Hindawi Publishing Corporation
EURASIP Journal on Audio, Speech, and Music Processing
Volume 2009, Article ID 153017, 23 pages
doi:10.1155/2009/153017

Research Article

Analytical Features: A Knowledge-Based Approach to Audio Feature Generation

François Pachet and Pierre Roy

Sony CSL-Paris, 6, rue Amyot, 75005 Paris, France

Correspondence should be addressed to François Pachet, pachet@csl.sony.fr

Received 4 September 2008; Accepted 16 January 2009

Recommended by Richard Heusdens

We present a feature generation system designed to create audio features for supervised classification tasks. The main contribution to feature generation studies is the notion of *analytical features* (AFs), a construct designed to support the representation of knowledge about audio signal processing. We describe the most important aspects of AFs, in particular their dimensional type system, on which are based pattern-based random generators, heuristics, and rewriting rules. We show how AFs generalize or improve previous approaches used in feature generation. We report on several projects using AFs for difficult audio classification tasks, demonstrating their advantage over standard audio features. More generally, we propose analytical features as a paradigm to bring raw signals into the world of symbolic computation.

1. Introduction

This paper addresses two fundamental questions of human perception: (1) to what extent are *human perceptual categorization* for items based on *objective* features of these items, and (2) in these situations, can we identify these objective features explicitly? A natural paradigm for addressing these questions is supervised classification. Given a data set with perceptive labels considered as *ground truth*, the question becomes how to train classifiers on this ground truth so that they can generalize and classify new items correctly, that is, as humans would? A crucial ingredient in supervised classification is the *feature set* that describes the items to be classified. In machine-learning research, it is typically assumed that features naturally arise from the problem definition. However, good feature sets may not be directly available, motivating the need for techniques to generate features from the raw representations of objects, signals in particular. In this paper, we claim that the generation of good feature sets for signal-based classification requires the representation of various types of knowledge about signal processing. We propose a framework to demonstrate and accomplish this process.

1.1. Concept Induction from Symbolic Data. The idea of automatically deriving features that describe objects or situations probably originated from Samuel's pioneering work on a program that played the game of checkers [1]. In order to evaluate a position, the program needed a number of features describing the important properties of the board [2]. These features were designed by hand, and Samuel considered the automatic construction of these features to be a major goal for AI [3]. The machine-learning community first addressed automatic feature generation through *supervised concept induction* [4–7]. Several algorithms adapted to problem solvers were developed to automatically infer object descriptions by combining elementary features using construction operators. Most works in inductive concept learning consider feature induction as a *transformational process* (see [3] for a review); new features are induced by operating on existing primitive features. The composition operators are based on general mathematical (e.g., Boolean or arithmetic) functions.

The next step taken in this area was to apply feature generation to *supervised classification*. In this context, classifiers are trained on labeled data sets based on these features. The performance of these classifiers strongly depends on

the feature characteristics, and a great deal of research in machine learning has addressed how to define a "good feature set" for a particular classification problem. Notably, the field of *feature selection* has been widely studied, leading to the development of many algorithms to selectively choose an expressive and compact set of features out of a possibly much larger set [8, 9]. In this field, the notions of feature interaction [10] and feature construction have naturally emerged, both for improving classifier performance and for improving the comprehensibility of the classifiers, for example, for explanation purposes. A number of feature generation algorithms have been proposed [11], resulting in significant improvements in performance over a range of classification tasks, and further establishing the field in machine learning. These works were generalized simultaneously by researchers in several areas. Markovitch and Rosenstein [12] proposed a general framework for feature generation based on a machine-learning perspective. This framework uses a grammar for specifying construction operators. An algorithm iteratively constructs new features and tests them using a training set. This framework was evaluated using the Irvine repository of classification problems, showing that generated features improved the performance of traditional classifiers on various problems. An interesting extension to this approach was applied to text categorization [13]. Krawiec [14] proposed a similar approach from a genetic programming perspective and performed the same evaluation on reference databases.

Despite these favorable results, a careful examination of the generated features in relation to the corresponding grammars reveals that it is relatively easy to generate the "right features" (as known by the researchers) from the grammars. Therefore, the question remains whether this general approach will scale up for concrete cases in which (1) knowledge about the domain cannot be expressed as an efficient grammar, or is otherwise "hidden" in the construction rules, and (2) the input data are not symbols but raw signals.

1.2. From Symbols to Signals. Earlier work addressed problems in which the object features were naturally obtained from the problem definition. In the game of checkers, the basic elements from which features are constructed are the rows and columns of the board. Similarly, reference classification problems (such as [15]) are also described by sets of features which contain all of the relevant problem information.

When the data to be classified consists of raw signals, the situation changes radically. As opposed to symbolic problems in which the data representation is naturally induced by the problem definition, there are no "universally good" features to represent signals.

Raw signals are not suited to the direct use by classifiers for two reasons, *size* and *representation*. In the audio signal domain, signals are represented as time series made up of thousands of discrete samples. Sampling a single second of monophonic audio at 22 kHz results in approximately 44000 16-bit values. Because of the curse of dimensionality [16], this amount of information would require an impractical number of training samples. The representation issue poses a larger problem: the temporal representation of signals (amplitude varying with time) is unsuitable for perceptual coding. For example, the superposition of two sine waves with a phase difference produces very different signals in the temporal domain (see Figure 1). However, since the human ear is insensitive to phase differences [17], it will not perceive any significant difference. In general, most perceptive dimensions (e.g., loudness, timbre) have no obvious signal counterpart [18].

