Learning Feature Extraction for Learning from Audio Data

Ingo Mierswa, Katharina Morik[1]

University Dortmund, Computer Science Department, LS VIII
{mierswa,morik}@ls8.cs.uni-dortmund.de,
http://www-ai.cs.uni-dortmund.de

Abstract. Today, large collections of digital music plays are available. These audio data are time series which need to be indexed and classified for diverse applications. Indexing and classification differs from time series analysis, in that it generalises several series, whereas time series analysis handles just one series a time. The classification of audio data cannot use similarity measures defined on the raw data, e.g. using time warping, or generalise the shape of the series. The appropriate similarity or generalisation for audio data requires feature extraction before classification can successfully be applied to the transformed data. 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 to tailor the feature set anew. Hence, we consider the construction of feature extraction methods from elementary operators itself a first learning step. We use a genetic programming approach. 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.

1 Introduction

Since music has become distributed via the internet and is stored in digital form, 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 users music records which they might like? Information retrieval has started several efforts to automatic indexing [11] and retrieval (e.g., querying by humming [2]). For classification, machine learning encounters a new challenge of scalability, when confronted with music data:

- Music databases store millions of records.
- Given a sampling rate of 44100 Hz, a three minute music record has the 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 [7, 8]. Music plays, however, differ considerably in length. More general, time series similarity is determined with respect to some (flexible and generalized) shape of curves [18, 6]. 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 in form of Midi data, i.e. a transcription according to the 12 tone system [13] [1]. This allows to include background knowledge from music theory. The data are given, however, in the form of – possibly compressed – waves records, the audio data. Hence, feature extraction from audio data has become a hot topic recently [12, 19, 3, 16]. 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, 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 the separation of techno and hip hop music, too. Classifying music according to user preferences even aggravates the problem. If there were a concise set of feature extraction methods, one could adopt the wrapper approach [10] in order to select the subset of features which is well suited for the given classification task. However, such a concise set does not exist and if it would, it would be extremely large. Therefore, we propose another procedure. Given some elementary operators, genetic programming constructs methods which are (nested) sequences of operators.

In this paper, we illustrate some operators and how they are combined to become extraction methods in Section 2. Section 3 describes the genetic programming approach to learning the feature extraction methods. 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 feature set (i.e., the extraction method). 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. Applying the method 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 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 [15] for classifier learning.

The approach is tested on the learning tasks of genre classification and user preferences (Section 4).

[1] For an overview, see [14].

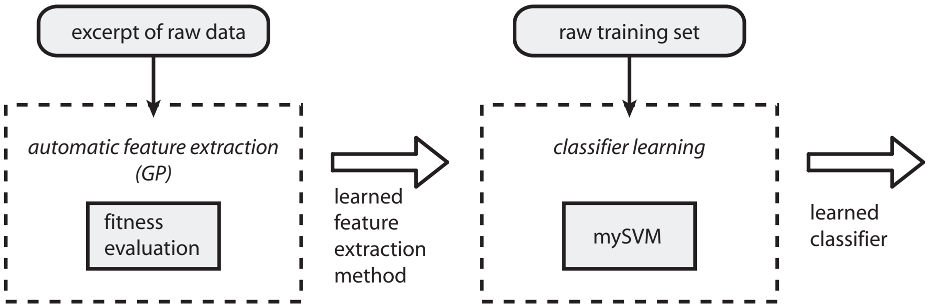


Fig. 1. The overall process of automatic feature construction for classification

2 Operators and 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 (Value series) *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 definiton of a scalar product for the space.

We structure the set of elementary operators as follows:

Basis transformations map the data from the given vector space into another space, e.g. frequency space, function space, phase space. The most popular transformation is the Fouries analysis.

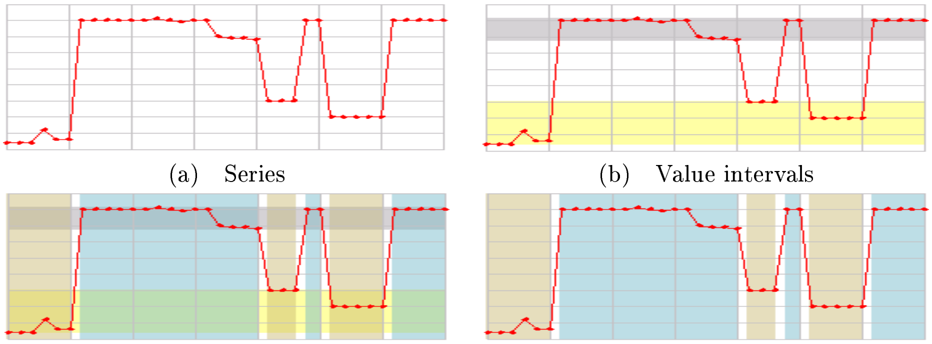
Filters transform elements of a given series to another location within the same space. Moving average or exponential smoothing are examples of filters.

