Automatic Feature Extraction from Large Time Series

Ingo Mierswa

Univ. Dortmund, Computer Science VIII, Germany
mierswa@ls8.cs.uni-dortmund.de

Abstract. The classification of high dimensional data like time series requires the efficient extraction of meaningful features. The systematization of statistical methods allows automatic approaches to combine these methods and construct a method tree which delivers suitable features. It can be shown that the combination of efficient methods also works efficiently, which is especially necessary for the feature extraction from large value series. The transformation from raw series data to feature vectors is illustrated by different classification tasks in the domain of audio data.

1 Introduction

Each instance for a numerical learning algorithm is described by the values of a given set of features. The learning scheme should find a hypothesis which allows the classification of unseen data (Mitchell (1996) and Witten and Frank (2000)). Transforming the given representation may ease learning such that a simple learning algorithm can solve the problem and provide better results (Morik (2000) and Pyle (1999)).

Music is a real-valued function of time. Therefore, audio data can be seen as an univariate value series. The amplitude a_i for each time point i (*sample point*) is given. A three-minute mono song consists of $44100Hz \cdot 180s \approx 8 \cdot 10^6$ values. The classification of these high dimensional series requires the extraction of features, so that a classification scheme can make use of the feature vectors instead of the large series data. By extracting the small feature vectors, both the improvement of results (Liu and Motoda (1998) and Ritthoff et al. (2002)) and a strong data compression is expected.

We have to face up with two problems: first, the great amount of data requires efficiently working methods to extract the features and second, it is not always clear which is the meaning of the extracted features. The automatic selection and combination of the best methods for feature extraction would be very useful.

The next section introduces a systematization of statistical methods, which allows automatic feature extraction from value series data. In section 3, an automatic approach for feature extraction based on genetic programming is presented and the runtime is analyzed. In section 4 the feature extraction from audio data is described and the results are discussed.

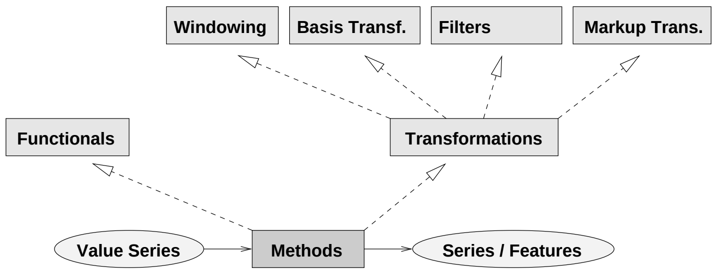

Fig. 1. All methods for value series analysis can be divided into groups according to their output. Transformations can be further divided in basis transformations, filters, mark-up transformations, and windowing.

2 Systematization of statistical methods

A systematization of statistical methods must be powerful enough to cover all known and future methods and it must be precise enough to allow automatic approaches to select and combine methods to find the optimal set of extracted features. A *method* is defined in an operator based way: it gets a value series as input and applies an arbitrary operation in order to deliver a result. This result turns out to be a good criterion to divide the methods into groups. We distinguish between:

Transformations: All methods which deliver a value series as output, i. e. a mapping $t : F \rightarrow F$ for a function space F of value series $(x_i)_{i \in \{1,\ldots,n\}}$.
Functions: All methods which deliver single values without any order, i. e. a mapping $f : F \rightarrow \mathbb{R}^m$ between a function space F and real numbers.

Transformations, which change the series itself without generating features, can be divided into several groups like *basis transformations* (e. g. Fourier transformation or state space reconstruction), *filters* (e. g. window functions or difference filter), and *mark-up transformations* (e. g. finding intervals in the series). Chains built from an arbitrary number of transformations and ending with a function deliver the desired features. Figure 1 shows the systematization.

2.1 Windowing extends the method space

In order to divide the existing methods for value series analysis (Bradley (1999) and Schlittgen and Streitberg (1997)) into the specified groups, a particular transformation requires a special treatment. With the aid of a *Windowing* operator a bunch of further transformations can be simulated and created:

Windowing: Given a value series $(x_i)_{i \in \{1,\ldots,n\}}$ with length n. A transformation is called *Windowing* if a window of size w is moved with step size s over the series and in each window the value of a function f is calculated:

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

The values y_j form a new series $(y_j)_{j \in \{0, \dots, \lfloor (n-w)/s \rfloor\}}$.