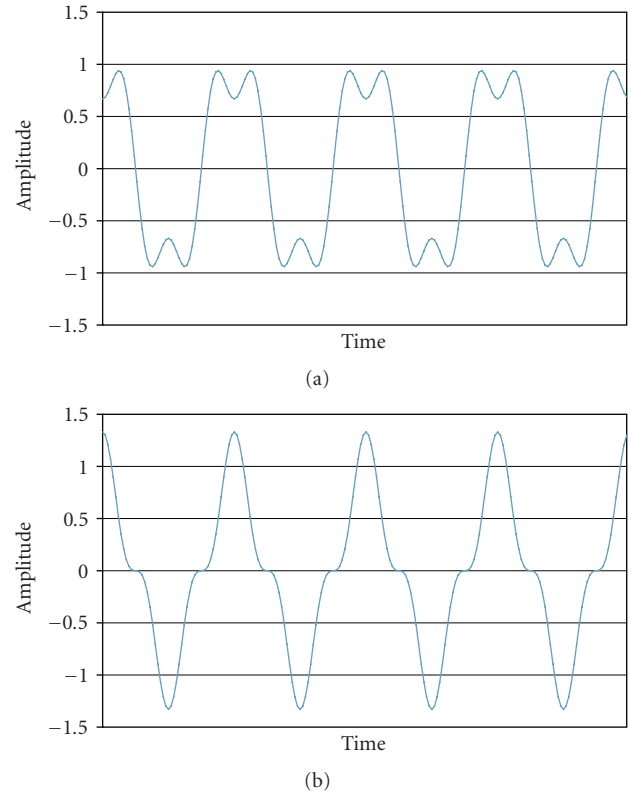

Figure 1: The human ear is insensitive to phase differences, making the temporal representation of audio signals ill-suited for perceptual coding; (a) shows the sum of two sine waves (1 kHz and 3 kHz); (b) 3 kHz sine wave was phase shifted by *pi* before summation. Although the waveforms are substantially different, the ear perceives them as similar.

Most of the work in signal-based classification performed to date exploits general-purpose features which are well known, are well understood, and have a precise mathematical definition. Indeed, technical features for describing signals abound in the huge body of signal processing literature; see, for example, [19]. A limitation to these methods is that the designer of a signal classifier has to select an adequate feature set based on intuition. Although this step is crucial to the design of an overall classifier, it tends to be neglected in the literature. The manual selection of features is a source of suboptimal solutions, and more importantly, gives the whole process of signal classification a debatable epistemological status. How can we interpret the performance of a classifier

which uses manually designed features? In particular, what happens if *other features* had been used instead?

Applying automated feature generation to signal classification is a promising avenue, as it may produce better features than those designed by humans. More importantly, feature generation replaces manual feature design by a systematic search process. Ritthoff et al. [20] proposed to combine feature generation and feature selection algorithms, and applied them to the interpretation of chromatography time series. They confirmed that generated features performed better as a feature set than the raw time series data. Recently, feature generation ideas have spread into many domains of signal classification. Lim et al. [21] describe an approach for hand-written Chinese character recognition, in which high-level signal features are constructed using the explanation-based learning (EBL) framework. The feature construction process in EBL emphasizes the use of domain-specific knowledge in the learning process, using explanations of training examples. Similar improvements were reported with approaches based on general image features. In the engineering domain, Guo et al. [22] describe a feature generation approach to fault classification. They use a small set of transformation operators to produce features of the raw vibration signals from a rotating machine. The same framework was applied to breast cancer diagnosis [23] with similar conclusions. In the biology domain, Dogan et al. [24] describe a feature generation algorithm for analyzing sequences, applied to splice-site reduction. Details on the feature generation algorithm were not provided, but they did report an improvement of around 6% compared with traditional approaches using well-known features. Similarly, speech prosody classification was addressed by Solorio et al. [25], applying ideas from computer vision to the definition of the primitive operators from which features are generated. More recently, Eads et al. [26] generalized their earlier work on lightning classification [27], by proposing a grammar-based approach for generating Matalab code to extract features from time series, similar to the work of Markovitch and Rosenstein [12]. Krawiec and Bhanu [28] extended earlier work [14] on *coevolutionary feature generation*, applying it to 3D-object recognition using a method in which several populations evolve simultaneously.

1.3. Audio Feature Generation. Audio classification has received a great deal of attention due to the explosion of electronic music distribution, and many studies have been devoted to finding good feature sets to classify signals into perceptual categories [29]. Most of this work addresses the issues of feature selection, classifier parameter tuning, and the inherent difficulty in producing reliable "ground truth" databases for training and testing classifiers. General-purpose feature sets have been applied to a great variety of musical classification tasks.

Much previous work has been based on the so-called "bag-of-frames" (BOFs) approach [30]. This approach handles the signal in a systematic and general fashion, by slicing it into consecutive, possibly overlapping frames (typically 50 milliseconds) from which a vector of *short-term* features is computed. These vectors are then aggregated, and fed to the rest of the chain. The temporal succession of the frames is usually lost in the aggregation process (hence the "bag"), and Aucouturier and Pachet [31], Aucouturier [32] showed that preserving the order does not generally improve the performance. First, a subset of available features is identified using a feature selection algorithm. Then, the feature set is used to train a classifier from a database of labeled signals (training set). The classifier is then tested against another database (test set) to assess its performance. The BOF approach is the most natural way to compute a signal-level feature (e.g., at the scale of one entire song).

