Machine Learning, 58, 127–149, 2005
2005 Springer Science + Business Media, Inc. Manufactured in The Netherlands.

Automatic Feature Extraction for Classifying Audio Data

INGO MIERSWA mierswa@ls8.cs.uni-dortmund.de
KATHARINA MORIK morik@ls8.cs.uni-dortmund.de
Artificial Intelligence Unit, University of Dortmund, Germany

Editor: Eamonn Keogh

Abstract. Today, many private households as well as broadcasting or film companies own large collections of digital music plays. These are time series that differ from, e.g., weather reports or stocks market data. The task is normally that of classification, not prediction of the next value or recognizing a shape or motif. New methods for extracting features that allow to classify audio data have been developed. However, the development of appropriate feature extraction methods is a tedious effort, particularly because every new classification task requires tailoring the feature set anew.

This paper presents a unifying framework for feature extraction from value series. Operators of this framework can be combined to feature extraction methods automatically, using a genetic programming approach. The construction of features is guided by the performance of the learning classifier which uses the features. Our approach to automatic feature extraction requires a balance between the completeness of the methods on one side and the tractability of searching for appropriate methods on the other side. In this paper, some theoretical considerations illustrate the trade-off. After the feature extraction, a second process learns a classifier from the transformed data. The practical use of the methods is shown by two types of experiments: classification of genres and classification according to user preferences.

Keywords: analysis of audio data, feature extraction, time series transformations, music recommender systems

1. Introduction

Since music is stored in digital form and distributed via the internet, there is a need for the management and retrieval of audio data. How can we index large numbers of audio records? How can we structure music databases according to genre (e.g., classic, pop, hip hop) or occasions (e.g., dinner, party, wedding)? How can a system automatically recommend music records to users? Information retrieval has started several efforts to automatic indexing (Kurth & Clausen, 2001) and retrieval (e.g., querying by humming (Ghias et al., 1995)). Machine learning has shown its benefits for text classification and ranked document retrieval with respect to user preferences (Joachims, 2002a, 2002b). It is straightforward to expect a similar benefit for the classification and personalized retrieval of music records.

Confronted with music data, machine learning encounters a new challenge of scalability:

– Music databases store millions of records.
– Given a sampling rate of 44100 Hz, a three minute music record has a length of about $8 \cdot 10^6$ values.

Moreover, current approaches to time series indexing and similarity measures rely on a more or less fixed time scale (Keogh & Smyth, 1997; Keogh & Pazzani, 1998). Music plays, however, differ considerably in length. More general, time series similarity is determined with respect to some (flexible and generalized) shape of curves (Yi, Jagadish, & Faloutsos, 1998; Kahveci & Singh, 2001). However, the shape of the audio curve does not express the crucial aspect for classifying genres or preferences. The i-th value of a favourite song has no correspondence to the i-th value of another favourite, even if relaxed to the $(i \pm n)$-th value. The decisive features for classification have to be extracted from the original data. Some approaches extract features from music given in the form of Midi data, i.e. a transcription according to the 12 tone system (Loy, 1989).[1] This allows to include background knowledge from music theory. Usually, music data is given in the form of—possibly compressed—wave records, the audio data. Hence, feature extraction from audio data has become a hot topic recently (Liu, Wang, & Chen, 1998; Zhang & Kuo, 1998; Guo & Li, 2003; Tzanetakis, 2002). Several specialized extraction methods have shown their performance on some task and data set. It is now hard to find the appropriate feature set for a new task and data set. In particular, the problems are:

Unifying framework missing: The large set of extraction methods has not yet been systematically investigated, a unifying framework is still missing. This makes it hard to compare the proposed feature sets and to detect missing feature extraction methods.

Variety of feature sets: Different classification tasks ask for different feature sets (see Section 4). It is not very likely that a feature set delivering excellent performance on the separation of classical and popular music works well also for the separation of techno and hip hop music. Classifying music according to user preferences even aggravates the problem.

Large search space for feature sets: There is no concisive feature set from which we would select an appropriate subset for a new task and data set by standard wrapper approaches (Kohavi & John, 1997). Even if it existed, it would be most cumbersome to enumerate it.

In this paper, we present our approach to tackle the problems. We present a unified framework for extraction methods in Section 2. The framework covers the known methods, and several new ones have been added. The repository of elementary extraction operators allows us to handle feature extraction as a sequence of data transformations which delivers a feature set in the end. Hence, we construct a feature set for each given task and data set, anew. Since it would be tedious to do so by hand, we apply a learning algorithm to construct the feature set for us. Section 3 describes the genetic programming approach to the automatic construction of (nested) sequences of data transformations, the method trees. The search within the universe of method trees is guided by a fitness function. Here, we embed a classification learner: the better the learning result using the transformed data, the higher the fitness of the feature set (i.e., the method tree). Genetic programming puts together the building blocks of feature extraction operators according to the targeted classification task and data set. It outputs a feature extraction method tree. Applying a method tree to the given audio data delivers a transformed data set, i.e., the examples rewritten by the corresponding feature set. This becomes the input to a second learning step, namely

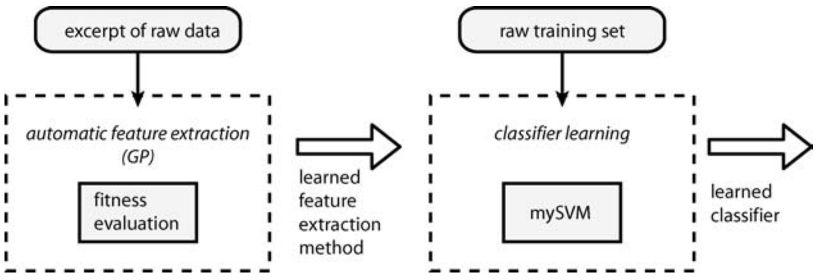

Figure 1. The overall process of automatic feature construction for classification.

classifier learning. Figure 1 shows the overall process with the two learning steps, one using genetic programming, the other using the support vector machine mySVM (Rüping, 2000) for classifier learning. Please note that the learning scheme used in the second learning step is also part of the feature extraction training. In contrast to the second learning step described here the embedded learning scheme works on an excerpt of the examples to estimate the accuracy and provide a fitness value (fitness evaluation). Further details are explained in Sections 3 and 4. The approach is tested on the learning tasks of genre classification and user preferences (Section 4).

2. Methods for feature extraction

Audio data are time series, where the y-axis is the current amplitude corresponding to a loudspeaker's membrane and the x-axis corresponds to the time. They are univariate, finite, and equidistant. We may generalize the type of series which we want to investigate to *value series*. Each element x_i of the series consists of two components. The first is the *index component*, which indicates a position on a straight line (e.g., time). The second component is a m-dimensional vector of values which is an element of the *value space*.

Definition 1. A VALUE SERIES is a mapping $x : \mathbb{N} \rightarrow \mathbb{R} \times \mathbb{C}^m$ where we write x_n instead of $x(n)$ and $(x_i)_{i \in \{1,\ldots,n\}}$ for a series of length n.