Mark-up of intervals corresponds to the mark-up of text fragments in that it annotates segments within a value series.

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

Functions calculate a single value for a value series. Typical examples are average, variance, and standard deviation.

Let us give some examples of the operators. Since the group of mark-up operators is newly introduced, a definition is given.



(a) Series (b) Value intervals

(c) Index intervals (d) Result

Fig. 2. The process of finding intervals in a series (a), first in the value dimension (b), then projected on the index dimension (c), delivering (d).

Definition 2 (Mark-up) *A mark-up $M : S \to C$ assigns a characteristic C to a segment S.*

Definition 3 (Interval) *An interval $I : S \to C$ 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.*

Operators finding intervals in the value dimension of a value series can be combined with the mark-up of intervals in the time (i.e. indexing) 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 2 illustrates this combination.

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 4 (Windowing) *Given the series $(x_i)_{i \in \{1,\ldots,n\}}$, a transformation is called windowing, if it shifts a window of width w using a step size of s and calculates in each window the function F:*

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

All y_j together form again a series $(y_j)_{j \in \{1,\ldots,(n-w)/s+1\}}$.

Definition 5 (General windowing) *A windowing which performs an arbitrary number of transformations in addition to the function F is called a 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. 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 only performs transformations for each windows and is called *piecewise filtering*. 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. Of course, this speeds up processing considerably.

The elementary operators can be combined so that methods of feature extraction are expressed. For audio data, the spectral flatness measure or the spectral crest factor can be expressed as an arithmetic combination of simple functions [5]. 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, a psychoacoustic filtering is performed, and the inverse Fourier transformation is applied to the result. Figure 3 shows how the operators for feature extraction are put together to form the cepstral coefficients. From these coefficients additional features can be extracted. It is seen how easily new, similar methods can be generated, e.g., replacing the frequency spectrum and its logarithm by the gradient of a regression line.

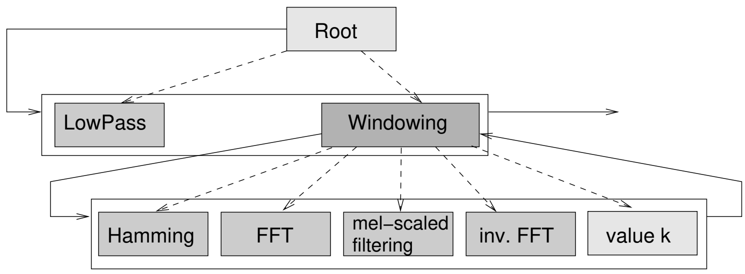


Fig. 3. Constructing the cepstral method from elementary extraction operators

3 Automatic construction of method trees

The elementary operators described above are combined in order to construct methods that extract features for classification tasks. Figure 3 already showed how elementary operators can be used for the reconstruction of known complex feature extraction methods. In addition, an example of a similar method combining different elements was given. There are many complex feature extraction methods which can be built using the operators. For instance, the general windowing may apply a Fourier transformation so that the peaks of the transformed can be related with windows in time:

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

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.

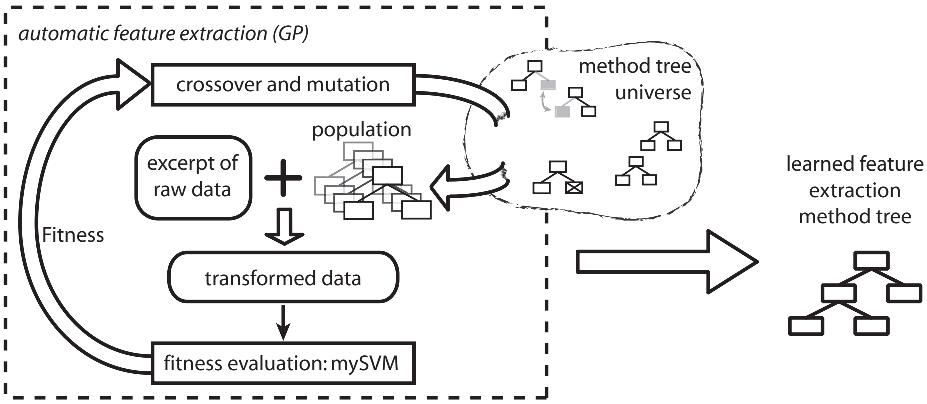


Fig. 4. Automatic feature extraction using genetic programming.

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 search space is too large to be inspected manually. Hence, *genetic programming* is applied in order to look for the best combination of methods [4]. 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 6 (Chain) *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.