If f is an average function, this definition of a windowing includes the well known moving average filters. But we take a step forward and allow all functions for windowing and additionally allow any number of transformations before we calculate the value of the function[1]. For large value series we must ensure that a windowing which uses efficient methods to calculate the values y_j also is an efficient method, i.e. has polynomial runtime. The overlap of a windowing is defined as $g = \frac{w}{s}$. Each windowing creates $\frac{n-w}{s} + 1 = \frac{n}{s} - g + 1$ windows. A windowing performing transformations and a function with runtime $O(n^2)$ on each window has an overall runtime of

$$\left(\frac{n}{s} - g + 1\right) \cdot w^2 = gnw - gw^2 + w^2.$$

To estimate the worst case we consider a windowing with step size $s = 1$. For a realistic overlap of $g = 2$ the runtime is $2nw - w^2$ which is smaller than n^2 for all window sizes $w < n$. The maximum amount of multiple used values is reached for a window size of $w = \frac{n}{2}$ and therefore an overlap of also $g = \frac{n}{2}$. For this worst case the runtime is

$$\begin{aligned} gnw - gw^2 + w^2 &= n \cdot \left(\frac{n}{2}\right)^2 - \left(\frac{n}{2}\right)^3 + \left(\frac{n}{2}\right)^2 \\ &= \frac{n^3}{8} + \frac{n^2}{4}. \end{aligned}$$

The runtime has a greater power in n but is still efficient. Similar calculations for the runtimes of other methods show that the usage of windowing with a realistic overlap like $g = 2$ always result in a smaller runtime than the application of the methods on the complete series.

2.2 Method trees for feature extraction

As we have mentioned, the extracted features are the result of a chain of transformations and a function at its end. A windowing is also a transformation. But this particular transformation performs other methods on windows to form a new series. One can see these methods as children of a windowing, which leads to the model of *method trees* for feature extraction.

Figure 2 shows an example for a method tree. The tree is traversed with a depth first search. The windowing is the root of a new tree, whose children

[1] Actually we can do a windowing without a function but with transformations only. This should only be done for windowings with $w = s$ and is called *piecewise filtering*.

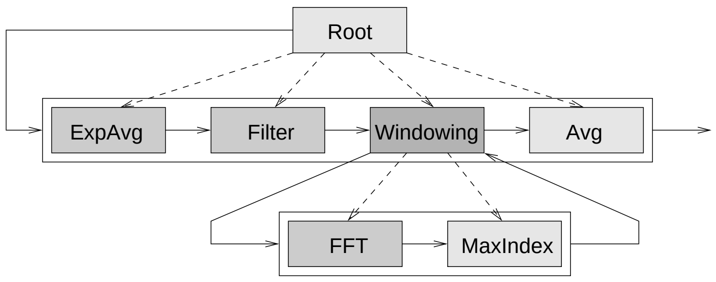

Fig. 2. A method tree that extracts the feature "average and variance of the maximum frequency in the progression of time". The windowing method is the root of a new tree.

are invoked once for each window. The dashed lines show the parent-child connection in the tree and the solid lines stand for the data flow. The last child in the chain is an average function which delivers the features "average and variance of the maximum frequency in the progression of time".

2.3 Dynamic windowing in method trees

Each method tree provides one or several features. The tree structure emerges from the nesting of windowing methods. It is quite clear that it is impossible to nest two windowings with the same window size w. Since the children of the parent windowing work on windows of size w, a nested windowing with window size w creates a series with length 1 which is actually not a series anymore. Therefore the overlap of each windowing method should be fixed, which makes sure that the windowing works efficiently for realistic overlaps. The size of the windows must be *dynamic* and is defined as $w = \frac{n}{d}$ for a parameter $d \in \{2, \ldots, \frac{n}{2}\}$.