This general definition covers time series as well as their transformations. All the methods described in the following refer to value series. They are not only applicable to audio data, but to value series in general. The usage of a complex number value space instead of a real number value space allows a convenient way to use basis transformations like the Fourier transformation. Finally, the introduction of the index component allows both equidistant and non-equidistant value series.

Feature extraction methods for value series can be described in a general framework. Such a framework offers the following advantages:

– Missing features for classification can be detected and their extraction methods be developed.

– Specialized methods can be decomposed into their general extraction methods.
– The repository of general extraction methods can easily be implemented and extended.
– Combinations of extraction operators can be built, generating a large variety of feature sets.

In other words, organizing a repository of elementary feature extraction methods allows us to see the feature extraction for a certain learning task as a sequence of methods. The known methods are fixed sequences of such elementary extraction methods. Here, we give an overview of the building blocks, so that we later on can flexibly construct sequences. Details are omitted.[2] Only the new notions of general windowing and interval mark-up are presented in more detail.

2.1. Basis transformations

Basis transformations map the data from the given vector space into another space. Audio data—like all univariate time series—are originally elements of the vector space $\mathbb{R}^2$. The basis B of a vector space V is a set of vectors which can represent all vectors in V by their linear combination. The only required operation on vector spaces as the domain of transformations is the scalar product. Since the most common basis transformation performed on audio data is the transformation into the infinite space of harmonic oscillations we assume *Hilbert spaces*.

Definition 2. Let H be a vector space with an inner product $\langle f, g \rangle$. H is called HILBERT SPACE if the norm defined by $|f| = \sqrt{\langle f, f \rangle}$ turns H into a complete metric space, i.e. any Cauchy sequence of elements of the space converges to an element in the space.

The assumption of Hilbert spaces is no constraint, because all finite-dimensional spaces with a scalar product (such as Euclidean space with ordinary scalar product) are Hilbert spaces. However, we use Hilbert spaces with an infinite number of dimensions to introduce the concept of Fourier transformations. Therefore, we need an infinite-dimensional Hilbert space of functions.

Definition 3. Let P be a Hilbert space. If the elements $f \in P$ are functions, P is called a FUNCTION SPACE.

Example 1. The set of all functions $f : \mathbb{R} \to \mathbb{R}$ with a finite integral

$$\int_{-\infty}^{\infty} f^2(x)\, dx$$

together with the inner product

$$\langle f, g \rangle = \int_{-\infty}^{\infty} f(x) g(x)\, dx$$

form a well known function space: L^2.

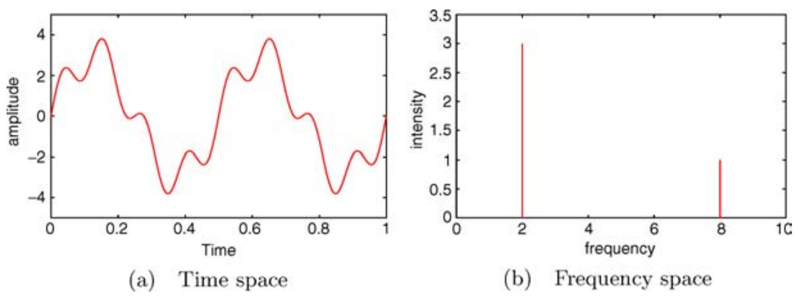

Figure 2. Overlay of two curves, $\nu_1 = 2\,\text{Hz}, a_1 = 3$ and $\nu_2 = 8\,\text{Hz}, a_2 = 1$, shown left in time space, right in frequency space after a Fourier transformation.

2.1.1. Frequency space. The goal of *Fourier analysis* is to write the series $(x_i)_{i\in\{1,\ldots,n\}}$ as a (possibly infinite) sum of multiples of the given base functions, which are $e^{i\nu x}$ for all frequencies ν. A Fast Fourier Transformation (Cooley & Tukey, 1965) maps the given time space into this frequency space and is valid for audio data (figure 2). The frequency space is a special case of a function space. Therefore, the transformation uses the infinite number of complex valued dimensions of a Hilbert space. Complex numbers are necessary because Fourier transformations actually deliver two values: the intensity of occurring frequencies and the phase shifts.

2.1.2. Correlation space. The frequency space expresses a sort of correlation between values in terms of frequencies. For some features it would be more appropriate to express the correlation in terms of time dependencies. Therefore, the transformation into another space is used.

Definition 4. The calculation of correlations of values between two points in time, i and $i + k$, produce the CORRELATION SPACE, where for each lag k their correlation coefficient in $[-1, +1]$ is indicated.

Transforming audio data into the correlation space eases the recognition of the speed of the music, measured in beats per minute. Assuming T is the number of beats per measure and SR the sampling rate. If we shift the original time series by *shift* $= T \cdot SR \cdot 60/X$ for several values of X we can determine the correlation between the original and the shifted time series. Maximal correlation corresponds to minimal difference between the shifted and the original series. Figure 3 shows the differences of original values with the shifted ones. Clearly, the difference at 97 beats per minute is minimal.

2.1.3. Reconstruction of state space. Nonlinear dynamic systems can be described with the aid of non-linear differential equations. The number of variables which must be known to completely describe the behavior of such a system corresponds to the dimension of this system. These variables are called *state variables*.

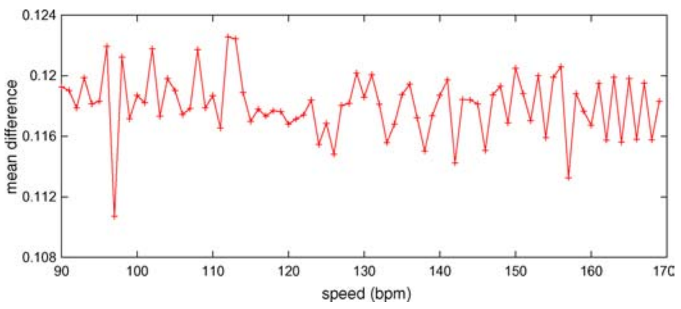

Figure 3. Autocorrelation differences for a phase shift depending on speeds ranging from 90 to 170 beats per minute.

Definition 5. The basis of the STATE SPACE of a dynamic system is given by the STATE VARIABLES of the system, i.e. the variables which must be known to describe the system. The elements of a state space represent the values of the state variables at the examined (time) points.

The *state space* emphasizes characteristics which can hardly be seen in the original space. Since the state variables are often unknown, a topologically equivalent space is constructed (Takens, 1980). This is known as *reconstruction of state space*.

Definition 6. Vectors within PHASE SPACE are constructed, where the components are parts of the original series:

$$\mathbf{p}_i = \left(x_i, x_{i+d}, x_{i+2d}, \ldots, x_{i+(m-1)d}\right)$$

where d is the delay, and m the dimension of the phase space. The set

$$P_{d,m} = \{\mathbf{p}_i \mid i = 1, \ldots, n - (m-1)d\}$$

is the phase space representation of the original series $(x_i)_{i \in \{1,\ldots,n\}}$.

