AMUSE 0.3 beta
(Advanced MUSic Explorer)
User Manual

Igor Vatolkin

July 12, 2021

TU Dortmund University
Department of Computer Science
Chair of Algorithm Engineering (Ls11)
Otto-Hahn-Str. 14
44227 Dortmund
Germany
igor.vatolkin[AT]udo.edu
http://ls11-www.cs.tu-dortmund.de
http://sig-ma.de

Contents

1 Introduction

1.1 License Notes

Amuse (Advanced MUSic Explorer) is an open-source Java framework for various music data analysis / music information retrieval tasks. It is developed within Computational Intelligence research group[1] headed by Prof. Dr. Günter Rudolph at the Chair of Algorithm Engineering, Department of Computer Science, TU Dortmund University.

1.2 Repository and Requirements

Amuse is available as the GitHub repository[3]. The framework itself is in the folder:

- `https://github.com/AdvancedMUSicExplorer/AMUSE/tree/master/amuse`

The Amuse plugins are in the following folders:

- `https://github.com/AdvancedMUSicExplorer/AMUSE/tree/master/amusePluginChromaToolbox`
- `https://github.com/AdvancedMUSicExplorer/AMUSE/tree/master/amusePluginKeras`
- `https://github.com/AdvancedMUSicExplorer/AMUSE/tree/master/amusePluginLibrosa`
- `https://github.com/AdvancedMUSicExplorer/AMUSE/tree/master/amusePluginMIRToolbox`
- `https://github.com/AdvancedMUSicExplorer/AMUSE/tree/master/amusePluginSonicAnnotator`

This manual is CURRENTLY UNDER DEVELOPMENT and does not contain the complete information how to use Amuse. We plan to release the version 0.3 during this year (2021) together with the updated manual.

You should have a Java 8 OpenJDK or a newer version and JavaFX installed on your system. AMUSE should also run with Oracle Java 1.8 or newer on most systems.

Some of the Amuse components and plugins use Matlab and Python. However, they are not required to run Amuse in general.

The current version was tested on Ubuntu Unix, Windows, and MacOS systems.

1.3 Remarks on History

The following versions of Amuse were previously released:

- **0.1**: The first version presented at ISMIR [14]
- **0.2**: Integration of the annotation editor for individual tracks (e.g., marking time events or segments) and multiple tracks (e.g., assigning genre or emotion tags)
- **0.3b**: The current version (beta) with support of fuzzy / multi-class / multi-label classification, plugins to Librosa [8] and Keras [1], as well as many further improvements

The GitHub commits distinguish between different code contributions:

- **BUGFIX**: A bug fix / correction of wrong behavior
- **UPDATE**: An update to an existing functionality, such as refactoring or code optimizations
- **NEW**: A new feature
- **EPIC**: A substantial update such as the first implementation of the annotation editor

[1] `https://ls11-www.cs.tu-dortmund.de/rudolph/start`
[2] `http://www.gnu.org/licenses`
[3] `https://github.com/AdvancedMUSicExplorer/AMUSE`

1.4 Citing AMUSE

If you use AMUSE for your research, please cite one of the following publications:

- THE FIRST PRESENTATION OF AMUSE AT ISMIR 2010: I. Vatolkin, W. Theimer, and M. Botteck: AMUSE (Advanced MUSic Explorer) – A Multitool Framework for Music Data Analysis. Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR), pp. 33-38, 2010 [14].
- A RECENT OVERVIEW OF THE MOST RELEVANT UPDATES TO AMUSE AFTER 2010: I. Vatolkin, P. Ginsel, and G. Rudolph: Advancements in the Music Information Retrieval Framework AMUSE over the Last Decade. Accepted for Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), 2021 [13].

The list of studies conducted with AMUSE is provided in Appendix B.

1.5 Acknowledgements

1.5.1 Financial Support

The AMUSE development was partly supported within the following projects:

- MUSICDESCRIBER: PERCEPTIONAL MUSIC CONTENT ANALYSIS
  Funded by Nokia Research Center Bochum
  2006–2008
- MULTI-OBJECTIVE OPTIMIZATION OF AUTOMATIC MUSIC CLASSIFICATION BASED ON HIGH-LEVEL FEATURES AND COMPUTATIONAL INTELLIGENCE METHODS
  Funded by Klaus Tschira Foundation
  2009–2013
- EVOLUTIONARY OPTIMIZATION FOR INTERPRETABLE MUSIC SEGMENTATION AND MUSIC CATEGORIZATION BASED ON DISCRETIZED SEMANTIC METAFEATURES
  Funded by German Research Foundation (DFG)
  2018–2021

1.5.2 Developers

Many contributions to the source code and substantial improvements were done by student assistants at the TU Dortmund University: Philipp Ginsel (since 2018), Fabian Ostermann (since 2015), Frederik Heerde (2017-2018), Daniel Stoller (2011-2015), and Clemens Wältken (2008-2011).