The BOF approach is highly successful in the music information retrieval (MIR) community, and has been applied to virtually every possible musical dimensions: genre classification [33, 34]; instrument classification [35, 36]; percussion instrument classification [37, 38]; audio fingerprinting [39]; noise classification [40]. It has also been applied to identification tasks, such as vocal identification [41] and mood detection [42]. The BOF approach achieves a reasonable degree of success on some problems. For instance, speech music discrimination systems based on BOF yield almost perfect results [43], as do carefully designed music genre classifiers. However, the BOF approach has limitations when it is applied to more difficult problems. Although classification difficulty is hard to define precisely, it can be observed that problems involving classes with a smaller degree of abstraction are usually much more difficult than others. For instance, we have noticed that genre classification works well to differentiate between abstract, large categories (Jazz versus Rock), but its performance degrades for more precise classes (e.g., Be-bop versus Hard-bop). For such difficult problems, the systematic limitations of BOF, referred to as *glass ceilings* [44], immunize it to all known techniques for improving classifier performance (e.g., feature selection, boosting, parameter tuning).

The realization that general purpose features do not always represent relevant information in signals from difficult classification problems was at the origin of the development of extractor discovery system (EDS), probably the first audio feature generation system [45, 46]. This paper provides a survey of the most important technical contributions of this work regarding current feature generation studies, and reports on its applications to several difficult audio classification problems.

Recently, researchers have recognized the importance of deriving ad hoc domain-specific features for specific audio classification problems. For instance, Mörchen et al. [47, 48] performed an extensive study of a large set of *bag-of-frames* features that were obtained from a cross-product of general short-term features using several temporal aggregation methods. The result was a space of about 40,000 features. Their systematic evaluation showed that some of these features could improve the performance of a music genre classifier, compared with features used in the literature.

An interesting approach to audio feature generation, with similar goals to EDS, was pursued in parallel by Mierswa and Morik [49], based on earlier work on time series feature generation [20]. They proposed a framework for extracting

domain-dependent "method trees", representing *ad hoc* features for a time series. They applied their framework to music genre classification and reported an improvement over approaches that use off-the-shelf audio features [50]. The proposed method trees are essentially BOF features constructed from basic audio operators, with the addition of a complexity constraint (method trees are designed to have polynomial complexity). Using this method, Schuller et al. [51] reported an improvement in classifier performance in the domain of speech emotion recognition.

In this paper, we describe two contributions to the field of signal feature generation, with an emphasis on the most important aspects for successfully generating "interesting" audio features. First, we describe a generic framework, EDS, which has been specialized for the generation of audio features and is used for supervised classification and regression problems. This framework uses *analytical features* to generally represent audio features. The framework bears some similarities with other feature generation frameworks such as those by Markovitch and Rosenstein [12] and Mierswa [50]. Notably, EDS uses genetic programming as a core generation algorithm to explore the function space. However, it differs from other work in signal feature generation in that it was specifically designed to integrate knowledge representation schemes about signal processing. Specifically, we graft *type inference*, *heuristics*, and *patterns* onto analytical features (AFs) to control their generation. Second, we describe several projects using EDS to solve difficult audio classification tasks, thereby extending the usual scope of examples used in the literature. We report on the main results achieved, and emphasize the lessons learned with these experiments.

2. Analytical Features

In this section, we introduce the notion of analytical features (AFs), which represent a large subset of all possible digital signal processing functions having an audio signal as input. We also introduce the algorithms that generate and select AFs for supervised classification and for regression problems.

2.1. Origin. Previous work in musical signal analysis has addressed the automatic extraction of accurate high-level music descriptors. Scheirer's tempo algorithm ([52], see Figure 2) is a prototypical example of a complex function implementing a high-level musical feature (the tempo) from a raw music signal (taken from a CD). This example clearly shows that the design of a good audio feature involves engineering skills, signal processing knowledge, and technical know-how. The initial motivation for EDS originated from the observation that manually designing such a descriptor is costly, yet to a large extent it could be automated by a search algorithm. In fact, a careful examination of Scheirer's seminal tempo paper reveals that many of the design decisions for the tempo extractor were arbitrary, and could be automated and possibly optimized. By examining other high-level feature extractors, it could be seen that there are general *patterns* in the way features are constructed. These observations have led

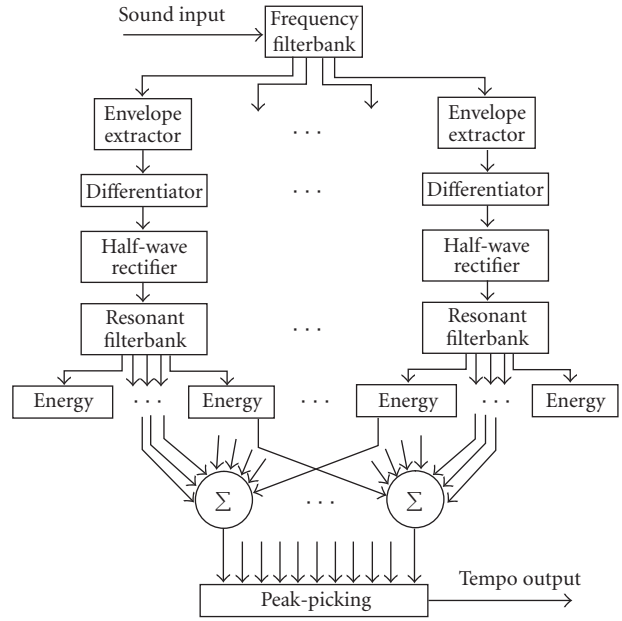

Figure 2: The tempo extractor from Scheirer [52] is a typical example of a high-level audio feature.

to the design of a system that automates the construction of high-level features in the context of supervised classification tasks. The key issue was not so much the algorithm, but rather the encoding of the necessary knowledge to guide this search.

