Method Trees: Building Blocks for Self-Organizable Representations of Value Series

How to Evolve Representations for Classifying Audio Data

Ingo Mierswa
Artificial Intelligence Unit
Department of Computer Science
University of Dortmund
mierswa@ls8.cs.uni-dortmund.de

Katharina Morik
Artificial Intelligence Unit
Department of Computer Science
University of Dortmund
morik@ls8.cs.uni-dortmund.de

ABSTRACT

In this paper we introduce a framework for automatic feature extraction from very large series. The extracted features build a new representation which is better suitable for a given learning task. The development of appropriate feature extraction methods is a tedious effort, particularly because every new classification task requires tailoring the feature set anew. Therefore, the simple building blocks defined in our framework can be combined to complex feature extraction methods. We employ a genetic programming approach guided by the performance of the learning classifier using the new representation. Our approach to evolve representations from series data 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 in the domain of music data classification: classification of genres and classification according to user preferences.

Categories and Subject Descriptors: I.2 [Computing Methodologies]: Artificial Intelligence

General Terms: Algorithms

Keywords: Evolving representations, analysis of audio data, feature extraction, time series transformations

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. Information retrieval has started several efforts to automatic indexing [14] and retrieval (e. g., querying by humming [5]). Machine learning has shown its benefits for text classification and ranked document retrieval with respect to user preferences [8, 9]. It is straightforward to expect a similar benefit for the classification and personalized retrieval of music records. However, before machine learning techniques can be employed the music data must be given in a representation from which a classification can be learned.

Current approaches to time series indexing and similarity measures rely on a more or less fixed time scale [12, 11]. Audio files, however, differ considerably in length. More general, time series similarity is determined with respect to some (flexible and generalized) shape of curves [10, 21]. However, the shape of the audio curve does not express the crucial aspect for classifying genres or preferences. The i-th value of a favorite song has no correspondence to the i-th value of another favorite, even if relaxed to the $(i \pm n)$-th value. The decisive features for classification have to be extracted from the original data to build a new and more suitable representation. Hence, feature extraction from audio data has become a hot topic recently [6, 15, 19, 22]. Several specialized extraction methods have shown their performance on some task and data set. However, different classification tasks ask for different feature sets. It is not very likely that a feature set delivering excellent performance on the separation of classical and popular music works well for techno and hip hop music, too. Classifying music according to user preferences even aggravates the problem.

This paper will focus on self-organizable building blocks to build suitable representations which ease classification learning. We illustrate some operators and how they are combined to become extraction methods in section 2. 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. We construct a representation for each given task and data set, anew. Hence, the evolutionary construction of method trees is equivalent to an evolutionary construction of a learning representation. Section 3 describes the genetic programming approach to automatically combine the basic feature extraction methods. It also discuss how it is assured that only valid combinations are constructed. The search within the universe of methods 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 representation (i. e., the extraction method).

Applying a method tree to the given audio data deliv-

GECCO'05, June 25–29, 2005, Washington, DC, USA.

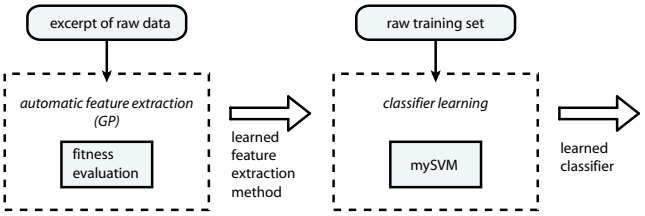


Figure 1: The overall process of automatic feature construction for classification.

ers a transformed data set, i. e., the examples rewritten by the corresponding feature set. This becomes the input to a second learning step, namely 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 [17] 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 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 section 3. The approach is tested on the learning tasks of genre classification and user preferences (section 4).

2. ELEMENTARY BUILDING BLOCKS

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 under consideration here refer to value series. They are not only applicable to audio data, but to value series in general. All that is required is the definition of a scalar product for the space.

We structure the set of elementary operators as follows:

Transformations map the data from the given vector space into another space or to another place in the same space.

Generalized windowing is required by many methods for feature extraction. We separate the windowing from the functionals applicable to values within the windows.

Functionals calculate a single value for a value series.

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 some of the building blocks, so that we later on can flexibly construct sequences with techniques derived from genetic programming. Details are omitted [1]. Only the new notion of general windowing, which leads to method trees, is presented in more detail.

2.1 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* $|f| = \langle f, f \rangle^{\frac{1}{2}}$ *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.

2.1.1 Frequency space

The goal of *Fourier analysis* is to write the given series $(x_i)_{i \in \{1,\ldots,n\}}$ as a (possibly infinite) sum of multiples of the base functions, which are $e^{i\nu x}$ for all frequencies ν. A Fast Fourier Transformation [2] efficiently maps the given time space into a frequency space containing a defined number of frequencies. 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 3. *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

[1]For details and the comprehensive set of methods see the YALE system [4] which is available at `http://yale.cs.uni-dortmund.de`.