2 Backgrounds

2.1 Classification Pipeline in AMUSE

AMUSE allows you to perform all algorithmic steps from the general classification pipeline, namely the extraction of features, their processing, training of classification models, their application and validation, as well as optimization of algorithm parameters. Each step is independently run by a corresponding AMUSE node and is called *task* in the following. The basic AMUSE tasks are:

- FEATURE EXTRACTION: extraction of Mel frequency cepstral coefficients [11] using jAudio library [7]
- FEATURE PROCESSING: normalization of feature values to a given range, application of the principal component analysis [5]
- MODEL TRAINING: creation of a random forest model for music tracks with already processed features and annotated genre labels using WEKA library [2]
- MODEL APPLICATION: application of the previously created random forest model to classify new music tracks
- VALIDATION: estimation of confusion matrix values to measure the performance of the previously created classification model
- OPTIMIZATION: search for the optimal length of classification frames which are assigned to genres

The lists of all available methods are provided in Appendices C-H.

This pipeline is visualized in Figure 1. AMUSE stores the result of each step in the corresponding "database" which is a folder in AMUSE workspace, except for the prediction of a classification model.

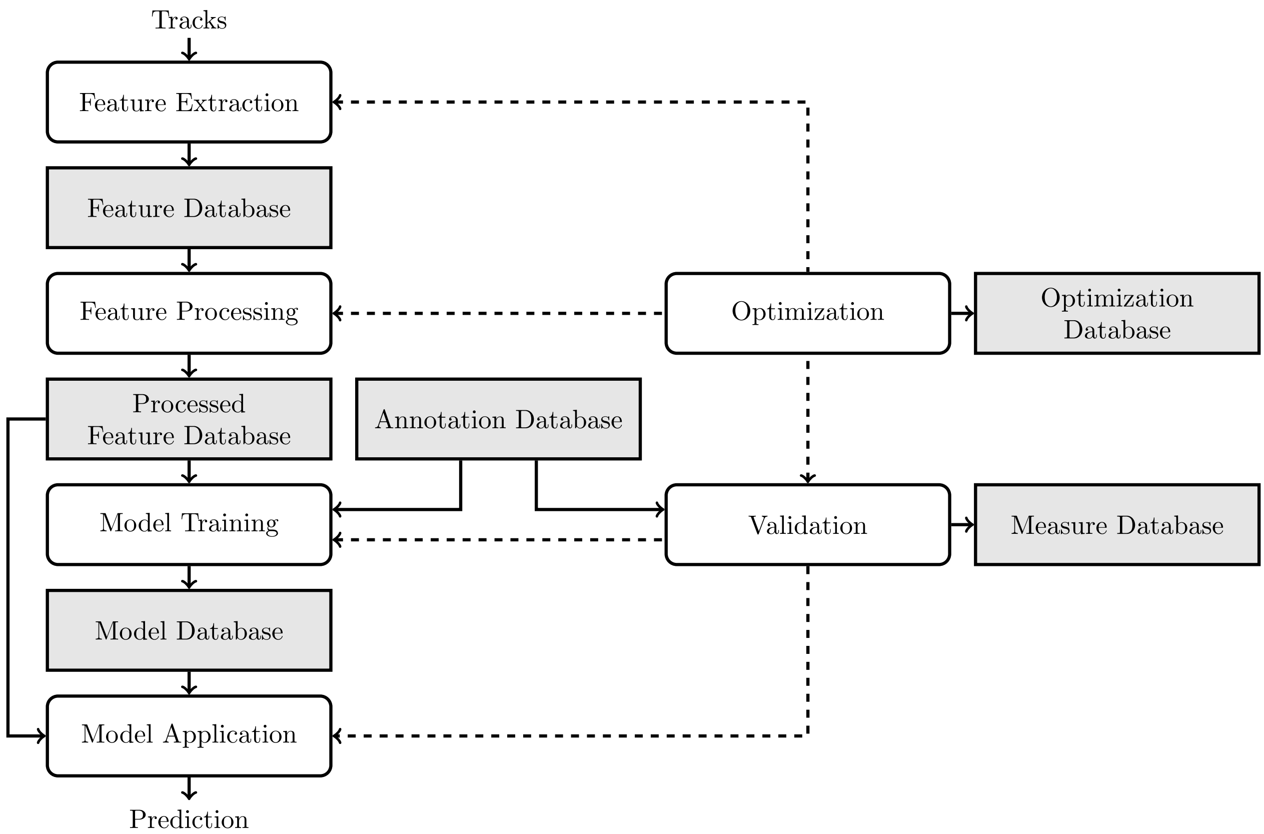

Figure 1: The classification pipeline in AMUSE.