An important source of inspiration was the automated mathematician (AM) system [53]. This system invented mathematical conjectures based on a set of primitive operations which were gradually combined using a set of powerful and general heuristics. Although AM was a milestone in AI, the a posteriori analysis by Lenat and Feigenbaum [54] revealed that it had deep limitations, as summarized in the famous quote, "machines only learn at the fringe of what they already know" [55]. Indeed, AM and related systems could explore only the fringe of the set defined by the initial basic operators. However, AM's success led to the development of genetic programming [56], which is a general method to search function spaces using genetic search [57], and is systematically used in feature generation systems. AM tutorial reconstruction by Haase [58] emphasized a crucial aspect of AM that is at the heart of EDS' design, notably the concept of "representational density". The foundations of this concept are that syntactic constructions should be optimally "interesting", as the system will only concretely explore forms which are relatively easy to generate. This remark was made during the last days of AM, when it was noticed that no interesting conjectures could be invented after the initial frenzy of mathematical discoveries were made; this is the so-called "AM malaise".

The driving force behind the design of EDS and of our primitive audio operators is that the generated features should be optimally interesting (statistically) and simple (syntactically). To this end, the feature generation should

exploit a large amount of knowledge about signal processing, as was demonstrated by Scheirer's tempo example. Without this knowledge, the feature space generated would be too small to contain interesting features, or would include features which are only *superficially better* than well-known ones. Moreover, some kernels (e.g., Boolean or arithmetic) embedded in classifiers (such as support vector machines, SVMs) are able to automatically reconstruct implicit feature combinations at the training phase [59]. These may incorporate domain-specific knowledge [60], and they eliminate the need for an extra and costly feature generation phase. Therefore, the primary task for feature generation is the construction of features containing in-depth signal information that cannot be extracted or reconstructed otherwise. This "representational density" argument was translated into a series of important extensions to the basic genetic programming algorithm. These are described in the next section, beginning with the definition of our core concept: *analytical features*.

2.2. Analytical Features: A Subset of All Possible Audio Digital Signal Processing Functions. Analytical Features (AFs) are expressed as a functional term, taking as only argument the signal as an input (represented here as x). This functional term is composed of basic operators, and its output is a value or vector of values that is automatically typed from the operators using a *dimensional type inference system*. The feature's value is designed to be directly fed to a classifier, such as a SVM, using a traditional train/test process.

AFs depart from approaches based on heterogeneous data types and operations, such as that of Mierswa [50], who distinguishes between basis transformations, filters, functions, and windowing. In contrast, AFs are designed with these simplicity constraints: (1) only one composition operation is used (functional composition), and (2) AFs encompass the whole processing chain, from the raw signal to the classifier. The processing chain includes in particular *windowing operations*, which are crucial for building signal features in a BOF approach [49]. As explained above, a signal of substantial length (say, more than a few hundred milliseconds) is frequently sliced into frames rather than being treated as a single time series. Features are then extracted for each frame, and the resulting vector set is aggregated using various statistical means, such as statistical moments or Gaussian mixture models [47].

These operations are all included in the definition of an AF, through a specific operator *Split*. For instance, the following AF (1) computes the mel-frequency cepstrum coefficients (MFCC) of successive frames of length 100 (samples) with no (0) overlap, and then computes the variance of this value vector:

$$\text{Variance}\,(\text{MFCC}\,(\text{Split}\,(x, 100, 0))). \tag{1}$$

This uniform view on windowing, signal operators, aggregation operators, and operator parameters introduces a great deal of flexibility in feature generation. It also simplifies the entire generation phase by eliminating the choice of specific windowing or aggregation parameters. Note that AF are not, by design, limited to BOF features (i.e., statistical reductions of short-term features), nor are they limited to polynomial-complexity features as are other feature generation approaches [49]. AFs can express arbitrary signal functions, and might not make obvious sense. For instance, the following AF (2) computes a complex expression involving a low-pass filter, whose cutoff frequency is itself computed as the maximum peak of the signal spectrum:

$$\begin{aligned}\text{MaxPos}(\text{FFT}(\text{Split}(\text{FFT}(\text{LpFilter}(x,\text{MaxPos}(\text{FFT}(x))),\\ 100, 0))),\end{aligned} \tag{2}$$

where FFT indicates a fast Fourier transform. To be manipulated efficiently, each AF is associated with a *dimensional type*, inferred automatically by a type inference system. These types, introduced in the next section, form the basis for implementing random function generators, for defining powerful heuristics, and for guiding the search through feature patterns.

It is important to note that AFs, by definition, do not capture all possible digital signal processing (DSP) functions that could be used as audio features, that is, that reduce a signal to a lower-dimensional vector. For instance, most programs written using an imperative programming style cannot be expressed as AFs. It is difficult to give a precise semantic definition of the space covered by AFs (this is an interesting avenue of research currently being investigated). As we will see below, Scheirer's tempo extractor can be reasonably well represented as an AF, as well variations about this feature. But it is easy to devise other DSP functions which cannot be expressed as AFs. We chose to restrict our investigation to AFs for a number of reasons.

(1) The space of "reasonable size" AFs is huge, and likely contains features which are sufficiently good for the problem at hand. The number of AFs of size (number of operators) less than 10 is estimated to 10^{20} [61], even without accounting for the combinatorial exploration of operator parameters.