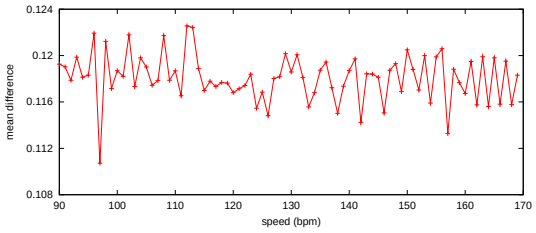


Figure 2: Autocorrelation differences for a phase shift depending on speeds ranging from 90 to 170 beats per minute.

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 2 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*.

DEFINITION 4. *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 some characteristics which can hardly be seen in the original space. Since the state variables are often unknown, a topologically equivalent space is constructed [18]. This is known as *reconstruction of state space*:

DEFINITION 5. *The components of the vectors within the* PHASE SPACE *are parts of the original series:*

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

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

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

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

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 3.

2.1.4 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 6. *Given a value series $(x_i)_{i\in\{1,...,n\}}$, 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.

2.2 Generalized windowing

Many known operators on times series involve windowing. We give a general definition of windowing operators which leads to a tree structure of the used methods. Separating the notion of windows over the index dimension from the functionals applied to the values within the window segment allows to construct many operators of the kind.

DEFINITION 7. *Given the series $(x_i)_{i\in\{1,...,n\}}$, a transformation is called* WINDOWING, *if it shifts a window of width w over $(x_i)_{i\in\{1,...,n\}}$ using a step size s and evaluates in each window the functional F:*

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

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

DEFINITION 8. *A windowing which performs an arbitrary number of transformations in addition to the functional F is called* GENERAL WINDOWING.

The functional F summarizes values within a window and thus prevents general windowing from enlarging the data set. 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 9. *The* OVERLAP *of a general windowing with step size s and width w is defined as $g = w/s$.*

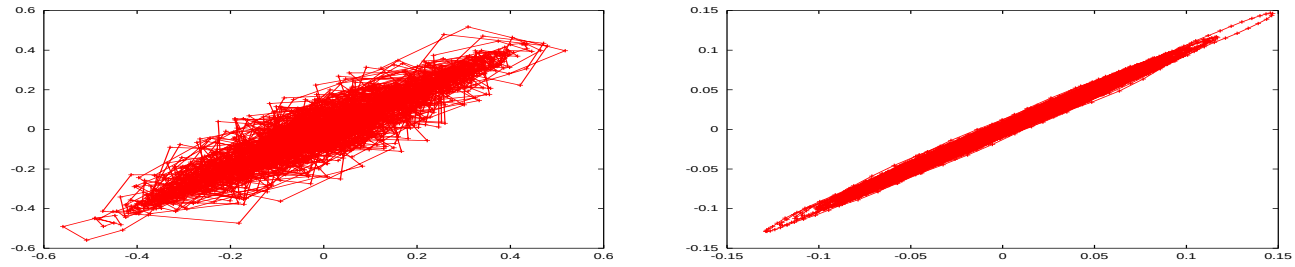
Only for windowings with overlap $g = 1$ the functional can be omitted. Such a windowing performs transformations for each window and is called *piecewise filtering*. The runtime effects of the overlap size for transformations used within general windowing are investigated in [16].

2.3 Functionals

Transformations convert a series into another series. In contrast, functionals calculate single values from a series. The group of functionals 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 functionals are those indicating peaks:

DEFINITION 10 (k-PEAKS FUNCTIONAL). *k-peaks functional delivers for a series $(x_i)_{i\in\{1,...,n\}}$ 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 functionals [7]. The *mel-frequency*



(a) pop music (b) classic

Figure 3: Phase space representation of a popular song (left) and a classical piece (right).

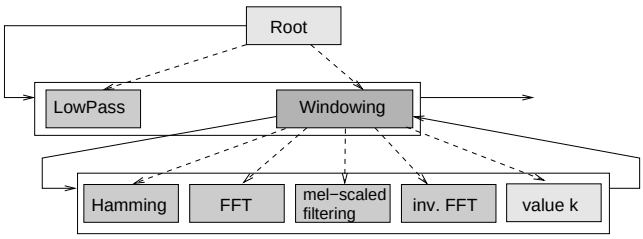


Figure 4: Constructing the cepstral method from elementary extraction operators.