Dynamic windowing: Given a series $(x_i)_{i \in \{1,\ldots,n\}}$ with length n and a parameter $d \in \{2, \ldots, \frac{n}{2}\}$. A windowing with overlap g, window size $w = \frac{n}{d}$ and step size $s = \frac{n}{gd}$ is called *dynamic windowing.*

A method tree built of dynamic windowings and other transformations and functions has a maximum depth of $\log_d n - 1$. A dynamic windowing divides the series in windows with size $\frac{n}{d}$. Therefore, each dynamic windowing which is a child of another windowing works only on $\frac{n}{d}$ values and builds windows with size $\frac{n}{d^2}$. After $\log_d n - 1$ nested windowings each window has

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

values. Another windowing would reduce the number of values for the next child to 1.

Now we are able to analyze the runtime of method trees on a value series with length n. The worst case runtime of all transformations and functions is given as $L(k)$ on k values. The runtime of a dynamic windowing is

$$\begin{aligned}&\left(\frac{n}{s} - g + 1\right) \cdot L\left(\frac{n}{d}\right)\\ &= (g(d-1)+1) \cdot L\left(\frac{n}{d}\right).\end{aligned}$$

We add another dynamic windowing as a child and replace $L(\frac{n}{d})$ by $(g(d-1)+1) \cdot L(\frac{n}{d^2})$ which leads to $(g(d-1)+1)^2 \cdot L(\frac{n}{d^2})$. We iterate these steps which delivers

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

as runtime of a method tree with depth i. We have shown that each method tree has a maximum depth of $\log_d n - 1$ which results in a runtime of

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

A method tree based on methods with a worst case runtime of $O(n^2)$ therefore has an overall runtime of

$$(g(d-1)+1)^{\log_d n - 1} \cdot d^2 = \frac{d^2}{g(d-1)+1} \cdot n^{\frac{1}{\log_{g(d-1)+1} d}}.$$

It has been shown that the runtime is never exponential, for realistic dynamic windowings with overlap $g = 2$ and $d = 2$ the runtime is $\frac{4}{3} \cdot n^{1.585}$ which is always smaller than n^2. Hence, method trees built from efficient methods are efficient too.

3 Automatic feature extraction

We have fulfilled two premises for automatic approaches for feature extraction: the search space is structured and the elements of this space work efficiently and extract features from high dimensional data in polynomial time. Now we introduce a simple way for automatic feature extraction based on *genetic programming*. The search space in which the algorithm tries to find the optimum is the space of all method trees which can be created with the given transformations and functions. Each individual is a method tree and the tree which provides the best features for the classification task at hand is delivered as the result.

Figure 3 shows the functioning of a genetic programming algorithm. The first step is to create a population consisting of a number of individuals. Here, randomly created method trees are used as individuals. Then the same steps are performed repeatedly until a termination criterion is satisfied. The

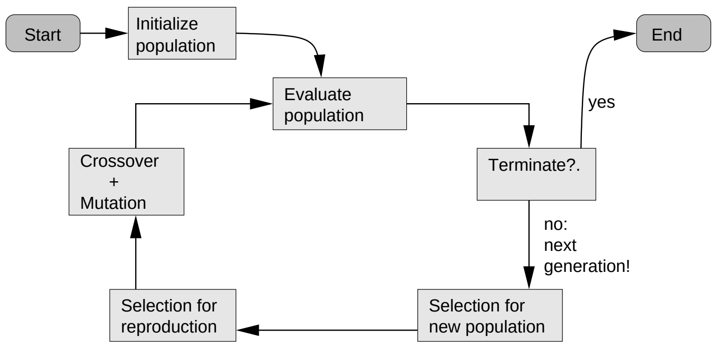

Fig. 3. After the initialization of the start population, the same steps are performed until a termination criterion is fulfilled. In each generation the best individuals (method trees) are selected and reproduced.

main steps are mutation and crossover, which will be discussed in the next section. In each generation the individuals are evaluated with a k-fold cross validation with respect to the learning task at hand. First, the method tree which should be evaluated is used to extract the features from the data. Then the performance of the learning task is estimated with an inner k-fold cross validation. The transformed data is divided into k parts, on $k-1$ parts a classifier is trained and on the last part it is applied. Individuals with a greater performance (fitness) will have a higher probability to survive.