2.1.1 Feature Extraction

Feature extraction stores numeric properties of audio signals which can be used later for music classification and data analysis. These properties are extracted from different domains, such as time domain, spectrum, or cepstrum. Some of them belong to rather low-level signal descriptors (like the number of zero-crossings), others are more interpretable (pitch class profiles or beats).

It is distinguished between the following kinds of features:

- **WINDOWEDNUMERIC**: A feature with a numeric value which is extracted from the frames of the same length and with the same step size, e.g., zero-crossings extracted from frames of 512 samples with no overlap (step size is equal to 512 samples). Note that features which are extracted from the complete audio track (like its duration in seconds) also belong to this category, with the frame length set to -1 samples.
- **WINDOWEDSTRING**: A feature with a string value which is extracted from the frames of the same length and with the same step size, e.g., the predominant chord.
- **EVENT**: Individual time events: onsets, beats, or segment boundaries.
- **SEGMENTEDNUMERIC**: A feature with a numeric value which is extracted from extraction frames of variable length, like the share of vocals in a music segment.
- **SEGMENTEDSTRING**: A feature with a string value which is extracted from extraction frames of variable length, like the chord progression.

Each feature has a unique AMUSE ID. However, it is possible to define additional configurations, e.g., for different extraction frames. After the extraction, the features are stored in the AMUSE feature database as ARFFs, for instance, as:

$$/\underbrace{\text{home/user/AmuseWorkspace/Features}}_{\textit{Path to feature database}}/\underbrace{\text{ACDC/Back\_in\_Black/Hells\_Bells}}_{\textit{Relative path to music track}}/\underbrace{\text{Hells\_Bells}}_{\textit{Track name}}\_\underbrace{400}_{\textit{ID}}.\text{arff}$$

Note that it is not recommended to use white spaces in file names, as some of feature extractors may fail in that case. You may consider to apply sanity.pl[4] script before feature extraction with AMUSE.

Some features require the previous installation of the corresponding plugin, see Appendix I. The list of currently available features in AMUSE is listed in Appendix C.

2.1.2 Feature Processing

The goal of feature processing is to construct classification windows from previously extracted features. As these features may be extracted from different frames, in the first step the harmonized feature matrix is estimated [12], as sketched in Figure 2.

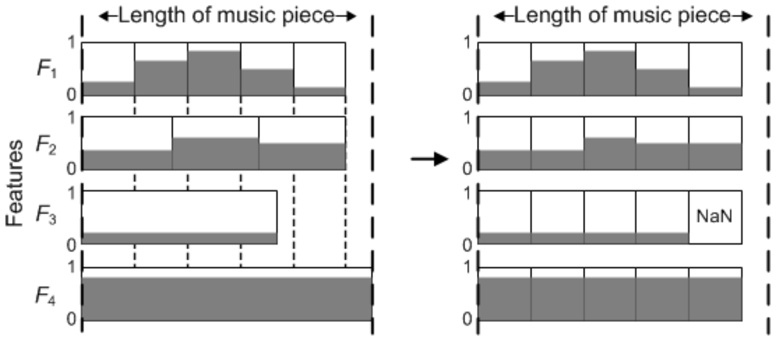

Figure 2: An example of the harmonization of feature matrix [12, p. 373].

Here, the shortest extraction frame length across all features is calculated (feature F_1), and all other feature values are also mapped to these frames. For features extracted from longer frames, the same values may contribute to several consecutive shortest frames, as for features F_2–F_4. In case the boundary of a longer frame is exactly in the shortest frame, the value from the longer frame with the largest overlap to the shortest extraction frame is stored. When no overlap between the longer frame and smallest frame exists (as for the last shortest frame and feature F_3), the "Not a Number" (NaN) value is stored.

Now, various processing methods can be applied which operate on feature or time dimension of the harmonized feature matrix.

- **PREPROCESSING** methods like normalization or replacement of missing values prepare the features for the application of further processing methods and typically do not change the dimensionality of the harmonized feature matrix.