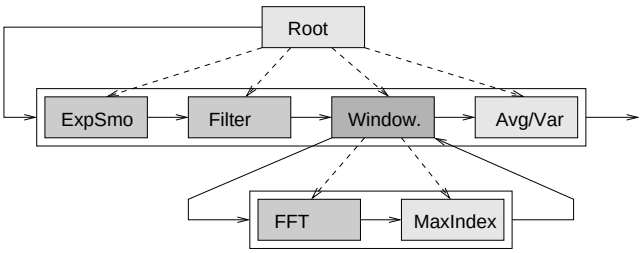


Figure 5: A method tree for feature extraction built of elementary methods. Solid arrows show the data flow, dashed lines define the tree structure.

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 4 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.

3. EVOLUTIONARY CONSTRUCTION OF METHOD TREES

The elementary methods described above are combined in order to construct more complex features for classification tasks. Figure 4 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. *Genetic programming* is applied in order to look for the best combination of methods [13]. The result is a complex method tree build from elementary building blocks. Its use for the classifier learning will be shown in section 4.

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

DEFINITION 11. *A* CHAIN *consists of an arbitrary number of transformations and a functional at the end.*

Incorporating windowings into chains leads to the concept of method trees:

DEFINITION 12. *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. The restriction that chains are concluded by a functional implies a level-wise structure of all possible method trees. The lowest level 1 entails only functionals. These are chains of length 1. The next level, 2, covers chains with a concluding functional. 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.

Another example of a method tree is shown in Figure 5, where the root identifies the element within the search space. Its four children are exponential smoothing, a filtering, another method tree (Fourier transformation with peaks applied to windows), and the average of the peaks. This last child returns the desired features. Such method trees are represented by XML expressions. The YALE system [4] executes such trees and takes care of the syntactic well-formedness.

Before the genetic programming approach is technically described, Figure 6 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

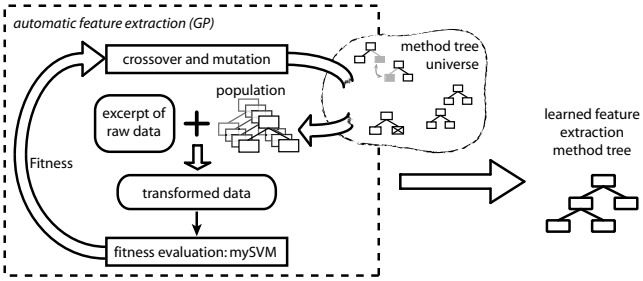


Figure 6: Automatic feature extraction using genetic programming.

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. 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 functional or transformation. It will be assured that all method trees remain valid after mutation, e. g. that the end functional of a windowing is not removed. 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.

The optimization cycle corresponds to the standard process of genetic algorithms. What differs from the standard is that the fitness evaluation is not merely a function but the result of running a learning algorithm. 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.

3.1 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 [3]. Here, we answer the first and second question.

3.1.1 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 functionals are allowed. Let T_0 be the number of transformations and F the number of functionals, 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:

$$K_0 = F \cdot \sum_{k=0}^{T_0} \frac{T_0!}{(T_0 - k)!} \tag{1}$$

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 (equation 1). Adding the number of windowed transformation chains K_0 to the other transformations T_0 returns the number of method trees at level 3:

$$T_1 = T_0 + K_0 = T_0 + F \cdot \sum_{k=0}^{T_0} \frac{T_0!}{(T_0 - k)!} \tag{2}$$

For nested structures, the recursive structure can be illustrated by equation 3:

$$T_2 = T_0 + K_1 = T_0 + F \cdot \sum_{k=0}^{T_1} \frac{T_1!}{(T_1 - k)!} \tag{3}$$

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:

$$|SearchSpace| \leq F \cdot \sum_{k=0}^{T_{n-1}} \frac{T_{n-1}!}{(T_{n-1} - k)!} \tag{4}$$

Each element in the search space delivers as many features as are determined by the concluding functional. If genetic programming has to construct more features, it can either be applied several times, or transformations can be applied more than once.

3.1.2 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 13. *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 1. *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.

The number of windows with overlap g on a series of length n is $n/s - g + 1$. 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 2. *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 $(x_i)_{i\in\{1,\ldots,n\}}$.*

Proof: The number of windows times the effort per window determines the overall effort. Using $n/s - g + 1$ 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 1 for the bound of i results in the total effort:

$$\begin{aligned} & (g(d-1)+1)^{\log_d n - 1} \cdot CM\left(\frac{n}{d^{\log_d n - 1}}\right) \\ = \; & (g(d-1)+1)^{\log_d n - 1} \cdot CM\left(\frac{n \cdot d}{n}\right) \\ = \; & (g(d-1)+1)^{\log_d n - 1} \cdot CM(d) \end{aligned} \quad (5)$$

For the worst case of $CM(d) = d^2$, equation 5 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.