Within the phase space, several features can be extracted, e.g., the angles between vectors. Small variances of angles indicate smooth changes of the state variables, large variances harsh changes. This is a dominant feature when separating classic from the more percussive pop music as shown in figure 4.

2.1.4. Reversibility. Basis transformations most often are reversible, because only the basis, not the position of the elements is changed. In contrast, if intervals in the index dimension are used, the transformation is not reversible. If, for instance, we summarize the original series by some time intervals and assign a value to each interval, the transformed

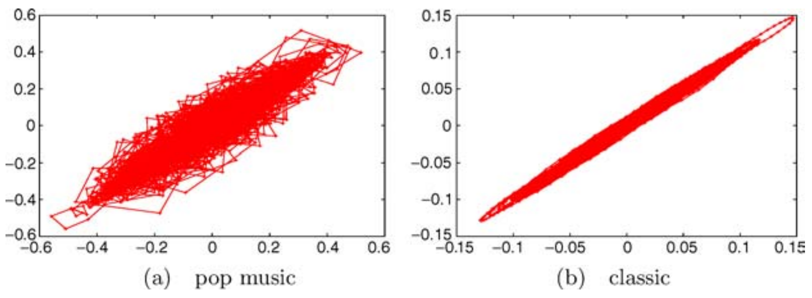

Figure 4. Phase space representation of a popular song (left) and a classical piece (right).

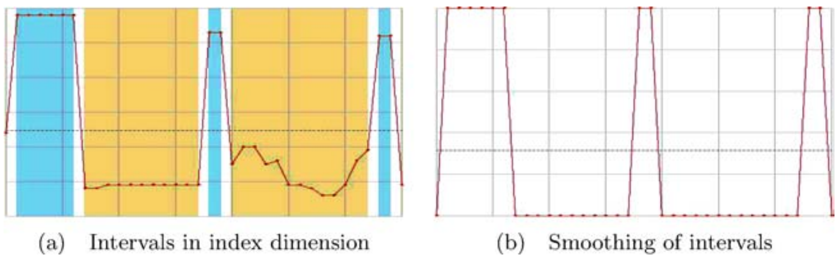

Figure 5. Intervals found in the index dimension are summarized.

series has still the same number of elements but fewer different values (see figure 5 for illustration). We will discuss some possible ways to detect intervals in different dimensions of a value series in Section 2.3.

2.2. *Filters*

Filters transform elements of a series to another location within the same space. Moving average and exponential smoothing, for instance, are filters. Many known transformations are subsumed by weighting functions. We consider the window functions Bartlett, Hanning, Hamming, Blackman-Harris, linear and exponential functions as particular instances of a function $f_w(i)$ which weights the position within the window.

Definition 7. Given a value series $(x_i)_{i\in\{1,\ldots,n\}}$, a filter $y_i = f_w(i) \cdot x_i$ is a WEIGHT FILTER. The weighting function f_w only depends on the position i.

Other filters are the frequency passes, filtering the extremes, the Bark-filter, and the ERB filter, which are all often used when analyzing music data.

2.3. Mark-up of intervals

In analogy to mark-up languages for documents, which annotate segments within a text, also segments within a time series can be annotated.

Definition 8. A MARK-UP $M : S \rightarrow C$ assigns an arbitrary characteristic C to a segment S.

We define special instances of mark-up by assigning characteristic value types to intervals in one of the dimensions of the considered space.

Definition 9. An INTERVAL $I : S \rightarrow C$ is a mark-up within one dimension. The segment $S = (d, s, e)$ is given by the dimension d, the starting point s, and the end point e. The characteristic $E = (t, \varrho)$ indicates a type t and a density ϱ.

Often clustering (e.g., k-means) is used in order to detect suitable intervals (Hastie, Tibshirani, & Friedman, 2001). A clustering scheme is only usable in dimensions with a non-equidistant value distribution. Additionally, clustering in one or several dimensions is a batch process to be applied to the complete series. An incremental process is the signal to symbol process (Morik & Wessel, 1999):

Signal to symbol processing: Given the series $(x_i)i \in \{1, \ldots, n\}$ with n values, a decision function f_e and an interval dimension,
initialize the interval counter with $t = 1$, start a new interval I_t and add the first point.

For the remaining points of the series, do:

1. If $f_e(I_t, x_i) = 1$, then add x_i to the current interval I_t.
2. Else close I_t, increase t by 1, and add x_i to the new I_t.

Typical examples of the decision function f_e refer to the gradient, delivering characteristics such as, e.g., *increase, decrease*. Signal to symbol processing is applied to the index dimension (time). If intervals have already been found in the value dimension, these can be used to induce intervals in the index dimension. For instance, whenever a interval change in the value dimension has been found, the current interval in the index dimension is closed and a new one is started. Figure 6 illustrates this combination.

2.4. Generalized windowing

Many known operators on times series involve windowing. Separating the notion of windows over the index dimension from the functions applied to the values within the window segment allows to construct many operators of the kind.

Definition 10. Given the series $(x_i)_{i\in\{1,\ldots,n\}}$, a transformation is called WINDOWING, if it shifts a window of width w over $(x_i)_{i\in\{1,\ldots,n\}}$ using a step size s and evaluates in each window

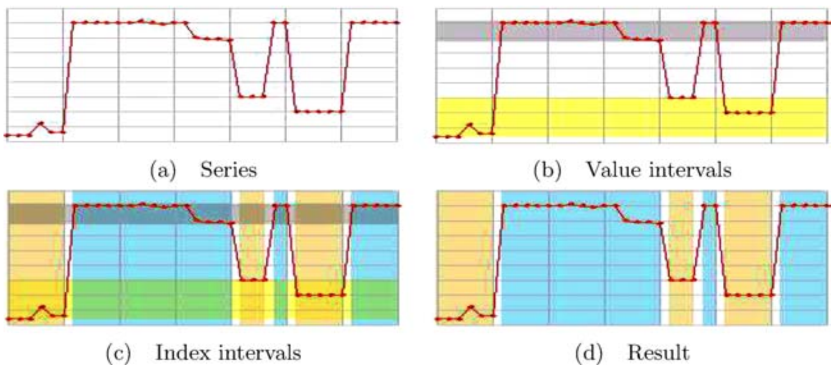

Figure 6. The process of finding intervals in a series (a), first in the value dimension (b), then projected on the index dimension (c), delivering (d).

the function F:

$$y_j = F\big((x_i)_{i\in\{(j\cdot s+1,\ldots,j\cdot s+w\}}\big).$$

All y_j together form again a series $(y_j)_{j\in\{0,\ldots,\lfloor(n-w)/s\rfloor\}}$.

Definition 11. A windowing which performs an arbitrary number of transformations in addition to the function F is called GENERAL WINDOWING.

The function F summarizes values within a window and thus prevents general windowing from enlarging the data set too much. Since the size of audio data is already rather large, it is necessary to consider carefully the number of data points which is handled more than once.