[4] https://github.com/splitbrain/sanity/blob/master/sanity.pl

- **FEATURE DIMENSION PROCESSING** methods typically reduce the number of features, for instance, selecting a limited number of principal components [5] or applying a feature selection strategy removing irrelevant and redundant features [4]. In some cases, feature dimension processing may increase the dimensionality, if new features are constructed by the application of mathematical operators for the original feature dimensions as applied for music data in [10].
- **TIME DIMENSION PROCESSING** methods usually select some frames with regard to musical events (e.g., storing only frames with or between beat events or only frames from the middles of music segments like intro, verse, and chorus). Further, more enhanced concepts like structural complexity [6, 3] can be applied.
- **AGGREGATION OF CLASSIFICATION WINDOWS** is a final step which estimates final feature vectors for each classification window. This can be done using a simple model like the mean and standard deviation for each feature dimension within the classification frame, or also applying more complex time series models which store, e.g., the coefficients of multiple linear regressions over the original feature time series [9].

2.1.3 Model Training

The model training task is responsible for the building of models which assign music data to categories or predict some numerical properties from others. In general, we may distinguish between following method types:

- **SUPERVISED CLASSIFICATION** requires not only processed features, but also annotations (labels) provided by experts or users. Then, the classifier creates some rules which predict these annotations from new data.
- **UNSUPERVISED CLASSIFICATION** assigns data to different groups or clusters without provided annotations. This kind of methods is currently not available in AMUSE but will be integrated in near future.
- **REGRESSION** does not assign categorical labels to processed features directly but estimates a numerical property (a feature or also a numeric label) from one or more other numerical properties. Regression is planned to be integrated in AMUSE in future.

For the ground truth annotations and predictions, it is distinguished between two relationship types:

- **BINARY, OR CRISP RELATIONSHIP** describes a processed feature vector as completely belonging or not belonging to a category, i.e. the relationship grade $y \in \{0, 1\}$.
- **CONTINUOUS, OR FUZZY RELATIONSHIP** describes to which extent a processed feature vector belongs to a category, i.e. the relationship grade $y \in [0, 1]$. As an example, a "progressive rock oper" track may have two relationships $y_{CLASSICAL} = 0.4$ and $y_{ROCK} = 0.8$ (note that as these values do not describe probabilities, they must not produce 1.0 in their sum).

For classification tasks, there exist following strategies to predict the labels:

- **SINGLE-LABEL** predictions assign a sole label to processed features in a binary way, i.e., rating the feature vector as "positive" (belonging to a category or group) or "negative" (do not belonging to a category a group).
- **MULTI-LABEL** predictions assign a sole label to processed features selecting this label from several ones (e.g., assigning a music track to a genre classical, pop, or rock).
- **MULTI-CLASS** predictions assign one or more labels to processed features.

3 Installation and Configuration

3.1 Installation in the Shell

You can get the current version from the GitHub repository using the command:

```
git clone https://github.com/AdvancedMUSicExplorer/AMUSE.git
```

3.2 Installation in Eclipse

To checkout the project with git, please follow these steps:

1. Click on File → Import...
2. Select Git → Projects from Git
3. Select Clone URI
4. Enter the repository location `https://github.com/AdvancedMUSicExplorer/AMUSE.git`.
5. A dialog as shown in Figure 3 should appear. Mark the branch(es) you want to load. Mark only "master", if you want to get only the code intended for users.