(2) Thanks to their homogeneity and conceptual simplicity, AFs can be easily typed, which eases the implementation of control structures (e.g., heuristics, function patterns, and rewriting rules), as explained below. Without these control structures, the generation can only superficially improve on existing features. Consequently, AFs are a good compromise between BOF features, as generalized by Mierswa [50], and arbitrary DSP programs [26], whose automatic generation is notoriously difficult to control.

(3) Many non-AF functions can be approximated by AFs or by sets of AFs. We give an example in Section 3.1.2, where EDS is used as a regression classifier to approximate a known audio feature that is not included in the basic operator set.

2.3. A Dimensional Type Inference System. The need for typing functions is well known in genetic programming (GP), and ensures that the generated functions are at least

syntactically correct. Different typing systems have been proposed for GP, including strong typing [62], which explicitly represents the "programming" types (floats, vectors, or matrix) of the inputs and outputs of functions. However, for this application, programming types are superficial and are not needed at the AF level. For example, the operator absolute value (*Abs*) can be applied to a float, a vector, or a matrix; this polymorphism provides needed simplicity in the function expression. However, we do need to distinguish how AFs handle the signals in terms of their *physical dimensions*, since knowledge about DSP operators naturally involves these dimensions. For instance, audio signals and spectrum can both be seen as *float* vectors from the usual typing perspective, but they have different dimensions. A signal is a time-to-amplitude function, whereas a spectrum is a frequency-to-amplitude function. It is important for the heuristics to be represented in terms of physical dimensions instead of programming types.

Surprisingly, to our knowledge, there is no type inference system for describing the physical dimensions of signal processing functions. Programming languages such as Matlab or Mathematica manipulate implicit data types for expressing DSP functions, which are solely used by the compiler to generate machine code. In our context, we need a polymorphic type inference system that can produce the type of an arbitrary AF from its syntactic structure. The design of a type inference system has been well addressed in theoretical computer science [63]. The choice of the primitive types is again determined by the representational density principle, in which the most used types should be as simple as possible so that heuristics and patterns can be expressed and generated easily. Accordingly, we define a primitive type system and type inference rules to apply to each operator in the next section.

2.3.1. Basic Dimensional Types. In the current version of EDS, we have chosen a dimensional type system based on only three physical dimensions: time "t", frequency "f", and amplitude or nondimensional data "a". All of the AF data types can be represented by constructing atomic, vector, and functional types out of primitive types.

"Atomic" types describe the physical dimension of a single value. For instance:

(i) position of a drum onset in a signal: "t",

(ii) cut-off frequency of a filter: "f",

(iii) amplitude peak in a spectrum: "a".

"Functional" types represent data of a given type, which yield a dimension of a different type. The target type is separated from the data type using the ":" notation. For instance,

(i) audio signal (amplitude in time): "$t : a$",

(ii) spectrum (amplitude in frequency): "$f : a$".

"Vector" types, notated "V", are special cases of functions used to specify the types for homogeneous sets of values. For instance,

(i) temporal positions of the autocorrelation peaks of an audio signal: "Vt"

(ii) amplitudes of autocorrelation peaks: "Va".

Vector and functional notations can be combined. Currently the type system is restricted to two-dimensional constructs. For instance:

(i) a signal split into frames: "$Vt : a$",

(ii) autocorrelation peaks for each frame: "VVa"

(iii) types like "$VVVa$" are currently not implemented.

Note that this notation is equivalent to the traditional notation of type inference, where ":" is notated " $\rightarrow$ " and "V" is notated "[]". Also, there are many ways to represent DSP function dimensions. For instance, the physical dimension of frequency is the inverse of time, and we could use a dimension system with exponents so that "f" is replaced by "t^{-1}". However, this involves a more complex type inference system (e.g., the type of *Energy* would be a^2). More importantly, we did not identify knowledge (heuristics or patterns) that would require such a detailed view of dimensions. But our choice is arbitrary and has limitations, as discussed in Section 4. Hereafter, in this paper, the word "type" will mean "dimensional type".

2.3.2. Typing Rules. For each operator, we define typing rules so that the type of its output data is a function of its input data types. For instance, the *Split* operator transforms

(i) a signal ("$t : a$") into a set of signals ("$Vt : a$");

(ii) a set of time values ("Vt") into multiple sets of time values ("VVt").

These rules are defined for each operator, so that arbitrary functions types can be inferred automatically by the type inference system. For instance, the following AF is typed automatically as follows (types are written as a subscript):

$$\mathrm{Min}_{\_a}(\mathrm{Max}_{\_Va}(\mathrm{Sqrt}_{\_Vf:a}(\mathtt{FFT}_{\_Vf:a}(\mathtt{Split}_{\_Vt:a}(\mathtt{x}_{\_t:a},1024))))). \tag{3}$$

(In traditional type notation: $\mathrm{Min}_a(\mathrm{Max}_{[a]}(\mathrm{Sqrt}_{[f->a]}(\mathtt{FFT}_{[f->a]}(\mathtt{Split}_{[t->a]}(\mathtt{x}_{t->a},1024)))))$.) This AF yields an amplitude value "a" from a given input signal x of type "$t : a$".

Equipped by a type system, EDS can support constructs for pattern-based generators, heuristics, and rewriting rules. These are described in the following sections.