Another fact is interesting for working with high-dimensional data: the building of method trees with genetic programming in order to extract an optimal feature set is like training the optimal feature extraction. We have *two* phases of training. The first phase is the training of a method tree for feature extraction which can be done on a subset of the data. This is especially useful for the great amount of data the high dimensionality brings. Then the best method tree is applied on the complete data and the second training phase starts: a hypothesis is learned from the feature vectors created by the method tree.

3.1 Mutation and Crossover

Mutations are operations which derive a new individual from one other individual. The probability for small distances between parent and child should be greater than for great distances. We use

Generating mutation: Create randomly a new method and adds it at an adequate place in the method tree.

Removing mutation: Remove a randomly chosen method from the method tree. Windowing with overlap $g > 1$ must contain a function.

Changing mutation: Change a randomly chosen method and replaces it with a method from the same group (transformation or function).

Another typical operation for evolutionary algorithms is *crossover*. Here the new individual is derived by combining the informations about two individuals. Crossover is realized by transfering subtrees of the same type between the selected parents.

4 Experiments

We used the discussed approach to extract features from audio data for three different classification tasks:

1. genre classification *classic* and *popular music*: CLA/POP
2. genre classification *techno* and *popular music*: TEC/POP
3. classification of user preference: USER_1, USER_2, and USER_3

The first one is considered an easy task, the other two problems seem to be much harder. CLA/POP contains 100 instances, TEC/POP contains 80 instances, and the USER data sets 50 instances for each class. The methods are implemented within a generic framework for value series preprocessing like the one demanded in Morik and Liedtke (2000). The experiments were done with the learning environment YALE[2] (Fischer et al. (2002) and Mierswa et al. (2003)). Feature extraction for a 60 second sample of music lasts approximately 20 seconds using a 1600 MHz CPU.

4.1 Extracted features from audio data

The following features were extracted from the data sets:

- average loudness
- average distance and variance between extreme values
- average distance and variance between zero crossings
- tempo and variance of autocorrelation
- k highest peaks after a Fourier transformation
- gradient of a linear regression function of the frequency spectrum
- fraction of geometric and arithmetic average of the spectrum
- fraction of maximum and arithmetic average of the spectrum
- average and variance of the strongest frequency in the progressing of time
- average and variance of the angles after a state space reconstruction
- average and variance of the distances after a state space reconstruction

They are described in detail in Mierswa (2003) and were collected during several runs of the genetic programming approach for the classification of audio data. The population size was 10, each mutation probability was 0.2, and crossover probability was 0.4. The maximum number of generations was 100.

[2] `http://yale.cs.uni-dortmund.de`

	CLA/POP	**TEC/POP**	**USER$_1$**	**USER$_2$**	**USER$_3$**
C4.5	1.67%	12.13%	8.12%	9.89%	5.02%
SVM	1.82%	13.22%	7.69%	9.44%	4.81%

Table 1. The classification error for the different classification tasks. For each task, an optimal subset of features was selected.

With a genetic algorithm (Ritthoff et al. (2002)), a subset of these features were selected for each classification task.

The application of a simple 1-R-Learner (Holte (1993)) delivers different features for the data sets. For the classification of CLA/POP, the variance of the distances after a state space reconstruction is the best feature. The created rule correctly classifies 184 of the 200 instances. In the domain TEC/POP the variance of the difference between the extreme values of the series was selected as best feature. The knowledge of this feature alone allows the correct classification of 121 of the 160 instances. Further results of the selection among the audio features are discussed in Mierswa (2003).

4.2 Results

The learning schemes used were the decision tree learner C4.5 (Quinlan (1993)) and a support vector machine (SVM) with a linear kernel function (Joachims (1999) and Rueping (2000)). The classification error was estimated with a 10-fold cross validation. The confidence for decision tree inducing was 0.25 with a minimum leaf size of 2. The support vector machine MySVM was used with default parameters. Table 1 shows the results for the classification tasks.