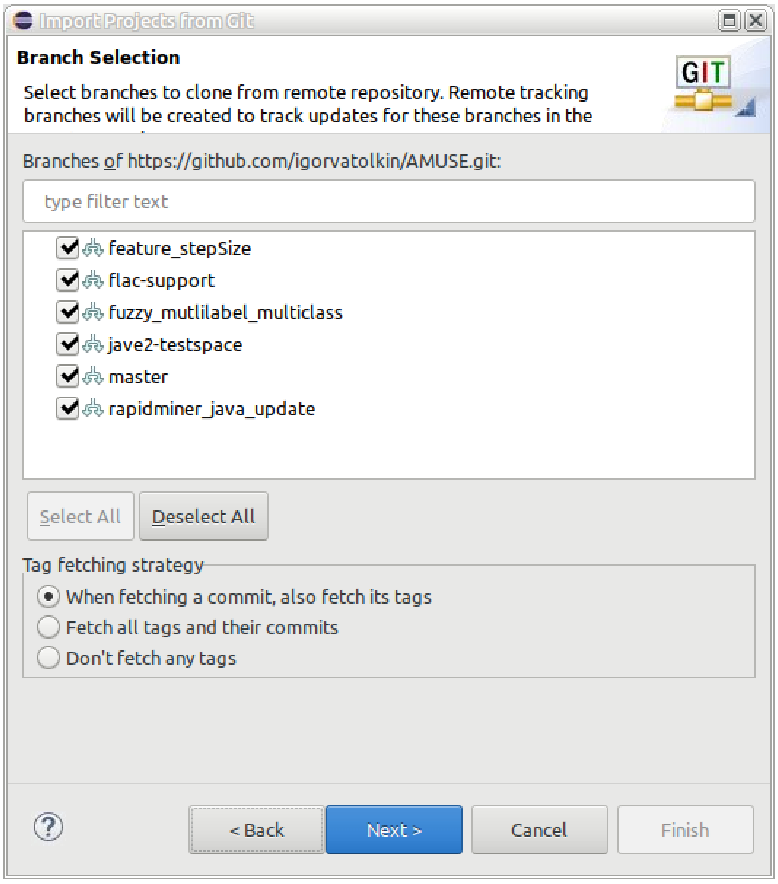

Figure 3: The dialog to select the branches that should be cloned.

6. In the next dialog, choose where AMUSE should be saved as seen in Figure 4.

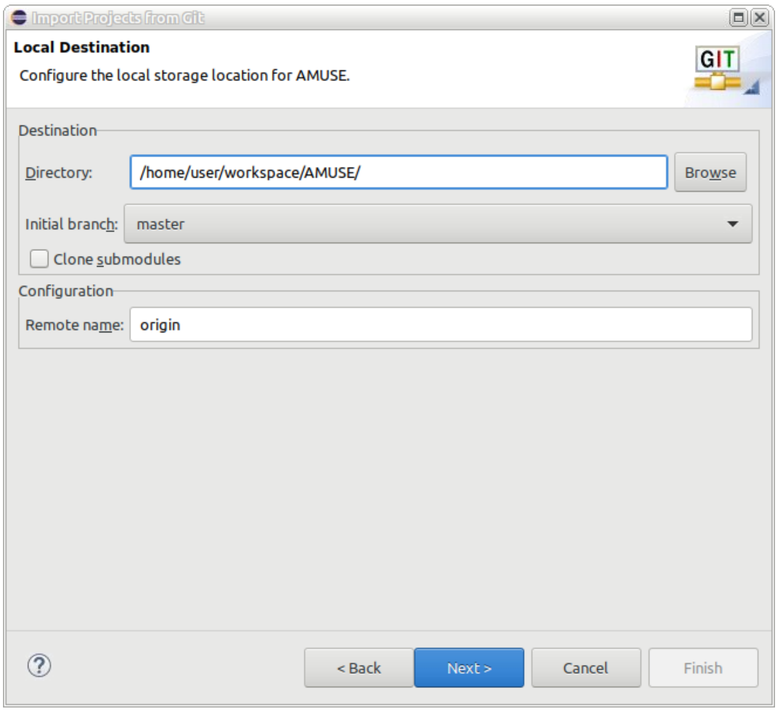

Figure 4: The dialog to select the folder that should be checked out.

7. Select Import existing Eclipse projects
8. In the next dialog (as seen in Figure 5), select the the project(s) you want to load. Mark only "amuse", if you want to get the code from the main project only.

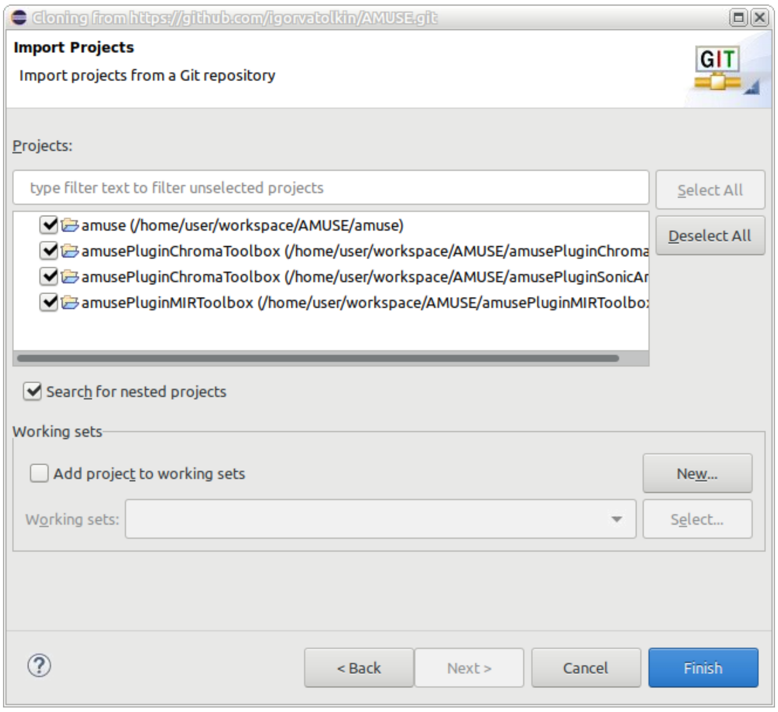

Figure 5: The dialog to select the folder that should be checked out.