Definition 12. The OVERLAP of a general windowing with step size s and width w is defined as $g = w/s$.

Only for windowings with overlap $g = 1$ the function can be omitted. Such a windowing performs transformations for each window and is called *piecewise filtering*. Section 2.6 investigates the runtime effects of transformations used within general windowing and the overlap.

Combining general windowing with the mark-up of intervals allows to consider each interval being a window. This results in an adaptive window width w and no overlap, i.e. $g = 1$. Of course, this speeds up processing considerably.

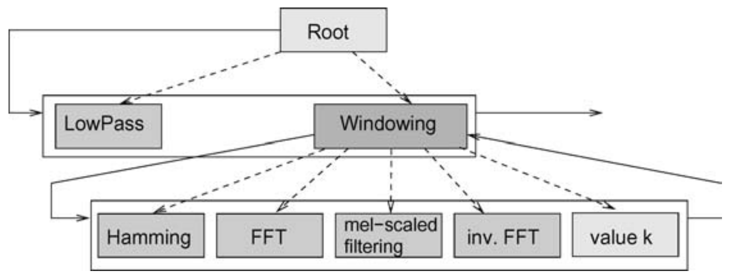

Figure 7. Constructing the cepstral method from elementary extraction operators.

2.5. *Functions*

Transformations convert a series into another series. In contrast, functions calculate single values from a series. The group of functions includes all kinds of statistics like different averages, variance and standard deviation. They refer to the value dimension. We may also consider the index dimension, for instance, the point with the largest value or highest amplitude. Often used functions are those indicating peaks.

Definition 13 (k-Peaks function). The k-Peaks function delivers for a series $(x_i)_{i\in\{1,\ldots,n\}}$ the position (index dimension), the height, and the width of the k largest peaks.

It is an instance of finding extremes (minimum, maximum). Similarly, the gradient of a regression line can be formulated. For audio data, the spectral flatness measure or the spectral crest factor can be expressed as an arithmetic combination of simple functions (Jayant & Noll, 1984). The *mel-frequency cepstral coefficients* can be constructed as a general windowing, where the frequency spectrum of the window is calculated, its logarithm is determined and a psychoacoustic filtering is performed, and the inverse Fourier transformation is applied to the result. Figure 7 shows how the methods for feature extraction are put together to compute the cepstral coefficients. From these coefficients additional features can be extracted. It is easy to see how variants of this series can be generated, e.g., replacing the frequency spectrum and its logarithm by the gradient of a regression line.

2.6. *Some properties of the methods*

Since large amounts of data have to be processed, each method must be fast. Of course, the mere sequence of methods is then fast, too. What needs to be checked is the effect of general windowing. How much does the runtime increase, if we apply methods running in $O(n^2)$ within the windows? How many data points are handled more than once? If the step size s is 1, k windows of width $n - k$ can be built. The number of values to be processed increases from n to $k \cdot (n - k)$. The value $k = n/2$ maximizes this product. In other words, for a step size of 1 the worst case is a window size of $w = n/2$ for a series of length n. Then $g = n/2$ values are handled more than once.

In general, the number of windows for a series of length n and a step size s is

$$k = \frac{n-w}{s} + 1 = \frac{n}{s} - g + 1 \tag{1}$$

We assume that all transformations occur only once in a sequence of methods and that the number of methods is finite. The resulting runtime for methods in $O(n)$ is generally stated by Lemma 1.

Lemma 1. *General windowing using methods in $O(n)$ has the worst case runtime of $O(gn - gw + w)$, where g is the overlap.*

Proof: Processing data within a window costs O(w). k windows are to be processed. The overall runtime is $k \cdot w$:

$$\begin{aligned} k \cdot w &= \left(\frac{n}{s} - g + 1\right) \cdot w \\ &= \frac{n}{s}w - gw + w \\ &= gn - gw + w \end{aligned} \qquad \square$$

Lemma 1 means, that for $g = 1$, a series is processed in $O(n)$. For $g = 2$ the effort becomes $2n - w$ which is more expensive than processing the methods on the overall series since w is always less than n. If $g = n/2$, the general windowing effort becomes $\frac{n^2}{2} - \frac{wn}{2} + w$ which is an order of magnitude larger than the runtime of the inner methods applied to the whole series.

Analogously, we state the runtime for general windowing for methods with logarithmic complexity, with quadratic complexity, and those with exponential complexity.

Lemma 2. *Assuming that each method is applied at most once, general windowing using methods in $O(n \log n)$ has the worst case runtime of $O(gn \log w - gw \log w + w \log w)$, where g is the overlap, n the length of the series, and w the width of the windows.*

Lemma 3. *Assuming that each method is applied at most once, general windowing using methods in $O(n^2)$ has the worst case runtime of $O(gnw - gw^2 + w^2)$, where g is the overlap, n the length of the series, and w the width of the windows.*

Lemma 4. *Assuming that each method is applied at most once, general windowing using methods in $O(n^p)$ has the worst case runtime of $O(gnw^{p-1} - gw^p + w^p)$, where g is the overlap, n the length of the series, and w the width of the windows.*

Lemma 5. *Assuming that each method is applied at most once, general windowing using methods in $O(a^n)$ has the worst case runtime of $O(\frac{gn}{w}a^w - ga^w + a^w)$, where g is the overlap, n the length of the series, and w the width of the windows.*

We elaborate on the most common overlap of 2 and the worst of $n/2$. The result indicates, when to use windowing and when to process the overall series. For all but the linear complexity methods, the general windowing is faster than applying the methods to the overall series, if $g = 1$, and at least as fast if $g = 2$. If $g = n/2$, the runtime for windowing is worse than the runtime for applying the methods to the overall series except for exponential inner methods.[3]

3. Automatic construction of method trees

The elementary methods described above are combined in order to construct more complex features for classification tasks. Figure 7 already showed how elementary methods can be used for the reconstruction of known complex feature extraction methods. There are many more complex feature extraction methods which can be built using the framework described above (Section 2). For instance, the general windowing may apply a Fourier transformation so that the peaks of the transformed series can be related with windows in time:

$$y_j = \max_{\text{index}} \left(FT\left(\{x_i\}_{i \in \{j \cdot s+1, \ldots, j \cdot s+w\}} \right) \right)$$

The result is a value series, where the value of y_j denotes the highest frequency for each window. From this series, the average and variance is built, yielding a good feature for the separation of techno and pop music—the variance is greater in pop music.

It is rather cumbersome to find such combinations that perform well for a classification task. We are looking for chains of method applications. Moreover, there might be some windowing within which such chains are applied. This is a rather large search space (see Section 3.3). It is too large to be inspected manually. Hence, *genetic programming* is applied in order to look for the best combination of methods (Holland, 1986; Koza, 1992). The result is a complex method. Its use for the classifier learning will be shown in Section 4.

In order to structure the huge search space, we may separate functions, chains of method applications, and general windowing, where a chain of method applications is applied to each window.