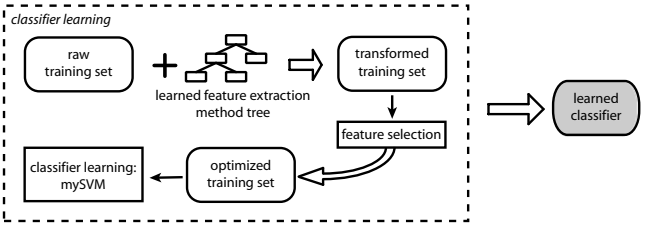


Figure 7: Classifier learning step using the best method tree found by the genetic programming approach.

4. EVALUATING THE REPRESENTATIONS DERIVED FROM 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 7 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 [1]. 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 this new representation 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. Concerning classic vs. pop, 93% accuracy, and concerning

	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%

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

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

hiphop vs. pop, 66% accuracy have been published [19, 20].

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).

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 favorite 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.

	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%

Table 3: Classification according to user preferences.

Four users brought 50 to 80 pieces of their favorite 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 3 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 supported (user 2). Surprisingly well is the learning result for a broad variety of genres among the favorites (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.

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. The windowing has been generalized which leads to method trees as new representation structure for series data. 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 can be built from the building blocks 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 functionals at the first level, chains concluded by a functional at the second level, and to windowings 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 size of the search space for method trees has been estimated and gives a first idea to analyze the complex structures which can be build using the described building blocks. Although this is a first step, a rigorous analysis of the runtime of a genetic programming algorithm building complex representations from simple building blocks is still missing.

6. REFERENCES

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

[2] J. W. Cooley and J. W. Tukey. An algorithm for the machine computation of the complex Fourier series. *Mathematics of Computation*, 19:297–301, April 1965.

[3] S. Droste, T. Jansen, and I. Wegener. On the analysis of the (1+1) evolutionary algorithm. Technical Report Reihe CI 21/98, SFB 531, Universität Dortmund, Germany, 1998.

[4] S. Fischer, R. Klinkenberg, I. Mierswa, and O. Ritthoff. Yale: Yet Another Learning Environment – Tutorial. Technical Report CI-136/02, Collaborative Research Center 531, University of Dortmund, Dortmund, Germany, Juni 2002. ISSN 1433-3325.

[5] A. Ghias, J. Logan, D. Chamberlin, and B. C. Smith. Query by Humming: Musical Information Retrieval in an Audio Database. In *Proc. of ACM Multimedia*, pages 231–236, 1995.

[6] G. Guo and S. Z. Li. Content-Based Audio Classification and Retrieval by Support Vector Machines. *IEEE Transaction on Neural Networks*, 14(1):209–215, January 2003.

[7] N. S. Jayant and P. Noll. *Digital Coding of Waveforms: Principles and Applications to Speech and Video*. Prentice Hall, 1984.

[8] T. Joachims. *Learning to Classify Text using Support Vector Machines*, volume 668 of *Kluwer International Series in Engineering and Computer Science*. Kluwer, 2002.

[9] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of Knowledge Discovery in Databases*, 2002.

[10] T. Kahveci and A. K. Singh. An efficient index structure for string databases. In *Proceedings of the 27th VLDB*, pages 352–360. Morgan Kaufmann, 2001.

[11] E. Keogh and M. Pazzani. 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*, pages 239 – 241, 1998.

[12] E. Keogh and P. Smyth. 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*, pages 24 – 30, 1997.

[13] J. Koza. *Genetic Programming: On the programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, 1992.

[14] F. Kurth and M. Clausen. Full-text indexing of very-large audio data bases. In *110th Convention of the Audio Engineering Society*, 2001.

[15] Z. Liu, Y. Wang, and T. Chen. Audio Feature Extraction and Analysis for Scene Segmentation and Classification. *Journal of VLSI Signal Processing System*, June 1998.

[16] I. Mierswa and K. Morik. Automatic feature extraction for classifying audio data. *Machine Learning Journal*, 58:127–149, 2005.

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

[18] F. Takens. Detecting strange attractors in turbulence. In D. A. Rand and L. S. Young, editors, *Dynamical systems and turbulence*, volume 898 of *Lecture Notes in Mathematics*, pages 366 – 381. Springer, Berlin, 1980.

[19] G. Tzanetakis. *Manipulation, Analysis and Retrieval Systems for Audio Signals*. PhD thesis, Computer Science Department, Princeton University, June 2002.

[20] G. Tzanetakis, G. Essl, and P. Cook. Automatic musical genre classification of audio signals. In *In Proceedings of the Int. Symposium on Music Information Retrieval (ISMIR)*, pages 205–210, 2001.

[21] B. Yi, H. Jagadish, and C. Faloutsos. Efficient retrieval of similar time series under time warping. In *Procs. 14th Conference on Data Engineering*, pages 201 – 208, 1998.

[22] T. Zhang and C. Kuo. 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, July 1998.