9. Click on "Finish".

3.3 Configuration and First Steps

You can run AMUSE from amuseGUI.bat or amuseGUI.sh or from your development environment - in the last case you should use AMUSE.SCHEDULER.GUI.CONTROLLER.WIZARDCONTROLLER as a main class. To ensure that AMUSE runs without issues you should add the environment variable "AMUSEHOME" to your run configuration with the path to your amuse folder and add "-javaagent:lib/jar-loader.jar" as an VM argument.

If you want to run AMUSE via the scripts and cannot start java just using a "java" command from your command line, please provide the path to it in the files amuseGUI.bat (for Windows) or amuseGUI.sh (for Unix). If you use Unix, check if the sh-script amuseGUI.sh is executable.

Now run amuseGUI.bat or amuseGUI.sh. A start screen will appear, cf. Fig. 6 in Section 4.1. Click on the button "Edit Amuse Settings" for the further setup.

4 Using AMUSE with GUI

4.1 Amuse Wizard

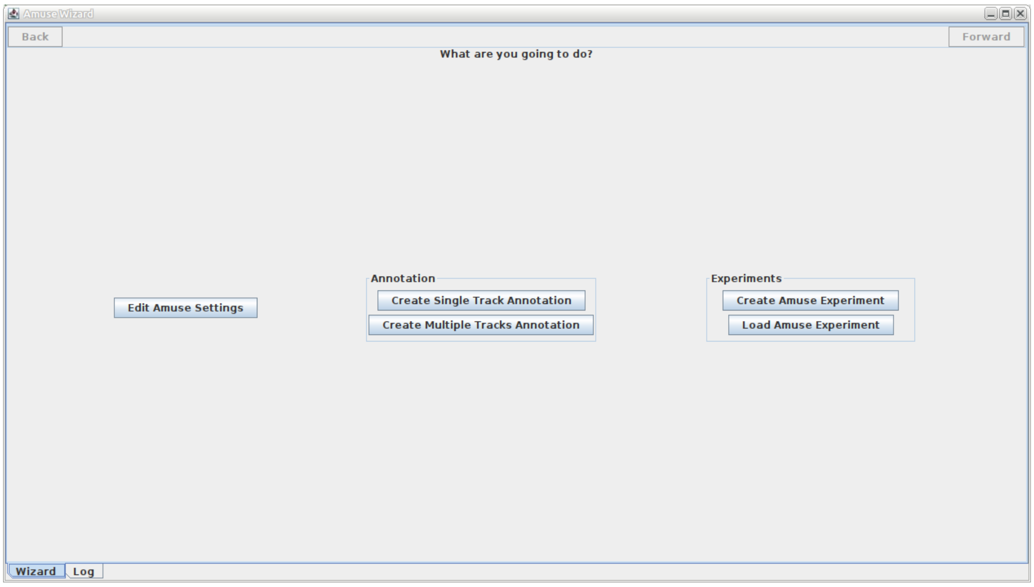

Figure 6: AMUSE Start Screen

The start screen of AMUSE is shown in Figure 6. From here, you can access the different functionalites from AMUSE.

- **EDIT AMUSE SETTINGS** Shows the screen for changing the preferences that is described in Section 4.2.
- **CREATE SINGLE TRACK ANNOTATION** Opens the editor used for annotations of a single track.
- **CREATE MULTIPLE TRACK ANNOTATION** Displays the editor used for annotations of multiple tracks.
- **CREATE AMUSE EXPERIMENTS** Opens the Experiment Configurator.
- **LOAD AMUSE EXPERIMENT** Shows a dialog in which you can select an experiment configuration to load.