Definition 14. A CHAIN consists of an arbitrary number of transformations and a function at the end.

A function is a chain with no transformations. It has the length 1. A longer chain consists of some transformations followed by a function. In any case, a chain delivers one value. Incorporating windowings leads to the concept of method trees:

Definition 15. A METHOD TREE is a general windowing whose children build a chain. If the chain entails a windowing, this becomes the root of a new, embedded method tree.

The methods which are performed on each window can be seen as children of the windowing operator. Together they output a value series. The tree structure emerges from the nesting of windowing operators.

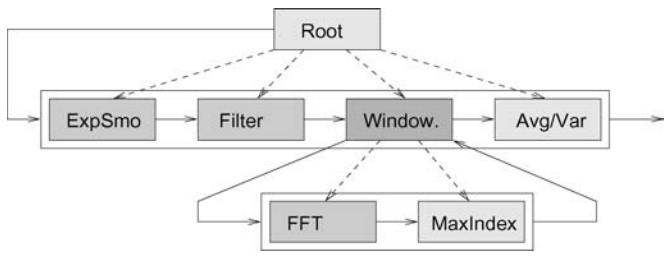

Figure 8. A method tree for feature extraction built of elementary methods. Solid arrows show the data flow, dashed lines define the tree structure.

An example of a method tree is shown in figure 8, where the root identifies the element within the search space. Its four children are exponential smoothing, a filtering, another method tree consisting of the chain just described (Fourier transformation with peaks applied to windows), and the average of the peaks. This last child returns the desired features.

Before the genetic programming approach is technically described, figure 9 presents the process of automatically extracting features for a given classification task and data set. The picture details on the first box of figure 1 above which shows the overall process. The search space within which the best method tree is to be found is called the universe of method trees. A population is a set of method trees. The navigation within the universe of method trees is a cycle of selecting a population, applying the method trees to the raw data, evaluating the fitness of the population, and enhancing the fittest method trees further to build a new population (Section 3.2). This cycle corresponds to the standard process of genetic algorithms. What differs from the standard is that method trees instead of binary vectors form the search space, that the search space is structured, and that the fitness evaluation is not merely a function but the result of running another learning algorithm.

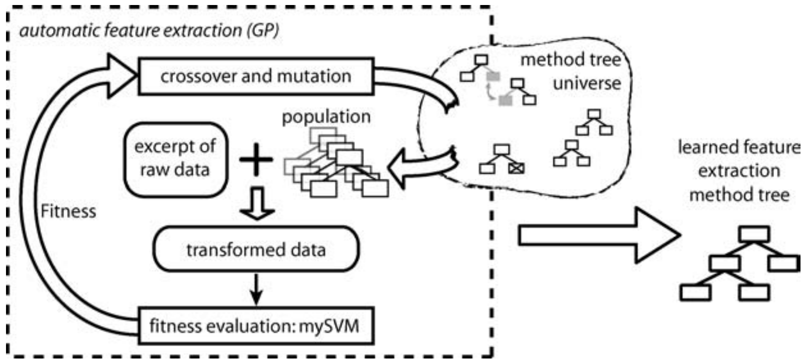
automatic feature extraction (GP)
crossover and mutation
method tree universe
population
excerpt of raw data
Fitness
transformed data
fitness evaluation: mySVM
learned feature extraction method tree

Figure 9. Automatic feature extraction using genetic programming.

```
<operator name="Root" class="ValueSeriesPreprocessing">
  <operator name="Chain 1" class="OperatorChain">
    <operator name="ExpSm" class="ExponentialSmoothing" />
    <operator name="Filter" class="FilterTransformation" />
    <operator name="Windowing" class="Windowing">
      <parameter key="overlap"   value="2"/>
      <operator name="Chain 2" class="OperatorChain">
        <operator name="FFT" class="FastFourierTransform" />
        <operator name="MaxIndex" class="MaxIndexPoint" />
      </operator>
    </operator>
    <operator name="Avg" class="AverageFunction" />
  </operator>
</operator>
```

Figure 10. XML method tree representation for YALE .

3.1. Representation

Genetic programming constructs finite automata. Here, method trees are to be constructed. They are represented by XML expressions. Figure 10 shows the representation of the method tree from figure 8. The YALE system executes such trees and takes care of the syntactic well-formedness.

The restriction that chains are concluded by a function implies a level-wise structure of all possible method trees. The lowest level 1 entails only functions. These are chains of length 1. The next level, 2, covers chains with a concluding function. Levels 3 and above entail windowing. Method trees are constructed according to their levels. The level-wise growing means small changes to a current method tree. On the one hand, this reduces the probability of missing the optimal method tree. On the other hand, it may slow down the search, if the fitness of the lower levels does not distinguish between good and bad method trees.

3.2. Mutation, crossover, and selection

The operations of genetic programming are mutation and crossover. By random, mutations insert a new method, delete a method, or replace a method by one of the same class, i.e. by a function or transformation. Crossover replaces a sub-tree from one method tree by a sub-tree from another method tree, respecting the well-formedness conditions. This means that the roots of the sub-trees must be of the same type of methods.

For selection purposes, the fitness of all method trees is expressed by a *roulette wheel*, i.e. fitness proportional parts of a wheel's 360 degrees. The larger the portion, the more likely it becomes that the particular individual is selected for the next generation or crossover. Other possible selection schemes include *tournament selection*, where t method trees are randomly chosen from the population and the winner of this set, i.e. the method tree with the largest fitness, is added to the next generation. The parameter t allows the definition of selection pressure.

3.2.1. Fitness evaluation. Since method trees serve classification in the end, the quality of classification is the ultimate criterion of fitness. Individuals which provide better classification results when used as features for the classification task at hand should have a greater probability to survive into the next generation. To evaluate the fitness of each method tree in a population the following steps are performed:

1. Each individual method tree is applied to an excerpt of the raw data.
2. This method application returns a transformed data set, which is used by classifier learning.
3. A k-fold cross validation is executed to estimate the performances of the learning scheme in combination with the feature sets provided by the method trees.
4. The mean accuracy, recall, and/or precision of the result becomes the fitness value of the applied feature construction method trees.
5. The fitness values of the method trees are used to build the next population with one of the selection schemes described above.

3.3. Some properties of the search space

Automatically constructing methods for feature extraction which deliver well suited feature sets for classifier learning is a demanding task. How fast can we expect a good result to be found? This general question can be split into three more specific ones. First, the size of the search space is important. Second, the complexity of processing one individual in the search space helps to bound the overall complexity of search. Third, the convergence to an optimum determines the speed of the genetic programming. The last issue is not yet solved. Even simple evolutionary algorithms demand complicated proofs (Droste, Jansen, & Wegener, 1998). Here, we answer the first and second question.

