technische universität
dortmund

AMUSE (Advanced MUSic Explorer) - A Distributed Music Classification System Based on Signal Features

Igor Vatolkin, Wolfgang Theimer,
Martin Botteck, Günter Rudolph

Algorithm Engineering Report
TR09-2-001
Jan. 2009
ISSN 1864-4503

Faculty of Computer Science
Algorithm Engineering (Ls11)
44221 Dortmund / Germany
http://ls11-www.cs.uni-dortmund.de/

AMUSE (Advanced MUSic Explorer) – A Distributed Music Classification System Based on Signal Features

Architecture of a desktop/server execution environment

***Igor Vatolkin*[1], *Wolfgang Theimer*[2], *Martin Botteck*[2], *Günter Rudolph*[1]**

[1]Technische Universität Dortmund, Germany, [2]Nokia Research Center Bochum, Germany

Igor.Vatolkin@cs.uni-dortmund.de, Wolfgang.Theimer@nokia.com,
Martin.Botteck@nokia.com, Guenter.Rudolph@cs.uni-dortmund.de

CONTENTS

ABBREVIATIONS

AMUSE	**A**dvanced **MUS**ic **E**xplorer
DB	database

1. INTRODUCTION AND PURPOSE

The capacities of hard drives and mobile devices are growing permanently and accordingly the sizes of music collections owned by people. It becomes tougher to keep the overview and to browse large music collections intuitively. In the area of music categorization many automatic methods and approaches based on user reviews were established in the recent years. The drawback of community-driven approach is that classification is done subjectively by various people and cannot consider individual preferences. If the user would like to create and manage personal music categories (e.g. “nice music” or “summer holidays”), automatic feature extraction provides classification independent from media reviews.

AMUSE is a framework for automatic music classification which takes into account the music categories created by user. Extensibility of the framework plays a very important role. The drawback of many existing applications is that they are almost always perfect only for certain tasks. Several tools focus on feature extraction, whereas others deal with many different classification methods. Some tools are designed with the focus on high computational performance aspects, others are just easy to use. AMUSE provides the possibility to use several tools for music classification tasks and provides the advantage of distributed computing in a grid.

The following chapter describes the AMUSE architecture.

2. AMUSE ARCHITECTURE

In this chapter, the AMUSE architecture is described. The shapes of diagram nodes on the following figures correspond to their meaning:

- *Rectangle*: Component, application thread
- *Rectangle with unsharpened edges*: Processing step
- *Rectangle with shortened edge*: Data file
- *Cylinder*: Several files, database

2.1 Component Diagram

The global structure of the AMUSE is depicted in Figure Figure 2-1. The general task is to classify the given music files (input data) by algorithms and save the results (output data). The components and data flow are described more precisely in Figure Figure 2-2.

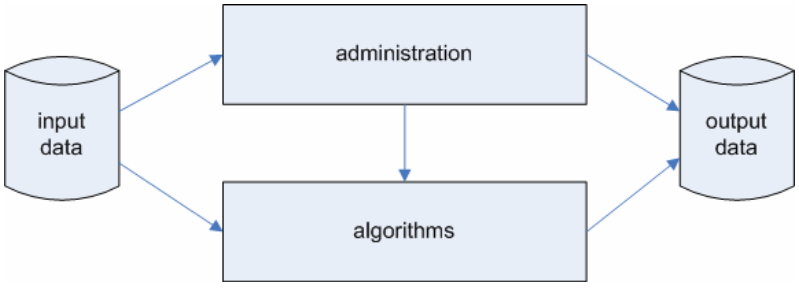

Figure 2-1: AMUSE components: Global view

Management of the classification is done by the user application (client side) and the scheduler (server side). The user application is an entry point. Here some music files can be selected (not necessarily located on the same machine, so we can speak about a distributed music database (DB)) and added to the file list managed on the server by the scheduler application. Music categories can be defined by user and their descriptions are added to the category list located on the server. Finally, a command to classify a given list of files or to show the results of previous classifications can be provided to the scheduler. The task of the scheduler is to coordinate the algorithms as well as to manage the file and category lists.

The algorithms needed for automatic music classification can be divided into three types: Feature extraction, feature pruning and classification. The way how each tool should be started (e.g. path to the tool folder) is described in the configuration database. Every algorithm task can be sent to a separate machine and the results are saved in different databases: Music features DB (one file for each feature of each music file), feature weights DB (they define the importance of different features for classification) and classification results DB (fuzzy mapping of music files to categories).