4.2 Preferences

4.2.1 General Settings

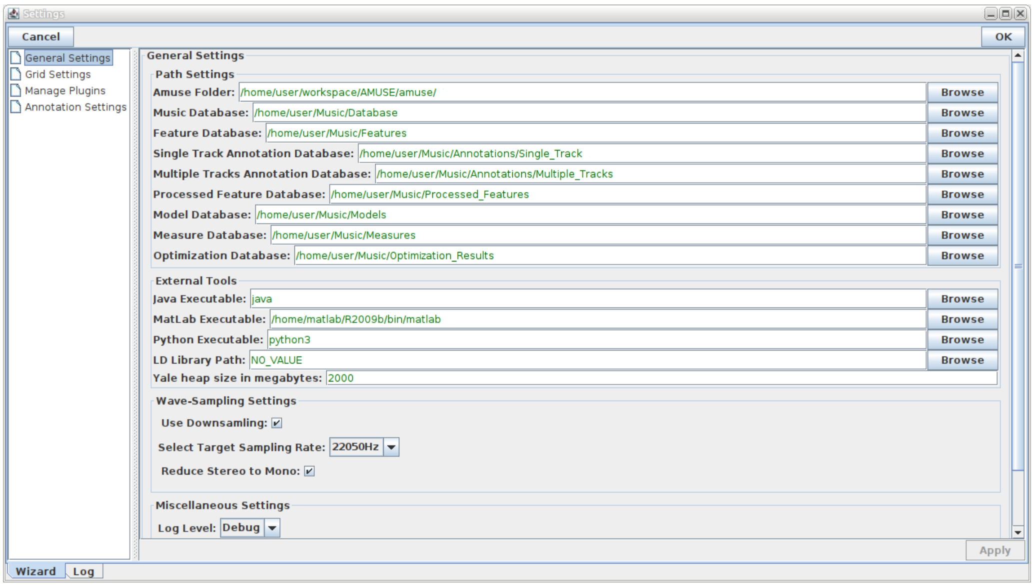

Figure 7: General Settings

In Figure 7, the screen for the general settings is shown.

- **PATH SETTINGS**: Here, the path to AMUSE folder must be set as well as several folders for AMUSE input/output data (called AMUSE databases). Music Database is a folder with music files (mp3s or waves) - however it is also possible to work with the music files from other directories on your machine. Features are stored in Feature Database, processed features in Processed Feature Database. Annotation Database is a folder for ground truth annotations. In Model Database, the classification models are saved. The measures estimated during the validation routine are saved in Measure Database, and the optimization data is logged to Optimization Database.
- **EXTERNAL TOOLS**: Here you can fill in the paths to your Java, Matlab, and Python executables.
- **WAVE-SAMPLING SETTINGS**: The default option is to downsample the given mp3s or waves to 22050 Hz and run stereo-to-mono conversion.
- **MISCELLANEOUS SETTINGS**: The log level defines the minimal level of importance that messages should have to be displayed in the log screen. Setting this level to debug can help to see more information if any problems occur. The maximal number of task threads can be changed if you want to make use of several processing units in your machine.

A Index

Annotation The description of a certain aspect of a track that was created by external tools or humans (e.g. via the Annotation Editor)

Experiment A set of consecutive tasks

Feature The description of a certain aspect of a track that was obtained by AMUSE Feature Extraction tasks

Task One entry in the Experiment Configurator. Corresponds to one step from the classification pipeline as seen in Figure 1.

Track A piece of music

B Studies with AMUSE

C List of Available Features to Extract

D List of Available Feature Processing Methods

Step Name	Description	Id	Arguments
Tatum Pruner	Selects features only from windows with or between tatum times	0	Time windows to select: (t) - select only windows which contain tatum times; (b) - only windows which contain the exact middle between tatum times
NaN Eliminator	Replaces NaN-values with medians of the corresponding feature	1	None
Zero Mean-Unit Variance Normalization	Performs zero mean - unit variance normalization	2	None
Beat Pruner	Selects features only from windows with or between beat times	3	Time windows to select: (t) - select only windows which contain beat times; (b) - only windows which contain the exact middle between beat times
Data Sampler	Selects only each x-th feature	4	Either an int value x or: (b) for sampling so that the number of selected windows is equal to beat times number; (t) - equal to tatum times number; (o) - equal to onset times number
Principal Component Analyzer	Calculates the given percent of principal components of feature vectors	5	Percent of components to select
Interval Selector	Selects only the features from an interval from the middle or the beginning of the song	6	Interval length in milliseconds—For selection from the beginning of the song: (b); from the middle of the song: (m)
Onset Pruner	Selects features only from windows with or between onset times	7	(t) - select only windows which contain onset times; (b) - only windows which contain the exact middle between onset times
Derivation Calculator	Calculates 1st and 2nd derivations of feature vectors	8	Select if 1st derivation should be calculated—Select if 2nd derivation should be calculated
Normalization with Given Min/Max Values	Performs normalization so that 0 corresponds to the given minimal value of a feature and 1 to the given maximum value of a feature. Each feature dimension is treated separately	9	None
Running Mean Calculator	Calculates running means of feature vectors	10	Subset size for running mean calculation
AOR Splitter	Distinguishes between features for different AOR intervals (Attack/Onset/Release)	11	Attack interval start:—Attack interval middle:—Onset:—Release interval middle:—Release interval end