3.3.1. Size of the search space. If all mathematical operations were allowed within method trees, the search space would become infinite, hence only a fixed set of transformations and functions are allowed. Let T_0 be the number of transformations and F the number of functions, n the length of the input series. The number of possible methods at level 1 becomes F. At level 2, transformations could be applied once in any order. The two levels can be summarized. For chains of length k, there exist the following number of different chains:

$$K_0 = F \cdot \sum_{k=0}^{T_0} \frac{T_0!}{(T_0 - k)!} \tag{2}$$

The higher levels are produced by windowing. Within the windows, a chain of transformations is executed, if no nested windowing is allowed. Hence, there exist as many windowing operations as there are chains (Eq. (2)). Adding the number of windowed transformation chains K_0 to the other transformations T_0 returns the number of method trees at level 3:

$$T_1 = T_0 + K_0 = T_0 + F \cdot \sum_{k=0}^{T_0} \frac{T_0!}{(T_0 - k)!} \tag{3}$$

For nested structures, the recursive structure can be illustrated by Eq. (4):

$$T_2 = T_0 + K_1 = T_0 + F \cdot \sum_{k=0}^{T_1} \frac{T_1!}{(T_1 - k)!} \tag{4}$$

The depth of nested windowing is restricted by the length of the series: after $n - 1$ levels of embedded windowing, there is no data of a series with n points left for further windowings. Hence the overall size of the search space of all method trees is upper bounded by:

$$|SearchSpace| \leq F \cdot \sum_{k=0}^{T_{n-1}} \frac{T_{n-1}!}{(T_{n-1} - k)!} \tag{5}$$

Each element in the search space delivers as many features as are determined by the concluding function. If genetic programming has to construct more features, it can either be applied several times, or transformations can be applied more than once. In the latter case, Eq. (5) states the lower bound of the search space size.

3.3.2. Processing a method tree. Until now we have ignored that the window size of embedded windowings must become smaller for increased depth of embedding. Regarding the embedded windowing operators leads to the notion of dynamic windowing.

Definition 16. Let $(x_i)_{i\in\{1,\ldots,n\}}$ be the original value series of length n and $d \in \{2, \ldots n/2\}$. Windowing with overlap g, width $w = n/d$ and step size $s = n/gd$ is called DYNAMIC WINDOWING.

The maximal depth of a method tree can now be determined.

Lemma 6. *Given a value series $(x_i)_{i\in\{1,\ldots,n\}}$ of length n, a method tree using dynamic windowing cannot exceed the depth of* $\log_d n - 1$.

Proof: Dynamic windowing splits the series into windows of width n/d. The width depends on the length of the series as well as on parameter d. For embedded windowing, only n/d values are available. Windows on this smaller series have a window width of n/d^2. Only $\log_d n - 1$ repetitions are possible. The last embedding of windowing with width

$$\frac{n}{d^{\log_d n - 1}} = \frac{n}{\frac{d^{\log_d n}}{d}} = \frac{nd}{n} = d$$

does not allow any further windowing, because then only one value would remain for a window. □

Remember that the number of windows with overlap g on a series of length n is $n/s - g + 1$ (Eq. (1)). Combining the maximal depth of a method tree with the number of windows and their computation efforts estimates the worst runtime complexity of a method tree.

Theorem 1. *Let $CM(n)$ be the complexity of applying an internal method of at most quadratic time complexity to a series of length n. Using dynamic windowing, no method tree requires a runtime which is exponential in the length of the series $(x_i)_{i\in\{1,\dots,n\}}$.*

Proof: The number of windows times the effort per window determines the overall effort. Using Eq. (1) this is for a first level:

$$\left(\frac{n}{s} - g + 1\right) \cdot CM\left(\frac{n}{d}\right) = (g(d-1)+1) \cdot CM\left(\frac{n}{d}\right)$$

Embedding a further windowing delivers at level i the following effort estimation:

$$(g(d-1)+1)^i \cdot CM\left(\frac{n}{d^i}\right)$$

Using Lemma 6 for the bound of i results in the total effort:

$$\begin{aligned}(g(d-1)+1)^{\log_d n-1} \cdot CM\left(\frac{n}{d^{\log_d n-1}}\right) &= (g(d-1)+1)^{\log_d n-1} \cdot CM\left(\frac{n\cdot d}{n}\right)\\ &= (g(d-1)+1)^{\log_d n-1} \cdot CM(d) \qquad (6)\end{aligned}$$

For the worst case of $CM(d) = d^2$, Eq. (6) becomes:

$$\begin{aligned}(g(d-1)+1)^{\log_d n-1} \cdot d^2 &= \frac{d^2}{g(d-1)+1} \cdot n^{\frac{1}{\log_{g(d-1)+1} d}}\\ &= \frac{d^2}{g(d-1)+1} \cdot n^{\log_d(g(d-1)+1)}\end{aligned}$$

As is easily seen, the runtime is not exponential in the length of the series, but is limited by the overlap and the parameter d and is therefore pseudo-polynomial. □

Dynamic windowing avoids a particular case with exponential effort, which otherwise could easily be constructed. If, for instance, the series was divided into two windows at each level with a fixed step size but dynamic window width, the effort would be $2^i \cdot O(\frac{n}{2^i})$ at the i-th level. After $n-1$ splits, no further embedding of windowing is possible, since only two values are left. Hence, the overall effort would be 2^n. This exponential construction, however, does not obey dynamic windowing with fixed overlap and dynamic width. Hence, it cannot happen in our scenario.

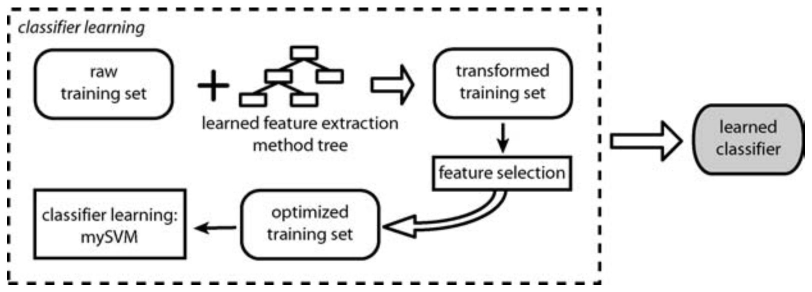

Figure 11. Classifier learning step using the best method tree found by the genetic programming approach.

4. Classification using learned method trees

Automatic feature construction aims at good results of a second learning step which uses the features, namely classifier learning. Remember figure 1 from the introduction, where genetic programming were presented to deliver the input to classifier learning. Now, figure 11 details the second box of the overall picture.

Feature construction is already guided by the classification task in that cross-validated learning determines the fitness of method trees (individuals of genetic programming). Now, also feature selection is performed by a simple evolutionary method, namely the (1+1)EA (Bäck, Hammel, & Schwefel, 1997). Again, the classification task decides upon the fitness. The feature set is built using a subset of the training data. The selected method trees are then applied to all the training data. The support vector machine mySVM is applied to these rewritten data and learns a classifier.

4.1. Classifying genres

Since results are published for the genre classification task, we have applied our approach to this task, too. Note, however, that no published benchmark data sets exist. Hence, the comparison can only show that feature construction and selection leads to similar performance as achieved by other approaches. For the classification of genres, three data sets have been built.