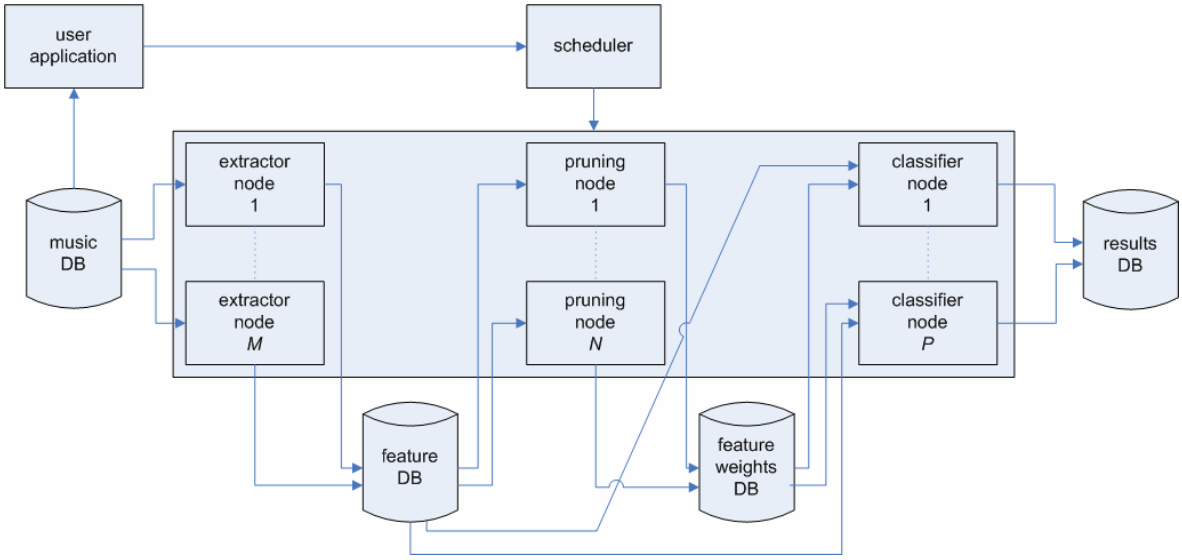

Figure 2-2: AMUSE components and data flow

2.2 File and Database Structure

2.2.1 File Formats

Beside the well-known XML format for configuration files, the ARFF format is used for the AMUSE databases. ARFF (Attribute-Relation File Format [WITTEN AND FRANK 2005]) is a text file format, which was developed at Waikato University, New Zealand. It is in the meantime implemented in numerous data mining tools and is easy to read and edit.

ARFF files generally contain a header and a data part. The header contains a comment describing the file, relation and attribute names. An example of an ARFF feature file with five MFCC coefficients is depicted in Figure Figure 2-3. Here attributes correspond to different features. The data part consists of strings with feature values delimited by commas; each string corresponds to one time window.

```
% First five mel frequency coefficients

@RELATION features

@ATTRIBUTE "MFCC0" NUMERIC
@ATTRIBUTE "MFCC1" NUMERIC
@ATTRIBUTE "MFCC2" NUMERIC
@ATTRIBUTE "MFCC3" NUMERIC
@ATTRIBUTE "MFCC4" NUMERIC

@DATA
-1.15E3,-8.527E-14,2.842E-14,-8.527E-14,5.684E-14
-1.15E3,-8.527E-14,2.842E-14,-8.527E-14,5.684E-14
-1.15E3,-8.527E-14,2.842E-14,-8.527E-14,5.684E-14
-1.15E3,-8.527E-14,2.842E-14,-8.527E-14,5.684E-14
-2.429E2,9.334E0,4.615E0,1.409E0,6.483E-1
-2.616E2,9.403E0,6.955E0,-2.429E0,-5.667E-1
-2.614E2,7.644E0,7.981E0,1.688E0,2.628E0
-2.596E2,8.874E0,6.441E0,2.305E-1,-5.564E-1
-2.596E2,8.692E0,8.657E0,7.637E-1,-2.817E0
-2.548E2,1.005E1,6.788E0,2.595E0,2.457E0
-2,587E2,8.369E0,8.069E0,5.322E-1,4.693E-1
```

Figure 2-3: ARFF file with feature values

2.2.2 Configuration Database

The information about the music data and its classification must be saved and administrated in the configuration database. The first task is to save and edit the information from the user application. This information consists of the editable list of music files for classification, where the unique file ID is mapped to the path and the name of music file. Another user-provided information is the list of music categories, which are also described by unique IDs. For learning of classifiers, some music files should be mapped to their categories. Additionally the classification management (we can describe it as the advanced user configuration) should be recorded in the database. It consists of the complete lists of features, feature extractors, pruners and classifiers.

Figure Figure 2-4 depicts the tables in the configuration database. The fileList table maps the unique music file ID to the path and the name of music file. The categoryList maps the unique music category ID to its description given by user. The fileToCategory table links music files described by their IDs to music categories with a given membership function. The featureList maps the unique feature ID to the description of the feature and the tool to extract this feature. The extractor tools are listed in extractorList, where the name of the XML file is given, which configures the extractor tool, as well as some additional configuration data.