2.4. The AF Generation Algorithm. AFs are generated with a relatively standard function search algorithm. The main goal is to quickly explore the most promising areas of the AF space for a given problem. Like many feature generation approaches, our AF generation algorithm is based on genetic programming [56], one of the most convenient methods to search a function space. The general framework is illustrated in Figure 3. It bears some similarity with the framework proposed by Markovitch and Rosenstein [12], and other

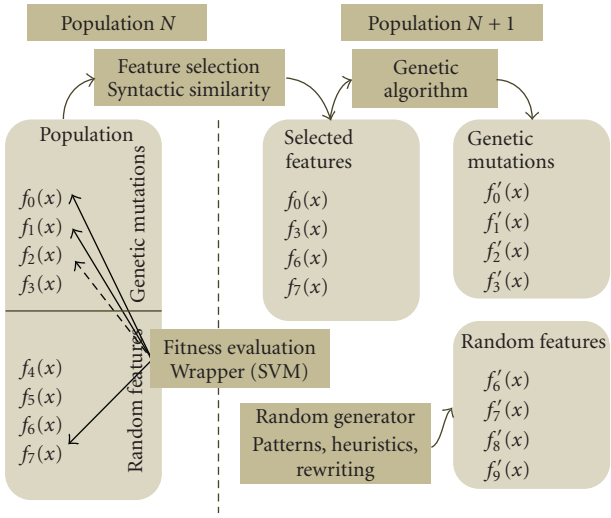

FIGURE 3: The architecture of AF generation. AFs are evaluated individually using a wrapper approach. AFs are selected for the next population using a syntactic feature selection algorithm.

frameworks proposed since in the literature, such as Mierswa and Morik [49]. As described in Section 2.2, AFs are designed specifically to support a series of mechanisms that represent operational knowledge about signal processing operators (see Section 2.5). From a machine-learning perspective, most feature generation frameworks use a *wrapper approach* to evaluate generated features on the training set. Our approach also differs in the way that AFs are evaluated (individually in our case), and in how AF populations are successively selected using a feature selection algorithm based on the syntactic structure of AFs, as described in the next section.

2.4.1. Main Algorithm. The algorithm works on audio description problems composed of a database containing labeled audio signals (numeric values or class labels). The algorithm builds an initial population of AFs, and then tries to improve them through the application of various genetic transformations. The precise steps followed by the algorithm are the following:

(1) build an initial population P_0 using random AFs, possibly constrained by patterns,

(2) compute the *fitness* of each AF in the population. This fitness depends on the nature of the problem and other criteria, as explained in Section 2.4.3,

(3) if (*Stop Condition*): STOP, then RETURN the best AFs,

(4) else, select the AFs with the highest fitness, and apply transformations to them to create a population P_{i+1},

(5) return to step (2) and repeat.

Arbitrary stop conditions can be specified. For example, a stop condition can be defined as a conjunction of several criteria.

(i) The maximum number of iterations is reached. The search stops automatically after N populations.

(ii) The fitness of the population converges and ceases to improve. That is, the fitness of the best function of population P_i = the fitness of the best function of population P_{i-N}, and this condition occurs after, say, $N = 5$ unimproved populations.

(iii) An optimal AF is found. That is, the fitness is maximal.

2.4.2. Genetic Operations. New populations are created by applying genetic transformations to the best fitted functions of the current population. These operations are relatively standard in genetic programming. In addition to selection, five transformations are used in EDS: cloning, mutation, substitution, addition, and crossover.

(i) *Cloning* maintains the tree structure of a function and applies variations to its constant parameters, such as the cut-off frequencies of filters or the computation window sizes. For example,

`Sum(Square(FFT(LpFilter(Signal, 1000 Hz))))`
can be cloned as
`Sum(Square(FFT(LpFilter(Signal, 800 Hz)))).`

(ii) *Mutation* removes a branch of a function and replaces it with another composition of operators of the same type. For example,

`Sum(Square(FFT(LpFilter(Signal, 1000 Hz))))`
can be mutated into
`Sum(Square(FFT(BpFilter(Normalize(Signal), 1100 Hz, 2200 Hz)))).`

(iii) *Substitution*, a special case of mutation, replaces a single operator with a type-wise compatible one. For example,

`Sum(Square(FFT(LpFilter(Signal, 1000 Hz))))`
can be replaced by:
`Sum(Square(FFT(BpFilter(Signal, 1100 Hz, 2200 Hz)))).`

Substitution typically corresponds to what a sound engineer would do when replacing a low-pass filter by a band-pass filter (see example above). Mutation, on the other hand, corresponds to a more general principle: replacing an elementary task by a more complex process, or the way around.

(i) *Addition* adds an operator to form a new root of the feature. For example,

`Sum(Square(FFT(Signal))`is an addition of
`Square(FFT(Signal)).`

(ii) *Crossover* cuts a branch from a function and replaces it with a branch cut from another function. For example,

`Sum(Square(FFT(Autocorrelation(Signal))))`
is a crossover between
`Sum(Square(FFT(LpFilter(Signal, 1000 Hz))))`
and `Sum(Autocorrelation(Signal))`.

In addition to the genetically transformed functions, the new population contains a set of new randomly generated AFs, thereby ensuring its diversity and introducing new operations to the population evolution. Random AF generators are described in Section 2.5.2.

The distribution of the features in a population is as follows: 25% are randomly generated; 10% are clones (of the features kept from the previous generation); 10% are mutations; 10% are additions; 10% are crossovers; 10% are substitutions; 25% are random.

2.4.3. Evaluation of Features and Feature Sets. The evaluation of features is a delicate issue in feature generation, as it is well-known that good individual features may not form a good feature set when they are combined with others, due to feature interaction. In principle, only feature sets should be considered during the search process, as there is no principled way to guarantee that an individual feature will be a good member of a given feature set. The approach taken by Markovitch and Rosenstein [12] is conservative in this sense, as feature interaction is considered up-front during the search. However, they point out that there is both a risk to narrowing the search and a high evaluation cost.