– Classic/pop: 100 pieces for each class were available in Ogg Vorbis format.
– Techno/pop: 80 songs for each class from a large variety of artists were available in Ogg Vorbis format.
– Hiphop/pop: 120 songs for each class from few records were available in MP3 format with a coding of 128 kbits/s.

The classification tasks are of increasing difficulty. Using mySVM with a linear kernel, the performance was determined by a 10-fold cross validation and is shown in Table 1.

Table 1. Classification of genres with a linear SVM using the task specific feature sets.

	Classic/pop	Techno/pop	Hiphop/pop
Accuracy (%)	100	93.12	82.50
Precision (%)	100	94.80	85.27
Recall (%)	100	93.22	79.41
Error (%)	0	6.88	17.50

Concerning classic vs. pop, 93% accuracy, and concerning hiphop vs. pop, 66% accuracy have been published (Tzanetakis, 2002; Tzanetakis, Essl, & Cook, 2001).

41 features have been constructed for all genre classification tasks. For the distinction between classic and pop, 21 features have been selected for mySVM by the evolutionary approach. Most runs selected features referring to the phase space (angle and variance). The use of features can also be inspected by restricting a top-down induction of decision trees to a few levels. For a one level stump, 93% accuracy could be achieved by just using the RMS volume, i.e. the root mean square average of the series.

For the separation of techno and pop, 18 features were selected for mySVM, the most frequently selected ones being the filtering of those positions in the index dimension where the curve crosses the zero line. The decision tree starts with a phase space feature, the average of angles. A one level stump uses the starting value of the second frequency band, giving a benchmark of 76% accuracy.

For the classification into hiphop and pop, 22 features were selected with the mere volume being the most frequently selected feature. The decision tree classifying hiphop against pop is rather complex. It starts with the length of the songs. Experiments with naive Bayes and *k*-NN did not change the picture: an accuracy of about 75% can easily be achieved, increasing the performance further demands better features.

To demonstrate the effect of tailored feature sets for each classification task we performed experiments with the same feature set for all data sets. We used only features which were used in at least 50% of all subsets produced by feature selection for all data sets to simulate a reasonable standard feature set. Table 2 shows the classification performance for a linear SVM estimated with a 10-fold cross validation. The performance is significantly lower than the performance which can be achieved using the tailored feature sets (see Table 1).

Table 2. Classification performance using the same non-tailored standard feature set for all classification tasks (linear SVM).

	Classic/pop	Techno/pop	Hiphop/pop
Accuracy (%)	96.50	64.38	72.08
Precision (%)	94.12	60.38	70.41
Recall (%)	95.31	64.00	67.65
Error (%)	3.50	35.63	27.92

Table 3. Classification errors with respect to different learning schemes.

	Classic/pop	Techno/pop
SVM (linear) (%)	0.00	6.88
SVM (rbf) (%)	1.50	14.38
C4.5 (%)	0.00	7.50
k-NN (%)	3.00	9.38
Naive Bayes (%)	2.50	10.63

Table 3 shows the achieved classification errors with respect to different learning schemes. Since the extraction of features and the transformation in another feature space is performed by the applied method tree, the usage of a linear kernel function is actually no restriction. Therefore, we use a linear SVM for all our experiments and as inner learner to estimate the fitness of the method trees. The conclusions which can be drawn from Tables 2 and 3 indicate that a tailored set of task specific features and not the quality of the learning scheme is the crucial aspect for the successful classification of audio data.

4.2. *User preferences*

Recommendations of songs to possible customers are currently based on the individual correlation of record sales. This collaborative filtering approach ignores the content of the music. A high correlation is only achieved within genres, because the preferences traversing a type of music are less frequent. The combination of favourite songs into a set is a very individual and rare classification. It is not a generalization of many instances. Therefore, the classification of user preferences beyond genres is a challenging task, where for each user the feature set has to be learned. Of course, sometimes a user is interested only in pieces of a particular genre. This does not decrease the difficulty of the classification task. In contrast, if positive and negative examples stem from the same genre, it is hard to construct distinguishing features. Genre characteristics might dominate the user-specific features. As has been seen in the difficulty of the data set for hiphop vs. pop, sampling from few records also increases the difficulty of learning. Hence, four learning tasks of increasing difficulty have been investigated.

Four users brought 50 to 80 pieces of their favourite music ranging through diverse genres. They also selected the same number of negative examples. User 1 selected positive examples from rock music with a dominating electric guitar. User 2 selected positive as well as negative examples from jazz music. User 3 selected music from classic over latin and soul to rock and jazz. User 4 selected pieces from different genres but only from few records. Using a 10-fold cross validation, mySVM was applied to the constructed and selected features, one feature set per learning task (user). Table 4 shows the results.

The excellent learning result for a set of positive instances which are all from a certain style of music corresponds to our expectation (user 1). The expectation that learning performance would decrease if positive and negative examples are taken from the same genre is not

Table 4. Classification according to user preferences.

	User_1	User_2	User_3	User_4
Accuracy (%)	95.19	92.14	90.56	84.55
Precision (%)	92.70	98.33	90.83	85.87
Recall (%)	99.00	84.67	93.00	83.74
Error (%)	4.81	7.86	9.44	15.45

supported (user 2). Surprisingly well is the learning result for a broad variety of genres among the favourites (user 3). This fact indicates that for this user the constructed feature set supports the building of preference clusters in feature space instead of dominating genre clusters. In contrast to this result the (negative) effect of sampling from few records can be seen clearly (user 4). Applying the learned decision function to a database of records allowed the users to assess the recommendations. They were found very reasonable. No particularly disliked music was recommended, but unknown plays and those, which could have been selected as the top 50.

Of course, this method of user preference recognition is just the first step. There are several ways to improve the results. For instance, users could indicate those examples that must be classified correctly, because they feel that it is an essential expression of their taste. Weighting examples as a further cost function for learning has been investigated in (Klinkenberg, 2004). The inspection of the chosen features by the users themselves is not yet possible. Regular users are not acquainted with Fourier transformation and peaks, for instance, but would like to see understandable and interpretable features. Perhaps some visualization of features and mySVM results could satisfy the users' curiosity. An open question is how to circumvent or at least decrease the required number of negative examples, since users don't like to go through long play lists in order to gather negative examples.

5. Conclusion

In this paper, operators for the analysis of large collections of audio data have been presented in a unifying framework. Some new operators have been developed, for instance those in the phase space. Other operators have been generalized, for instance, the windowing and mark-up operators. The operators are organized by method trees, which extract complex features. All known feature extraction methods for audio data are covered, either directly as an operator, or as the result of a method tree. Many different method trees (features) can be built from the primitives of the framework. The method trees are automatically generated for a certain classification task by a genetic programming approach.