Definition 7 (Method tree) *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.*

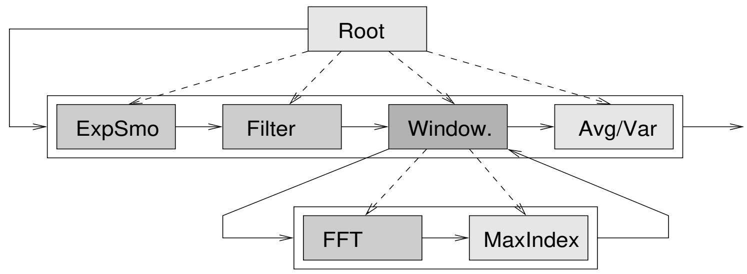


Fig. 5. A method tree for feature extraction built of elementary methods. Dashed arrows show the data flow, solid lines define the tree structure.

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.

An 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 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 4 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 become a new population. This cycle corresponds to the standard process of genetic programming. 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.

Genetic programming constructs finite automata. Here, method trees are to be constructed. They are represented by XML expressions. Figure 6 shows the representation of the method tree from Figure 5. 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

```xml
<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>
```

Fig. 6. XML method tree representation for YALE.

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.

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, regarding the well-formedness conditions. That 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. Since method trees serve classification in the end, the quality of classification is the ultimate criterion of fitness. Each individual method tree is applied to the raw data. This method application returns a transformed data set, which is used by classifier learning. A k-fold cross validation is executed. The mean accuracy, recall, and/or precision of the result becomes the fitness value of the applied feature construction method tree.

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 shown to deliver the input to classifier learning. Now, we describe the second step, namely classifier learning. 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

	Classic/pop	Techno/pop	Hiphop/pop
Accuracy	100%	92,89%	82,38%
Precision	100%	94,00%	84,15%
Recall	100%	92,64%	78,44%
Error	0%	7,11%	17,62%

Table 1. Classification of genres

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: 105 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 hiphop vs. pop, 66% accuracy has been published [16, 17].

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). 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. For the classification into hiphop and pop, 22 features were selected with the mere volume being the most frequently selected feature. 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.

4.2 User preferences

Recommendations of songs to possible customers are currently based on the correlation of records, user who bought record A frequently also bought record B.

	$\mathbf{User}_1$	$\mathbf{User}_2$	$\mathbf{User}_3$	$\mathbf{User}_4$
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 2. Classification according to user preferences

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 2 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 favourites (user 3). 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 the person feels that it is an essential expression of his taste. Weighting examples as a further cost function for learning has been investigated in [9]. 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, a genetic programming approach to learning feature extraction for a certain data set and classification task has been presented. Using elementary operators as building blocks, method trees are learned under the guidance of the performance of a classification learner. 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 investigate 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 build-up 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.

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. Do the methods deliver features which are well suited for sequence learning, as well?

Acknowledgments

The support of the Deutsche Forschungsgemeinschaft (SFB 475, "Reduction of Complexity for Multivariate Data Structures") is gratefully acknowledged.

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. Asif. Ghias, Jonathan Logan, David Chamberlin, and Brian C. Smith. Query by Humming: Musical Information Retrieval in an Audio Database. In *Proc. of ACM Multimedia*, pages 231–236, 1995.
3. 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.
4. J. H. Holland. Escaping brittleness: The possibilities of general–purpose learning algorithms applied to parallel rule–based systems. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning – An Artificial Intelligence Approach*, volume 2, chapter 20, pages 593–624. Morgan Kaufmann, Palo Alto, CA, 1986.

5. N. S. Jayant and P. Noll. *Digital Coding of Waveforms: Principles and Applications to Speech and Video.* Prentice Hall, 1984.
6. Tamer Kahveci and Ambuj K. Singh. An efficient index structure for string databases. In *Proceedings of the 27th VLDB*, pages 352–360. Morgan Kaufmann, 2001.
7. 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.
8. Eamonn Keogh and Michael 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.
9. Ralf Klinkenberg. 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), May 2004. To appear.
10. R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.
11. Frank Kurth and Michael Clausen. Full-text indexing of very-large audio data bases. In *110th Convention of the Audio Engineering Society*, 2001.
12. 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.
13. G. Loy. Musicians make a standard: the MIDI phenomenon. *Computer Music Journal*, 9(4), 1989.
14. Jeremy Pickens. A survey of feature selection techniques for music information retrieval. Technical report, Center of Intelligent Information Retrieval, Department of Computer Science, University of Masschusetts, 1996.
15. Stefan Rüping. *mySVM-Manual*. Universität Dortmund, Lehrstuhl Informatik VIII, 2000. http://www-ai.cs.uni-dortmund.de/SOFTWARE/MYSVM/.
16. George Tzanetakis. *Manipulation, Analysis and Retrieval Systems for Audio Signals.* PhD thesis, Computer Science Department, Princeton University, June 2002.
17. George Tzanetakis, Georg Essl, and Perry Cook. Automatic musical genre classification of audio signals. In *In Proceedings of the Int. Symposium on Music Information Retrieval (ISMIR)*, pages 205–210, 2001.
18. 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.
19. 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.