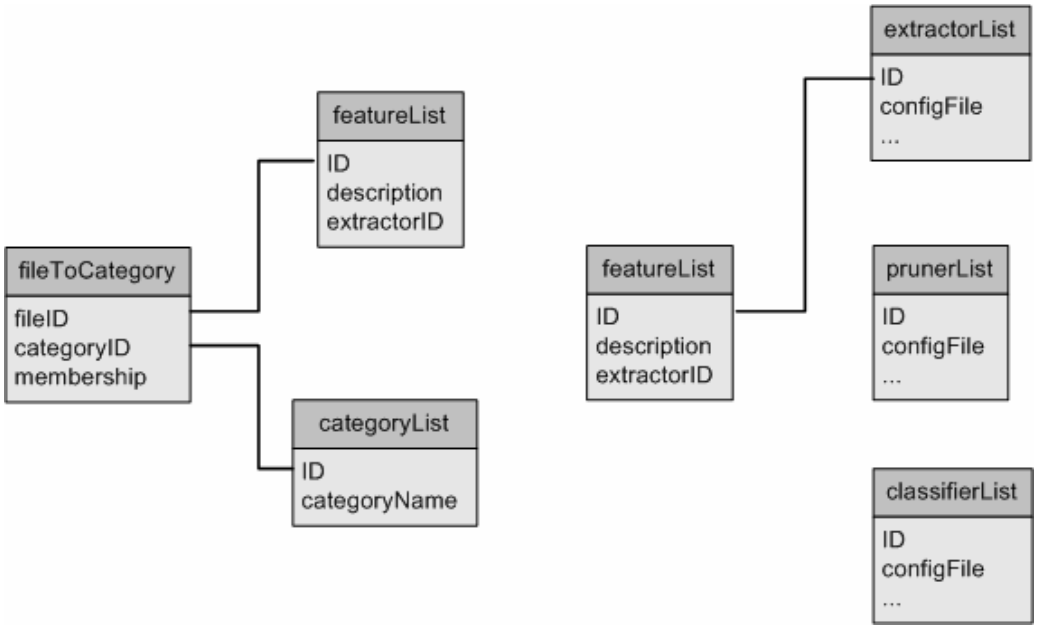

Figure 2-4: Configuration database

The tables are saved in ARFF files which can be manually edited. Examples of ARFF configuration tables are given in Figure Figure 2-5.

```
% Music feature table

@RELATION features

@ATTRIBUTE "ID" NUMERIC
@ATTRIBUTE "DESCRIPTION" STRING
@ATTRIBUTE "EXTRACTOR ID" NUMERIC

@DATA
0, "Root mean square", 0
1, "Spectral centroid", 0
2, "MFCC coefficients", 2
3, "Beats per minute", 0
4, "Chroma vector", 2
```

```
% Music file table

@RELATION musicfiles

@ATTRIBUTE "ID" NUMERIC
@ATTRIBUTE "PATH" STRING

@DATA
0, "/home/music/classic/Mozart"
1, "/home/music/classic/Bethoven"
2, "/home/music/oldies/Sinatra - Strangers In The Night"
3, "/home/music/oldies/Beatles - Yesterday"
4, "/home/music/progrock/Dream Theater - Pull Me Under"
```

Figure 2-5: Feature table and file table

The second task of the configuration database considers management of components for feature extraction, pruning and classification. Here no interaction with the user application is needed, but the scheduler has to know exactly how each tool can be started and what scripts should be generated for the tools. An example of a feature extractor configuration is given in Figure Figure 2-6. Here the unique ID, the name of the extractor, its folder, the folder of the script and the command to start are saved in the XML file.

```
<extractor id = "1">
        <entry>
          <name>name</name>
          <description>Name of extractor</description>
          <type>java.lang.String</type>
          <value>Yale</value>
        </entry>
        <entry>
          <name>homepath</name>
          <description>Path to extractor folder</description>
          <type>java.lang.String</type>
          <value>/home/tools/Yale</value>
        </entry>
        <entry>
          <name>homepath</name>
          <description>Path to script</description>
          <type>java.lang.String</type>
          <value>/home/scripts/yalefeatures.xml</value>
        </entry>
        <entry>
          <name>command</name>
          <description>Command to run the extractor from its folder</description>
          <type>java.lang.String</type>
          <value>/scripts/yale yalefeatures.xml</value>
        </entry>
</extractor>
```

Figure 2-6 Example of XML configuration table

2.2.3 Music Database

The music database is nothing else but a collection of music files in standard formats such as MP3 or wave. Since each file is saved with the path to it, files from several machines or available from the internet via http links can be classified.