With experience based on our experiments with large-scale feature generation, we chose another option in EDS, in which we favor the exploration of large areas of the AFs space. Within a feature population, the features are evaluated individually. Feature interaction is considered during the selection step used to create new populations.

Individual Feature Evaluation. There are several ways to assess the fitness of a feature. For classification problems, the Fisher Discriminant Ratio [64] is often used because it is simple to compute and reliable for binary classification problems. However, it does not adapt to multiclass problems, in particular those with nonconvex distributions of data. To improve feature evaluation, we chose a wrapper approach to feature selection [65]. In this approach, features are evaluated using an SVM classifier built during the feature search, and there is a 5-fold cross-validation on the training database. The fitness is assessed from the performance of a classifier built with this unique feature. As we often deal with multiclass classification (not binary), the *average F-measure* is recommended to assess the classifier's performance [66]. However, as training databases are not necessarily balanced class-wise, the average F-measure can be artificially high. Therefore, the fitness in EDS is defined by an *F-measure vector* (one F-measure per class) of the wrapper classifier. For regression problems, we use the Pearson correlation coefficient [67]. Other methods also could be applied, such as a wrapper approach with a regression SVM.

Note that training and testing an SVM classifier on a single scalar feature require little computation time. Indeed, the fitness computation is generally much faster than the actual feature extraction.

Feature Set Evaluation: Taking Advantage of the Syntactic Form of AFs. After a population has been created and each feature has been individually evaluated, we need to select a subset of these features to be retained for the next population. Any feature selection algorithm could be used for this purpose, such as InfoGain [68]. However, feature selection algorithms usually require a calculation of redundancy measures, for example, by computing correlations of a feature's values across samples [9]. As our features are all AFs, we can take advantage of their syntactic expression to efficiently compute an approximate redundancy measure. This is possible because syntactically similar AFs have (statistically) correlated value series [69]. Also, our algorithm considers the performance of features in each class, rather than globally for all classes.

Finding an optimal solution in a problem requires a costly multicriteria optimization. As an alternative, we propose a low-complexity algorithm as a one-pass selection loop. First the best feature is selected, and then the next best feature that is not redundant is iteratively selected, continuing until the required number of features is reached. The algorithm cycles through each class in the problem, and accounts for the redundancy between a feature and the currently built feature set, using the syntactic structure of the feature. The algorithm is stated (in its simplest form) as in Algorithm 1.

The syntactic correlation (`s-correlation`) between two features is computed based on their syntactic form. This not only speeds up the selection, but also forces the search algorithm to find features with a greater diversity of operators. S-correlation is defined as the tree edit-distance [70] between two AFs. The edit distance uses specific edit operation costs, taking into account the AF's types. More precisely, the cost of replacing operator *Op1* by *Op2* in an AF is

```
if Op1 = Op2 return 0
        else if (type(Op1) == type (Op2)) return 1
                else return 2
```

To yield a Boolean s-correlation function, we compute the *edit distance* for all pairs of features in the considered set (the AF population in our case), and get the maximum (`max-s-distance`) values for these distances. S-correlation is defined as

```
s-correlation (f,g):
        return tree-edit-distance (f,g)
                ≤ 1/2∗max-s-distance
```

With this procedure, our mechanism can efficiently evaluate individual features, allowing the exploration of a larger feature space. It also ensures a syntactic diversity within feature

```
FS ← {}; the feature set to build
For each class C of the classification problem
        S ← {non-selected features, sorted by decreasing performance wrt C};
        For each feature F in S
                If (F is not s-correlated to any feature in FS)
                FS ← FS + {F}; Break;
        If (FS contains enough features) break;
Return FS;
```

Algorithm 1

populations. A comparison of this feature selection scheme with an information-theoretic approach was performed for the Pandeiro study, described in Section 3.3.

The syntactic correlation is used for the feature selection, after the generation process. Using the syntactic correlation *during* the generation is an open problem that is not investigated in this article.

2.5. Function Patterns and Function Generators . Genetic programming traditionally relies on the generation of random functions to create the initial population and to supplement the population at each iteration. However, relying on random generation only, our experiments showed that feature generation algorithms only explore a superficial region of the search space, and do not find really novel features in many cases. This limitation can be overcome by providing specific search strategies to the system based on the designer's intuitions about particular feature extraction problems.

2.5.1. Patterns. To perform specific search *strategies*, we constrain the random generation so that the system explores specific areas of the AF space. Although there is no known general paradigm to extract relevant features from the signal, the design of such features usually follows regular *patterns*. One pattern consists of filtering the signal, splitting it into frames, applying specific treatments to each frame, and aggregating the results to produce a single value. This occurred in the beat tracking system described by [52]. That system includes an expansion of the input signal into several frequency bands, followed by a treatment of each band, and concluded with an aggregation of the resulting coefficients using various aggregation operators, ultimately yielding a float value representing (or strongly correlated to) the tempo.

To represent this kind of knowledge, we introduce the notion of *function pattern*. A function pattern is a regular expression denoting subsets of AFs that correspond to a particular building strategy. Syntactically patterns look like AFs, with the addition of regular expression operators such as "!", "?", and "$*$". Patterns make use of types to specify the collections of targeted AFs in a generic way. More precisely (the current system uses additional regular expression tokens not described here, notably to control the clarity of operators):

(i) "?_τ" stands for one operator whose type is τ,

(ii) "$*$_τ" stands for a composition of several operators, all of which have types τ,

(iii) "!_τ" stands for several operators whose final type is τ (the types of the other operators are arbitrary).

