AMUSE 0.3 beta
(Advanced MUSic Explorer)
User Manual

Igor Vatolkin

March 6, 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.

AMUSE is free software: you can redistribute it and/or modify it under the terms of the GNU Affero General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

AMUSE is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License [2] along with AMUSE.

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 installed on your system.

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 [7]
- **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 [5] 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 the following publication [7]: 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.

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. Some examples of the tasks are:

- FEATURE EXTRACTION: extraction of Mel frequency cepstral coefficients [6] using jAudio library [4]
- FEATURE PROCESSING: normalization of feature values to a given range, application of the principal component analysis [3]
- 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.

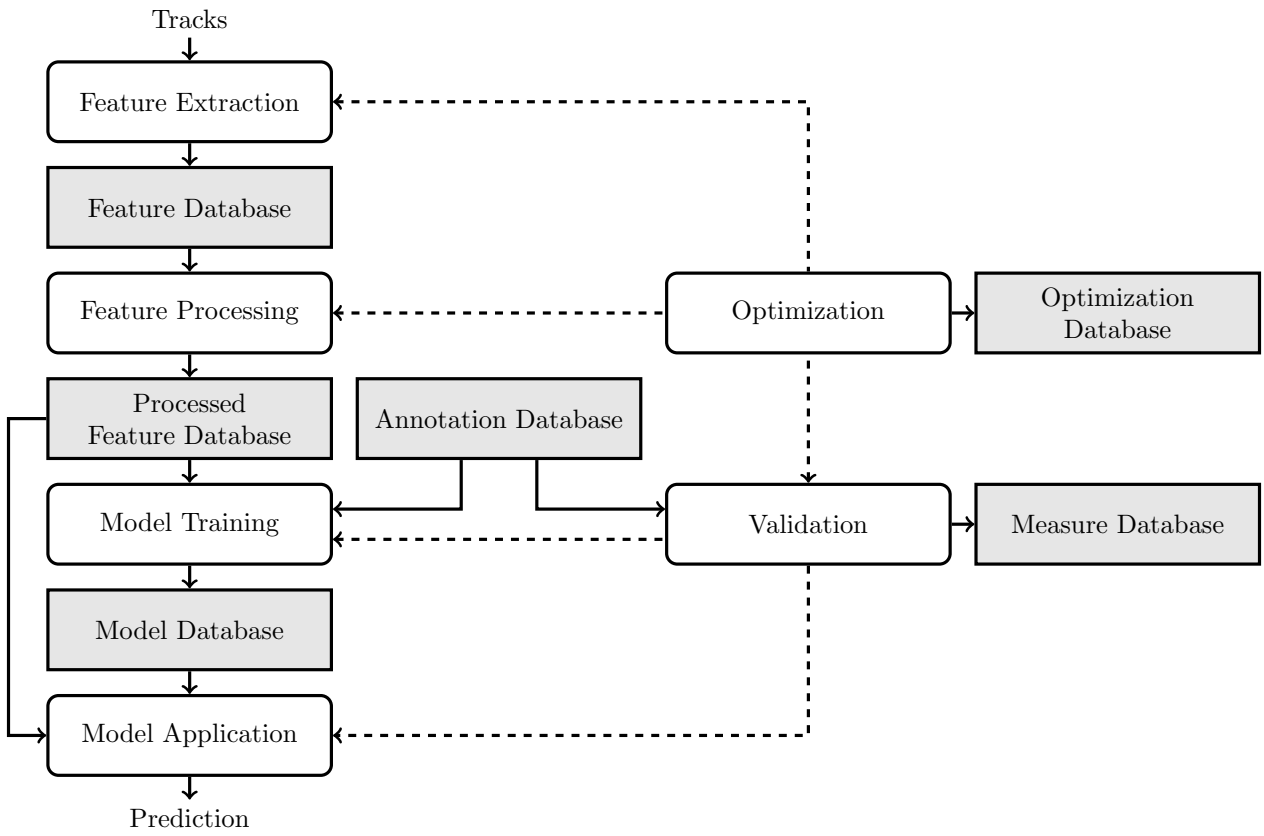

Figure 1: The classification pipeline in AMUSE.

2.1.1 Feature Extraction

2.1.2 Feature Processing

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 2: 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 3: 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

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] JOLLIFFE, I. T. *Principal Component Analysis*. Springer, 2002.

[4] 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.

[5] 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.

[6] RABINER, L., AND JUANG, B.-H. *Fundamentals of Speech Recognition*. Prentice Hall, Upper Saddle River, 1993.

[7] 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.