2.2.4 Feature Database

Each feature database file contains the values of one feature for exactly one music file. A complex feature corresponds to several features which can be described together as one feature group, e.g. MFCC coefficients. Some features can be calculated only for the whole music file, other features are calculated for sample windows. An example of ARFF feature file with five MFCC coefficients was given above in Figure Figure 2-3.

2.2.5 Feature Weights Database

Feature weights describe the relevance of the individual feature types for a classification task: Features with low relevance can be suppressed in the pruning stage. The feature weights are saved in ARFF files which contain feature IDs and the corresponding weights as attributes. Additionally, the pruner ID is saved.

2.2.6 Results Database

The results database maps music files to categories like the table fileToCategory from the configuration database does. The first difference between results DB and the fileToCategory ARFF table is that results DB contains the output of the classifiers and not the training information from the user. The second difference is that the results of several classifiers can be saved. Therefore the classifier ID must be saved in the result table as well as the music file ID, the category ID and the membership rate (for a fuzzy classification).

2.3 Client-Server Communication

The user application controls the classification. The user can initiate several actions:

- *Add/remove files*: Adds/removes files from the music file list of the music DB.
- *Add/edit/remove categories*: Adds/edits/removes user defined categories (e.g. “jazz”, “nice music”, “summer holidays”...).
- *Add/remove relations between music files and categories*: Adds/removes information about the membership function of music files in music categories.
- *Start classification process*: Starts pruning and classification. These classification steps should be started after the definition of an adequate number of categories. In contrast, feature extraction is launched by the scheduler immediately after the update of the music file list.
- *Retrieve classification results*: Get the classification results from the results database and show and/or print them.
- *Delete classification results*: Delete all results in the results database.

2.4 Feature Extraction and Classification

2.4.1 Feature Extraction Node

The feature extraction node is depicted in Figure Figure 2-7. From a given music file, multiple features should be extracted and saved to the feature database. Since many feature extraction tools already exist and different tools compute different and partially disjunct feature sets, several tools can be started simultaneously or one after another. The purpose of each tool (termed “feature extractor” in the diagram) is to load the data from the music file, perform some preprocessing (e.g. reduce the sample frequency, convert stereo to mono..) and then extract the features from the sample data. The scheduler forwards to each extractor node the path to the music file for feature extraction and the extractor scripts. These scripts are generated once by scheduler for the whole feature extraction. They configure the features which should be calculated by a certain extractor tool. The extractor node scheduler sets the correct music file name for the scripts and the extractor tools read the appropriate music files. The decoder is started only once, so that extractors do not need to convert e.g. MP3 format to wave files any more.

The length of music signal windows required for the feature extraction differ significantly from feature to feature. For example, harmony analysis is based on tones and their relationship to each other. For determination of pitch the number of samples to analyze should not correspond to the music recording longer as the shortest available note. That's why windowing is applied, e.g. successive windows with 10 – 20 ms duration are taken from the music file and the features are calculated for every window. Often the windows overlap, e.g. by 50 percent. Other features such as mean energy or loudness can be calculated over the whole piece without any windowing.

Since different features may be calculated within different domains (for example, RMS in the time domain and the spectral centroid in the frequency domain), transforms may be applied before extraction. It is recommended to save computing time through the usage of

transform results for several features. For example, spectral centroid and spectral bandwidth can be calculated together based on a preceeding Fourier transform.

Every feature for each file should be saved in a separate file. In the strict sense several feature values can be combined to vectors and saved together if they have the same meaning, for example a set of MFCC features. The extracted features are saved in the ARFF format described in Chapter 2.2.1 and are fed into the consolidator. Since different tools may output the ARFF data in slightly different formats, the consolidator converts it to the unique AMUSE ARFF format and saves it in the feature database.