Of course, complexity has been an issue. The construction of method trees is restricted to functions at the first level, chains concluded by a function at the second level, and to windows embedding chains at higher levels. The complexity of windowings including methods of a certain complexity has been investigated. It was shown under which circumstances windowing decreases runtime, compared to processing the overall value series. Dynamic windowing with reasonable parameters prevents the approach from becoming infinite or exponential in the length of a series. The countable search space for method trees has been shown to be very large but at least not infinite if each transformation is only allowed once. On the one hand, the size of the search space makes automatic feature extraction a hard task. This seems to be a disadvantage. On the other hand, the large number of elements in the search space means that very many feature extraction methods are covered by the implemented system. This seems to be an advantage. A tight bound tells how many different method trees can be constructed.

The crucial question is whether the covered methods and their automatic construction are tractable, feasible, and deliver good classification results in the end. Two types of classifier learning tasks have been tried in order to answer this question: classifying genres and user preferences. The experiments showed encouraging results. Further work should decrease the required number of negative examples and ease user interaction by visualization of both, features and classification results. Since all methods are defined for value series in general, it would be interesting to see whether the approach can also be applied to image data in the form of vectors or other high-dimensional series data. Another topic for further research is the real-time classification according to user preferences. Genre classification can be done in real-time, because an appropriate feature set can be learned in advance. In contrast, a real-time classification of the preferences of diverse users does not allow to find the feature sets beforehand. Recommendation systems address different users, so that they cannot adjust to a particular one. Real-time recommendation would require automatic feature construction while the user classifies some examples. Here, a combination of collaborative filtering and feature extraction for types of users could be combined. Finally, an *electronic DJ* would be a challenging system to develop. In addition to the classification of single music plays, appropriate sequences of plays would then need to be taken into account.

Notes

1. For an overview see (Pickens, 1996).
2. For details and the comprehensive set of methods see the YALE system which is available at `http://yale.cs.uni-dortmund.de` (Fischer et al., 2002).
3. When constructing features from audio data, exponential methods are not used.

References

Bäck, T., Hammel, U., & Schwefel, H.-P. (1997). Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation, 1:1*, 3–17.

Cooley, J. W., & Tukey, J. W. (1965). An algorithm for the machine computation of the complex Fourier series. *Mathematics of Computation, 19*, 297–301.

Droste, S., Jansen, T., & Wegener, I. (1998). On the analysis of the (1 + 1) evolutionary algorithm. Technical Report CI 21/98, SFB 531, Univ. Dortmund, Germany.

Fischer, S., Klinkenberg, R., Mierswa, I., & Ritthoff, O. (2002). Yale-yet another learning environment tutorial. Technical Report CI 136/02, SFB 531, Univ. Dortmund, Germany.

Ghias, A., Logan, J., Chamberlin, D., & Smith, B. C. (1995). Query by humming: Musical information retrieval in an audio database. In *Proc. of ACM Multimedia* (pp. 231–236).

Guo, G., & Li, S. Z. (2003). Content-based audio classification and retrieval by support vector machines. *IEEE Transaction on Neural Networks, 14:1*, 209–215.

Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics, Springer.

Holland, J. H. (1986). Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine Learning–An artificial intelligence approach*, Chapt. 20, Vol. 2 (pp. 593–624). Palo Alto, CA: Morgan Kaufmann.

Jayant, N. S., & Noll, P. (1984). *Digital coding of waveforms: Principles and applications to speech and video*. Prentice Hall.

Joachims, T. (2002a). *Learning to classify text using support vector machines*, Vol. 668 of Kluwer International Series in Engineering and Computer Science. Kluwer.

Joachims, T. (2002b). Optimizing search engines using clickthrough data. In *Procs. of the 8th Conference on Knowledge Discovery in Databases*.

Kahveci, T., & Singh, A. K. (2001). An efficient index structure for string databases. In *Proceedings of the 27th VLDB* (pp. 352–360). Morgan Kaufmann.

Keogh, E., & Pazzani, M. (1998). An enhanced representation of time series which allows fast classification, clustering and relevance feedback. In *Procs. of the 4th Conference on Knowledge Discovery in Databases*. (pp. 239–241).

Keogh, E., & Smyth, P. (1997). An enhanced representation of time series which allows fast classification, clustering and relevance feedbacA probabilistic approach to fast pattern matching in time series databases. In *Procs. of the 3rd Conference on Knowledge Discovery in Databases* (pp. 24–30).

Klinkenberg, R. (2004). Learning drifting concepts: Example selection vs. example weighting. *Intelligent Data Analysis (IDA), Special Issue on Incremental Learning Systems Capable of Dealing with Concept Drift, 8:3*. (to appear).

Kohavi, R., & John, G. H. (1997). Wrappers for feature subset selection. *Artificial Intelligence, 97:1/2*, 273–324.

Koza, J. (1992). *Genetic programming: On the programming of computers by means of natural selection*. Cambridge, MA: MIT Press.

Kurth, F., & Clausen, M. (2001). Full-text indexing of very-large audio data bases. In *110th Convention of the Audio Engineering Society*.

Liu, Z., Wang, Y., & Chen, T. (1998). Audio feature extraction and analysis for scene segmentation and classification. *Journal of VLSI Signal Processing System*.

Loy, G. (1989). Musicians make a standard: The MIDI phenomenon. *Computer Music Journal, 9:4*.

Morik, K., & Wessel, S. (1999). Incremental signal to symbol processing. In K. Morik, M. Kaiser, & V. Klingspor (Eds.), *Making robots smarter–combining sensing and action through robot learning* Chapt. 11. (pp. 185–198). Kluwer Academic Publ.

Pickens, J. (1996). A Survey of feature selection techniques for music information retrieval. Technical report, Center of Intelligent Information Retrieval, Department of Computer Science, University of Masschusetts.

Rüping, S. (2000). mySVM-Manual. Universität Dortmund, Lehrstuhl Informatik VIII. http://www-ai.cs.uni-dortmund.de/SOFTWARE/MYSVM/.

Takens, F. (1980). Detecting strange attractors in turbulence. In D. A. Rand & L. S. Young (Eds.), *Dynamical systems and turbulence*, Vol. 898 of *Lecture Notes in Mathematics* (pp. 366–381). Berlin: Springer.

Tzanetakis, G. (2002). Manipulation, analysis and retrieval systems for audio signals. Ph.D. thesis, Computer Science Department, Princeton University.

Tzanetakis, G., Essl, G., & Cook, P. (2001). Automatic musical genre classification of audio signals. In *Procs. of the Int. Symposium on Music Information Retrieval (ISMIR)* (pp. 205–210).

Yi, B., Jagadish, H., & Faloutsos, C. (1998). Efficient retrieval of similar time series under time warping. In *Procs. 14th Conference on Data Engineering* (pp. 201–208).

Zhang, T. & Kuo, C. (1998). Content-based classification and retrieval of audio. In *SPIE's 43rd Annual Meeting–Conference on Advanced Signal Processing Algorithms, Architectures, and Implementations VIII*. San Diego.

Received March 22, 2004
Revised October 7, 2004
Accepted October 7, 2004
Final manuscript October 7, 2004