The genre classification CLA/POP can be done with an error of 1.67%. The more difficult classification of user preferences can be handled with classification errors between 4% and 10%. The genre classification TEC/POP is the hardest discipline among these classification tasks. But the predictions were done with an error of 12.13%.

5 Conclusion

The methods of value series analysis can be divided into groups and systematized. Together with an extended concept of windowing operators these methods can build method trees for feature extraction. It has been shown that the windowing of efficient methods also is efficient, the same applies for method trees. Therefore, these method trees can be used for fast feature extraction from large value series.

The systematization and the efficiency of the methods allow automatic approaches to extract an optimal set of features from value series. The discussed approach is based on genetic programming. The individuals are method trees which work on a subset of the data and are mutated and recombined. The result is a method tree which can be used on the complete data set for feature extraction.

Audio data can be seen as time series with an extraordinary length. We have seen a set of features which was automatically extracted from audio data. With this set of features the reduction of the classification error to nearly 1% for the genre classification classic/popular music was achieved. The prediction of user preferences can be done with an error below 10%.

6 Acknowledgments

This article describes results from my master thesis written at the artificial intelligence chair of the department of computer science of the university of Dortmund. I would like to thank Prof. Dr. Katharina Morik and Dipl.-Inform. Michael Wurst for the outstanding supervision.

References

BRADLEY, E. (1999): Intelligent Data Analysis: An Introduction. In M. Berthold and D. Hand (eds.): *Intelligent Data Analysis: An Introduction.* Springer, Berlin.

FISCHER, S. and KLINKENBERG, R. and MIERSWA, I. and RITTHOFF, O. (2002): YALE: Yet Another Learning Environment. *Technical report CI-136/02, University of Dortmund*

HOLTE, R. C. (1993): Very simple classification rules perform well on most commonly used datasets. *Journal of Machine Learning, 11, 63–90.*

JOACHIMS, T. (1999): Making large-Scale SVM Learning Practical. In: *Advances in Kernel Methods - Support Vector Learning.* MIT Press.

LIU, H. and MOTODA, H. (1998): *Feature Extraction, Construction, and Selection: A Data Mining Perspective.* Kluwer.

MIERSWA, I. (2003): Beatles vs. Bach: Merkmalsextraktion im Phasenraum von Audiodaten. In: *LLWA 03 - Tagungsband der GI-Workshop-Woche Lernen - Lehren - Wissen - Adaptivität.*

MIERSWA, I. and KLINKENBERG, R. and FISCHER, S. and RITTHOFF, O. (2003): A Flexible Platform for Knowledge Discovery Experiments: YALE – Yet Another Learning Environment. In: *LLWA 03 - Tagungsband der GI-Workshop-Woche Lernen - Lehren - Wissen - Adaptivität.*

MITCHELL, M. T. (1996): *Machine Learning.* McGraw Hill, New York.

MORIK, K. (2000): The Representation Race – Preprocessing for Handling Time Phenomena. In: *Proc. of the 11th European Conference on Machine Learning.* Springer, Berlin, 4–19.

MORIK, K. and LIEDTKE, H. (2000): Learning about time. *MiningMart Deliverable No. 3, University of Dortmund.*

PYLE, D. (1999): *Data Preparation for Data Mining.* Morgan Kaufmann.

QUINLAN, R. (1993): *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Diego.

RITTHOFF, O. and KLINKENBERG, R. and FISCHER, S. and MIERSWA, I. (2002): A Hybrid Approach to Feature Selection and Generation Using an Evolutionary Algorithm. *Technical report CI-127/02, University of Dortmund.*

RÜPING, S. (2000): mySVM - Manual. University of Dortmund.

SCHLITTGEN, R. and STREITBERG, B. H. J. (1997): *Zeitreihenanalyse.* Oldenbourg Verlag München.

WITTEN, I. H. and FRANK, E. (2000): *Data Mining.* Morgan Kaufmann, San Diego.