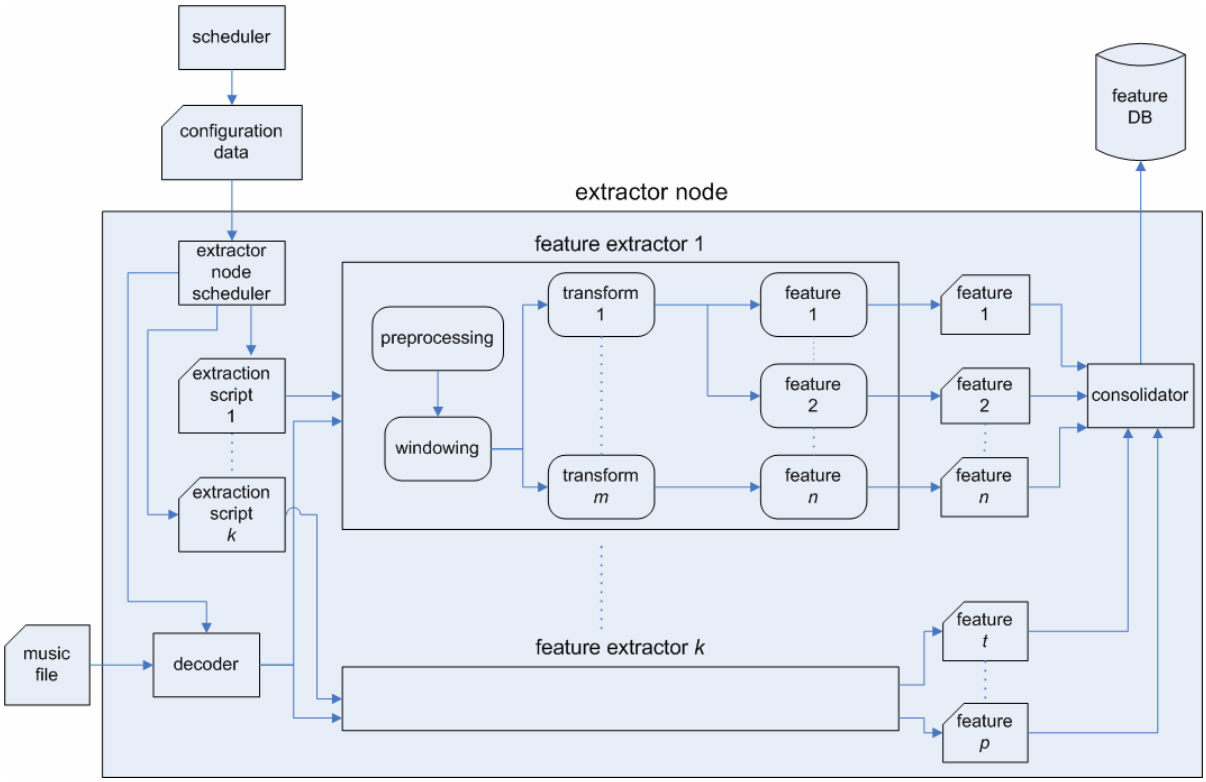

Figure 2-7: Data flow during feature extraction

As mentioned above, the extraction of different features is distributed between several extractors. Currently the extraction is done by Matlab and also two free Java tools – jAudio [McEnnis et al 2005] and Yale [Mierswa et al 2006], see also [Vatolkin and Theimer 2006]. An example of the basic extraction script for Yale is given in Figure Figure 2-8. The scheduler creates scripts for different tools due to the ARFF configuration table which maps features to extractors. In the shown Yale example script, transform and feature calculation operators can be enabled or removed using “AMUSE EnableTransform” and “AMUSE EnableFeature” operators, so that the original complete scripts for each tool can be easy modified to extract or not extract the given features.

```
<operator name="FeatureExtraction" class="Experiment">
  <parameter key="logfile"    value="logfile.log"/>
  <parameter key="logverbosity"     value="warning"/>
  <parameter key="random_seed"      value="2345"/>
  <operator name="Input" class="MusicPreprocessing">
    <parameter key="random_sample"  value="false"/>
    <parameter key="source_dir"     value="1.mp3"/>
    <parameter key="use_interval_attributes"    value="false"/>
    <parameter key="use_series_attributes"      value="true"/>
    <operator name="Preprocessing" class="OperatorChain">
      <operator name="MultivariateWindowing" class="MultivariateWindowing">
        <parameter key="overlap"    value="0.0"/>
        <parameter key="step_size"  value="512"/>
        <parameter key="window_size"      value="512"/>
        <operator name="Branching" class="Branching">
          <amuseEnableTransform>
            <parameter key="enable" value="false"/>
            <operator name="FreqDomain" class="OperatorChain">
              <operator name="FastFourierTransform" class="FastFourierTransform">
              </operator>
              <amuseEnableFeature id="14">
                <parameter key="enable"   value="false"/>
                <operator name="Centroid" class="Centroid">
                </operator>
              </amuseEnableFeature>
            </operator>
          </amuseEnableTransform>
          <amuseEnableFeature id="4">
            <parameter key="enable" value="false"/>
            <operator name="RMS" class="Average">
            </operator>
          </amuseEnableFeature>
        </operator>
      </operator>
    </operator>
  </operator>
  <operator name="ArffExampleSetWriter" class="ArffExampleSetWriter">
    <parameter key="example_set_file"     value="Centroid.arff"/>
  </operator>
</operator>
```

Figure 2-8: Yale script

2.4.2 Learning from Data

After the process of feature extraction is finished, the classification process can be started. Obviously one can extract a very large amount of features and it can be very difficult if not impossible to predict the importance of each feature for the upcoming classification tasks. The “brute force”' method just to include all possible features into the calculation implies huge classification time. But leaving out features without justification is also problematic. Both situations – using too many or too few features – can lead to suboptimal classification results. So the task of pruning is to pick out the optimal subset from the feature set for the given classification task and use it for the classification algorithms.

