - `minmax` for linear scaling such that all values of one feature are in a given interval (by default $[-1, 1]$),
  - `musigma` for linear scaling such that all values of one feature have the given mean μ and standard deviation σ (by default $\mu = 0, \sigma = 1$),
  - `none` for no scaling.
- `blissart.classification.scaling.lower` – lower bound of the scaling interval if `blissart.classification.scaling.method` is set to `minmax`.
- `blissart.classification.scaling.upper` – upper bound of the scaling interval if `blissart.classification.scaling.method` is set to `minmax`.
- `blissart.classification.scaling.mu` – desired mean of the feature values if `blissart.classification.scaling.method` is set to `musigma`.
- `blissart.classification.scaling.sigma` – desired standard deviation of the feature values if `blissart.classification.scaling.method` is set to `musigma`.

4.9.7 Browser

The browser configuration file `browser.properties` contains options for the audio file preview, and the default settings for importing audio files. The options are listed below:

- `browser.featureExtraction.numThreads` – the default number of threads to use for feature extraction. Default 1.

- `browser.mainwindow.height` – stores the size of the browser window. Default 768.
- `browser.mainwindow.isMaximized` – stores whether the browser window is maximized. Default `false`.
- `browser.mainwindow.width` – stores the width of the browser window. Default 1024.
- `browser.preview.alwaysEnabled` – indicates whether the audio preview should be enabled by default. Default `true`.
- `browser.preview.normalizeAudio` – indicates whether the audio preview should be normalized in amplitude. Default `true`.
- `browser.processCreation.costFunction` – the default NMF cost function (0 for KL divergence, 1 for squared Euclidean distance). Default 0.
- `browser.processCreation.maxIterations` – the default number of NMF iterations. Default 100.
- `browser.processCreation.numComponents` – the default number of NMF components. Default 20.
- `browser.processCreation.numThreads` – the default number of NMF separation threads. Default 1.
- `browser.processCreation.overlap` – the default overlap to use for Fourier Transformation and NMD/NMF processes. Default 0.5.
- `browser.processCreation.windowFunction` – the default window function to use for Fourier Transformation and NMD/NMF processes (0 = Square root of Hann function, 1 = Hann function, 2 = Hamming function, 3 = Rectangle function). Default 0.
- `browser.processCreation.windowSizeMS` – the default window size in milliseconds to use for Fourier Transformation and NMD/NMF processes. Default 25

Bibliography

[1] D. D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, pp. 788–791, October 1999.

[2] P. Smaragdis and J. C. Brown, "Non-negative matrix factorization for polyphonic music transcription," in *Proc. of IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, New Paltz, NY, USA, 2003, pp. 177–180.

[3] D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in *Proc. of NIPS*, Vancouver, Canada, 2001, pp. 556–562.

[4] P. Smaragdis, "Discovering auditory objects through non-negativity constraints," in *Proc. of SAPA*, Jeju, Korea, 2004.

[5] M. Helén and T. Virtanen, "Separation of drums from polyphonic music using non-negative matrix factorization and support vector machine," in *Proc. of EUSIPCO*, Antalya, Turkey, 2005.

[6] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*, Morgan Kaufmann, San Francisco, 2005.

[7] S. Young, D. Kershaw, J. Odell, D. Ollason, V. Valtchev, and P. Woodland, *The HTK Book version 3.0*, Cambridge University Press, 2000.

[8] B. Schuller, A. Lehmann, F. Weninger, F. Eyben, and G. Rigoll, "Blind enhancement of the rhythmic and harmonic sections by NMF: Does it help?," in *Proc. of the International Conference on Acoustics (NAG/DAGA 2009)*, Rotterdam, Netherlands, 2009, pp. 361–364, DEGA.

[9] "SQLite database engine," `http://www.sqlite.org/download.html`, February 2009.

[10] M. N. Schmidt and R. K. Olsson, "Single-channel speech separation using sparse non- negative matrix factorization," in *Proc. of Interspeech*, Pittsburgh, PA, USA, 2006.

[11] K. W. Wilson, B. Raj, P. Smaragdis, and A. Divakaran, "Speech denoising using nonnegative matrix factorization with priors," in *Proc. of ICASSP*, Las Vegas, NV, USA, 2008, pp. 4029–4032.

[12] K. W. Wilson, B. Raj, and P. Smaragdis, "Regularized non-negative matrix factorization with temporal dependencies for speech denoising," in *Proc. of Interspeech*, Brisbane, Australia, 2008.

[13] P. D. O'Grady and B. A. Pearlmutter, "Discovering convolutive speech phones using sparseness and non-negativity constraints," in *Proc. of ICA*, London, UK, 2007.

[14] P. O. Hoyer, "Non-negative sparse coding," in *Proc. of IEEE Workshop on Neural Networks for Signal Processing*, Martigny, Switzerland, 2002, pp. 557–565.

[15] P. O. Hoyer, "Non-negative matrix factorization with sparseness constraints," *Journal of Machine Learning Research*, vol. 5, pp. 1457–1469, 2004.

[16] J. Eggert and E. Körner, "Sparse coding and NMF," in *Proc. of Neural Networks*, Dalian, China, 2004, vol. 4, pp. 2529–2533.

[17] T. Virtanen, "Monaural sound source separation by nonnegative matrix factorization with temporal continuity and sparseness criteria," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 15, no. 3, pp. 1066–1074, March 2007.

[18] P. Smaragdis, "Convolutive speech bases and their application to supervised speech separation," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 15, no. 1, pp. 1–14, 2007.

[19] W. Wang, A. Cichocki, and J. A. Chambers, "A multiplicative algorithm for convolutive non-negative matrix factorization based on squared Euclidean distance," *IEEE Transactions on Signal Processing*, vol. 57, no. 7, pp. 2858–2864, July 2009.

[20] J. F. Gemmeke and T. Virtanen, "Noise robust exemplar-based connected digit recognition," in *Proc. of ICASSP*, Dallas, TX, USA, March 2010.

[21] A. Hyvärinen, "New approximations of differential entropy for independent component analysis and projection pursuit," in *Proc. of NIPS*, Denver, Colorado, USA, December 1998, pp. 273–279.

[22] C. Uhle, C. Dittmar, and T. Sporer, "Extraction of drum tracks from polyphonic music using independent subspace analysis," in *Proc. of ICA*, Nara, Japan, 2003.