Figure 8: Available Feature Processing steps, stored in “processorAlgorithmTable.arff”

Example for a configuration of all steps:

```
0[t]-1-2-3[t]-4[15]-5[75]-6[2000_m]-7[t]-8[false_true]-9-10[10]-11[false_true_true_true_false]
```

E Available Matrix Conversion Methods

Step Name	Description	Id	Arguments
GMM1	Performs conversion of the given feature matrix with GMM1 model (saving mean value and deviation for each feature)	0	Save the mean—Save the standard deviation, e.g., false_true
AdaptiveOnsetGMM1	Performs conversion of the given feature matrix with GMM1 model (saving mean value and deviation for each feature) for partitions spanned between attack intervals starts and release interval ends	1	Save the mean—Save the standard deviation, e.g., false_true
Quartiles	Calculates quartile boundaries for each feature from the given feature matrix	2	None
StructureGMM1	Saves mean value and deviation for each feature selecting only the partitions from the middles of structural parts (e.g. refrain or bridge)	3	Number of partitions to select from the middle of each structural part
StructuralComplexity	Saves the statistics of the structural complexity of given feature vectors	5	First time scale in seconds (power of two)—Second time scale in seconds (power of two), e.g., [0_2]

Figure 9: Available matrix conversion nethods, stored in “processorConversionAlgorithmTable.arff”

F List of Availiable Classification Training Methods

G List of Availiable Validation Methods

H List of Availiable Optimization Methods

I List of Plugins

References

[1] Chollet, F., et al. Keras. `https://keras.io`, 2021. Accessed on: 01.03.2021.

[2] Frank, E., Hall, M. A., Holmes, G., Kirkby, R., and Pfahringer, B. WEKA - A machine learning workbench for data mining. In *The Data Mining and Knowledge Discovery Handbook*, O. Maimon and L. Rokach, Eds. Springer, 2005, pp. 1305–1314.

[3] Ginsel, P., Vatolkin, I., and Rudolph, G. Analysis of structural complexity features for music genre recognition. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)* (2020), IEEE, pp. 1–8.

[4] Guyon, I., Nikravesh, M., Gunn, S., and Zadeh, L. A., Eds. *Feature Extraction. Foundations and Applications*, vol. 207 of *Studies in Fuzziness and Soft Computing*. Springer, Berlin Heidelberg, 2006.

[5] Jolliffe, I. T. *Principal Component Analysis*. Springer, 2002.

[6] Mauch, M., and Levy, M. Structural change on multiple time scales as a correlate of musical complexity. In *Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR)* (2011), A. Klapuri and C. Leider, Eds., University of Miami, pp. 489–494.

[7] McEnnis, D., McKay, C., Fujinaga, I., and Depalle, P. jaudio: An feature extraction library. In *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR)* (2005), pp. 600–603.

[8] McFee, B., Raffel, C., Liang, D., Ellis, D. P. W., McVicar, M., Battenberg, E., and Nieto, O. Librosa: Audio and music signal analysis in python. In *Proceedings the Python Science Conference* (2015), pp. 18–25.

[9] Meng, A., Ahrendt, P., Larsen, J., and Hansen, L. K. Temporal feature integration for music genre classification. *IEEE Transactions on Audio, Speech, and Language Processing 15*, 5 (2007), 1654–1664.

[10] Mierswa, I., and Morik, K. Automatic feature extraction for classifying audio data. *Machine Learning Journal 58*, 2-3 (2005), 127–149.

[11] Rabiner, L., and Juang, B.-H. *Fundamentals of Speech Recognition.* Prentice Hall, Upper Saddle River, 1993.

[12] Vatolkin, I. Feature processing. In *Music Data Analysis: Foundations and Applications*, C. Weihs, D. Jannach, I. Vatolkin, and G. Rudolph, Eds. CRC Press, 2016, pp. 365–388.

[13] Vatolkin, I., Ginsel, P., and Rudolph, G. Advancements in the music information retrieval framework amuse over the last decade. In *Accepted for Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)* (2021).

[14] Vatolkin, I., Theimer, W., and Botteck, M. AMUSE (Advanced MUSic Explorer) - a multitool framework for music data analysis. In *Proceedings of the 11th International Society on Music Information Retrieval Conference (ISMIR)* (2010), J. S. Downie and R. C. Veltkamp, Eds., pp. 33–38.