The classification algorithms must be trained on the basis of user information, which maps given music pieces to their categories. After the training the classification algorithms are ready to classify and can be validated on the test set which may be the part of training set, for example using the cross-validation explained in Chapter 2.4.4.

The steps after feature extraction are depicted in Figure Figure 2-9 and are described in detail in the following subsections.

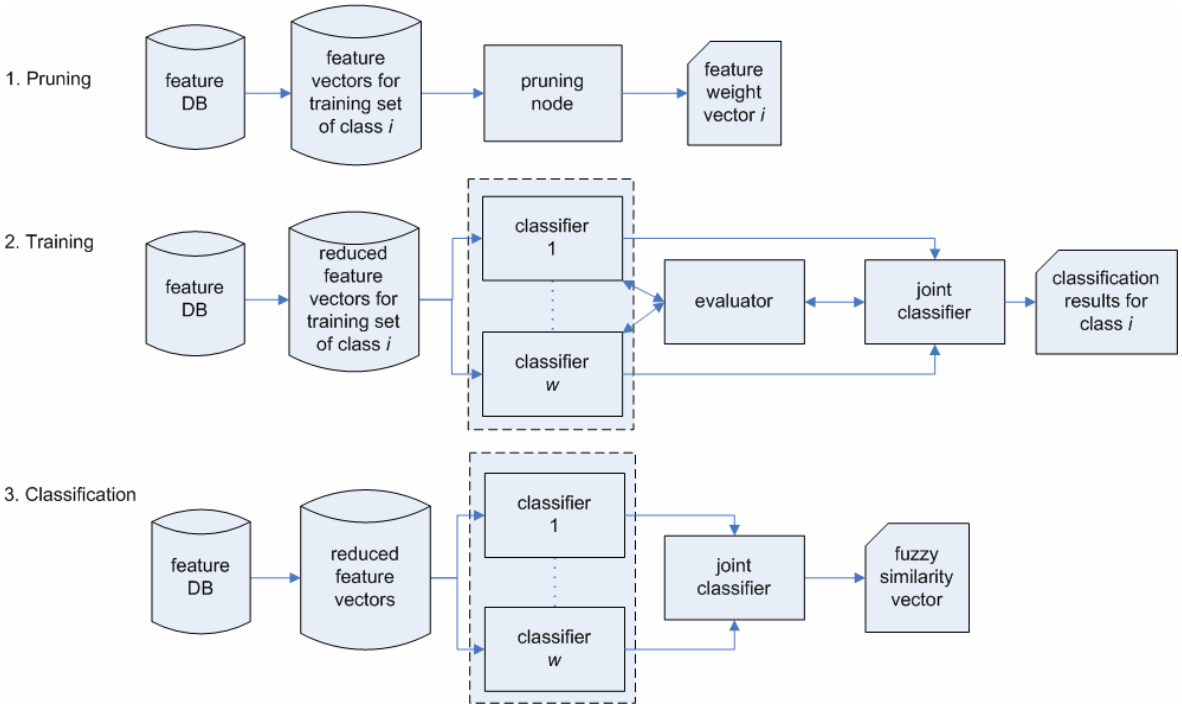

Figure 2-9: Data flow during classification steps

2.4.3 Feature Pruning

The task of feature pruning is to reduce the number of features used by classification algorithms without decreasing the correct classification rate. The best solution ever is the minimal set of features which allows the best classification, so that any other added feature does not influence the classification results (significantly) any more. For a given set of music files belonging to a certain category, a feature pruning node processes all previously extracted features and tries to determine the most important ones. The output is the weight vector which declares the importance of each feature. Basically it can be distinguished between “important”' and “not important” (weights equal to one resp. zero), or each feature is assigned a real value between 0 and 1 representing its importance. The ways how pruning tools work can be very different and three possible strategies are described here.

Principal Component Analysis

The first example of feature pruning system shown in Figure Figure 2-10 does not require any information about music categories and applies PCA (Principal Component Analysis [SMITH 2002]). The features are arranged as a multidimensional vector, with the reduction of dimension number as a goal. Eigenvectors correspond to the new coordinate axes, and the importance of these new axes (more important axis responds to the greater variance of data along the axis) is measured with eigenvalues, so that the new axes can be sorted by their importance. The axes with the smallest data variance along them can be discarted and the number of dimensions is reduced.

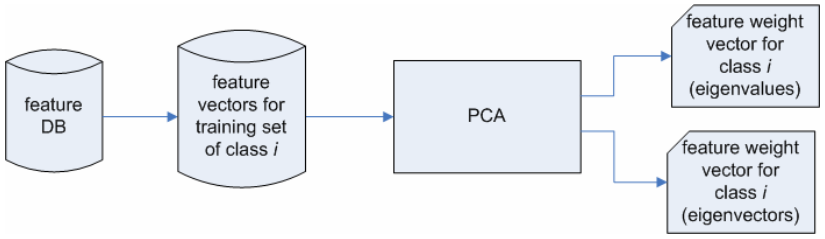

Figure 2-10: Pruning with PCA (data flow)

Evolutionary Algorithms

Another possibility to search for important features are EAs (Evolutionary Algorithms [HOLLAND 1975]). This pruning strategy interacts with a given classification algorithm and can not be clearly separated from it as PCA pruner. Initially, a solution (or a set of initial solutions) with certain feature weights (e.g. randomly set) is generated. Now the classifier is run and the results are evaluated. The classification success rate is used as the fitness function for the EA solutions (individuals of the population) representing the weight vector. The important design step is the choice of crossover and mutation operators for construction of new individuals. It is thinkable to let mutation set the certain group of features on/off with a given probability, for example switching between “use/do not use all tempo features”.

The process of generation of new solutions and their evaluation is repeated until the exit condition is fulfilled. The exit condition may be the number of generations (i.e. repetition steps) or the achievement of a certain success rate.

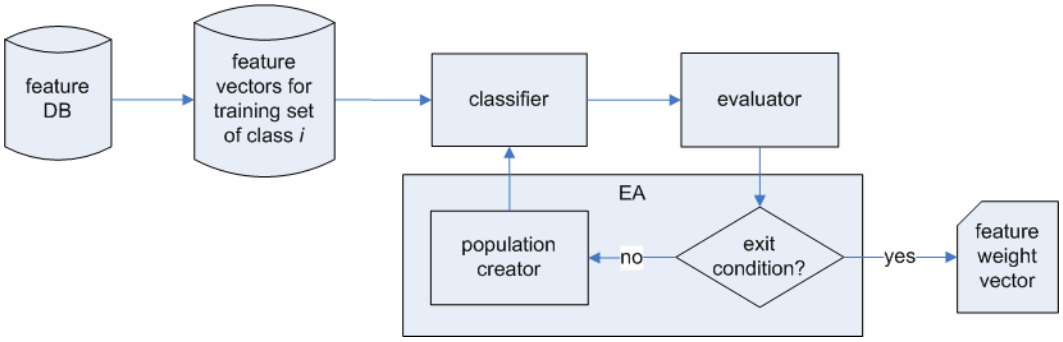

Figure 2-11: Pruning with EA (data flow)

Pareto Density Estimations

The last example introduced here is taken from MusicMiner [MÖRCHEN ET AL 2005]. Here PDEs (Pareto Density Estimations) are used to measure the likelihood of music pieces to the certain categories. Several quality measures are discussed, which are used to imply the feature weights. A specialist score considers the separation distance between a certain category and the other categories in the feature domain. An allrounder score measures the overall performance of the feature in separating all categories from each other.

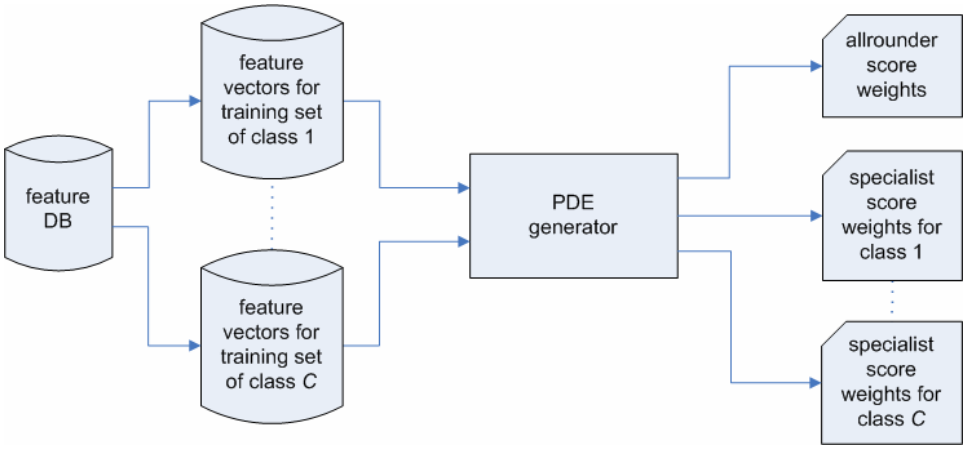

Figure 2-12: Pruning with PDEs (data flow)

2.4.4 Classification

After the pruning step, classification algorithms deal with the reduced feature set. XML files from the configuration database describe exactly the cooperation between pruners and classifiers, so that each classifier can be configured to use the results from a given pruner.

The goal of a training is to set up the classification algorithms for the certain tasks and classify correctly the given music pieces. The learning is based on the given music data described as belonging to one certain class. Several classifiers can be trained simultaneously. It is possible to run different strategies or to use the same classifier for different feature sets [FLEXER ET AL 2006]. The evaluator measures the success rate of the classifiers. A common evaluation technique is *m*-fold cross validation, where the data set is divided in *m* partitions. *m* -1 are used for training and the last one for validation. The whole process is repeated *m* times, so that every partition is used for validation once. After the training process, the joint classifier combines the results from the different classifiers. But it is also possible to separate test and training sets to avoid overfitting.

Now classification can be started. The reduced set of features serves as input for classification strategies, and the results are summed up to fuzzy similarity vector which contains the information about the membership rate of each music file in the defined categories.

2.5 Logging

Logging plays a very important role. Apart from debug and error logs it is always reasonable to save the information about the (recent) steps applied by different framework components.

Since AMUSE uses several tools, it should be distinguished between tool logs and the own AMUSE loggin messages. As depicted in Figure Figure 2-13 for the feature extraction node, the node logger collects the logs from each extractor and consolidates them with the node logs. The combined log file is available for the scheduler. For the pruner and classifier nodes, the logging works similarly. If the tools do not support logging, only AMUSE logs are created.

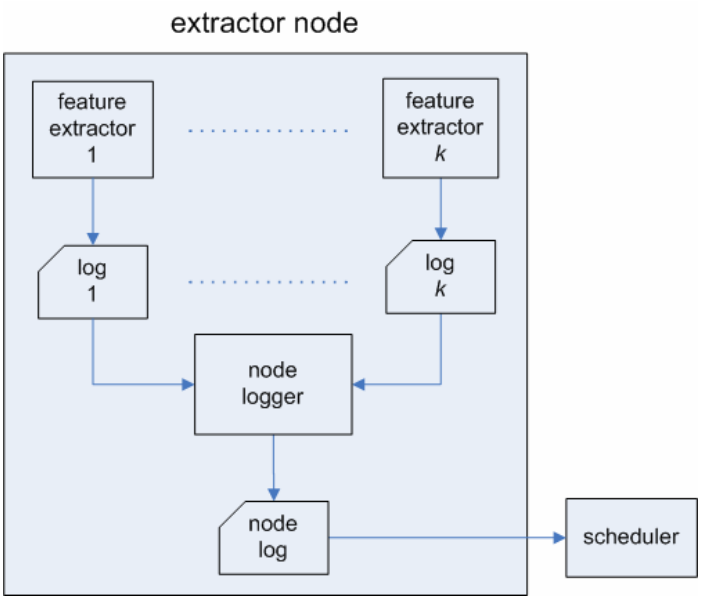

Figure 2-13: Logging steps (data flow)

Log4J [GÜLCÜ 2004] is a standard logging library for Java frameworks and is also used in the AMUSE. It enables to distinguish between several logging categories:

- *Debug*: Important only during the debugging and testing
- *Info*: Standard information output e.g. about initialization of components
- *Warn*: Warning indicates the occurrence of some error, which does not have any influence on the results of AMUSE work
- *Error*: Indicates the occurrence of some error, which has influence on the results of AMUSE work
- *Fatal*: Fatal error indicates the occurrence of critical error which causes AMUSE to exit

3. BIBLIOGRAPHY

1 [Flexer et al 2006]

Flexer, A., F. Gouyon, S. Dixon, and G. Widmer (2006). *Probabilistic Combination of Features for Music Classification*. In R. Dannenberg, K. Lemström, and A. Tindale (Eds.), Proceedings of the 7th International Conference on Music Information Retrieval (ISMIR), S. 111-114.

2 [Gülcü 2004]

Gülcü, C. (2004). *The Complete Log4j Manual.*

3 [Holland 1975]

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor: The University of Michigan Press.

4 [McEnnis et al 2005]

McEnnis, D., C. McKay, I. Fujinaga, and P. Depalle (2005). *JAudio: A Feature Extraction Library*. In Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR), S. 600-603.

5 [Mierswa et al 2006]

Mierswa, I., M. Wurst, R. Klinkenberg, M. Scholz, and T.Euler (2006). *YALE: Rapid Prototyping for Complex Data Mining Tasks*. In Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD). ACM Press.

6 [Mörchen et al 2005]

Mörchen, F., A. Ultsch, M. Thies, I. Löhken, M. Nöcker, C. Stamm, N. Efthymiou, and M. Kümmerer (2005). *MusicMiner: Visualizing Timbre Distances of Music as Topographical Maps*. Technical Report No. 47, Philipps-University Marburg, Germany.

7 [Smith 2002]

Smith, L. I. (2002). *A tutorial on Principal Components Analysis.*

8 [Vatolkin and Theimer 2006]

Vatolkin, I. and W. Theimer (2006). *Introduction of Methods for Automatic Classification of Music Data.* Submitted to TASLP Special Issue on Music Information Retrieval.

9 [Witten and Frank 2005]

Witten, I. H. and E. Frank (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgen Kaufman.