For instance, the pattern "!_a" represents all scalar AFs, for example, $Zcr(x)$, and the pattern "!_Va" represents all scalar vector patterns, for example, *Mfcc(split(x, 521)*.

The following pattern represents a construction strategy which abstracts the tempo extractor strategy investigated manually by Scheirer [52]:

$$?\_a\,(!\_\,\_Va\,(\text{Split}(*\_t:\ a(x)))). \tag{4}$$

This pattern can be paraphrased as follows.

(1) "Apply some signal transformations in the temporal domain" ($*\_t : a$).

(ii) "Split the resulting signal into frames" (*Split*).

(iii) "Find a vector of characteristic values, one for each frame" (!_Va).

(iv) "Find one operation that aggregates a unique value for the entire signal" (?_a).

It can be instantiated by the following concrete AFs (5):

$$\begin{aligned}&\text{Sum\_}a\,(\text{Square\_}\,Va(\text{Mean\_}Va(\text{Split\_}\,Vt:a(\text{HpFilter\_}t:a\\ &\qquad(x\_t:a,1000\ \text{Hz}),100))))\\ &\quad\text{Log}\,10\_a(\text{Variance\_}a\,(\text{NPeaks\_}Va(\text{Split\_}Vt:a\\ &\qquad(\text{Autocorrelation\_}\,t:a(x\_t:a),100),10))).\end{aligned} \tag{5}$$

Another typical example of a pattern, referred to as the BOF pattern, is

$$(\text{BOF})\ ?\_a(?\_Va(\text{Hanning}(\text{Split}(x,512,0.5)))).$$

This pattern imposes a fixed windowing operation with a *Hanning* filter, on successive frames of 512 samples with a 50% overlap, and then allows one operation to be performed for each frame (either in the temporal or spectral domain). This is followed by an aggregation function, which is typically a statistical reduction (including operators such as *Mean*, *Variance*, *Kurtosis*, and *Skewness*). This pattern corresponds approximately to the set of about 66 000 features described and explored manually by Mörchen et al. [47].

It is difficult to propose patterns corresponding exactly to the method proposed by Mierswa [50], due to lack of information about their system. However, it is likely that their "method trees" could be reasonably approximated by one or several AF patterns.

Another interesting example of a pattern is the BOBOF pattern, in which windowing of various sizes is chained together to produce a nonlinear aggregation method from the short-term feature vectors:

$$\text{(BOBOF)}\quad ?\_a(?\_Va(\text{Hanning}(\text{Split}(?\_Va(\text{Hanning}(\text{Split}(x, 512, 0.5)))), 1024))).$$

Another pattern example is more complex:

$$\text{(Another)}\ !\_Vt(!\_f : a(?\_t : a(x))).$$

It consists of transforming the signal to the spectral domain and back to the temporal domain, eventually yielding a time series. This pattern may be instantiated by the following AFs:

$$\text{PeakPos}\_Vt(\text{FFT}\_t : a(\text{FFT}\_f : a(\text{Hanning}\_t : a(x\_t : a\_)))) \text{ or}$$
$$\text{PeakPos}\_Vt(\text{FFT}\_t : a(\text{Triangle}\_f : a(\text{FFT}\_f : a(\text{Arcsin}\_t : a(x\_t : a))))).$$

Patterns are used by AF generators to generate correspondingly random AFs. The systematic generation of all possible concrete AFs for a given pattern is a difficult task, but is probably not needed. Instead, we designed a random generator that generates a given number of AFs satisfying a given pattern, as described in the following section.

2.5.2. Pattern-Based AF Generators. Most patterns can, in principle, generate arbitrarily complex features. For several reasons, we generate only *simple* features from a given template, that is, the features should use the fewest possible type transitions and should not be too long. The generated features are kept simple since the subsequent genetic operations will later introduce complexity into them.

Technically, the main problem is the pattern "$!\_\tau$", because it allows arbitrary type transitions. In its first phase, our algorithm rewrites the pattern to an explicit type transition, so that the final pattern no longer contains "$!\_\tau$". Each "$!\_\tau$" has an input type given by its child and an output type τ. The explicit type transition is found by computing the shortest type transition path between the input and output types. This shortest path algorithm uses a type transition table that contains all of the valid type transitions (some type transitions are not possible for a given operator library).

For instance, the pattern $!\_Vt(!\_f : a(?\_t : a(x)))$ will be rewritten as $*\_Vt(*\_t : a(*\_f : a(?\_t : a(x))))$ since no single operator in the default library can transform a structure of type $f : a$ into a vector of type Vt. In other words, the shortest path from $f : a$ to Vt in the type transition graph is $f : a \rightarrow t : a \rightarrow Vt$.

Once the pattern is rewritten in this form, it can be completed by randomly drawing operators corresponding to each variable type. For the case of "$*\_\tau$", a random number n of operators between 1 and 10 are drawn first, followed by n operators that are randomly drawn and chained together.

2.5.3. Heuristics. Most of the approaches used for feature generation have been purely algorithmic. Genetic programming is applied as the search paradigm, and generates functions from primitive operators. The current literature lacks descriptions about the details of the search sessions produced. It appears that only relatively "simple" features are generated, considering the information given in the primitive attributes, the construction process [50], or in the grammars [12].

Our AF framework does not behave in this way. Based on random generation and the genetic transformation operators as a search paradigm, our observations show that AFs rarely converge to really interesting features, unless these features are found quickly within the very first iterations of the search.

Another observation is that real world problems described in the literature only partially reuse the proposed frameworks (see, e.g., the text application by Gabrilovitch and Markovitch [13], the work of Markovitch and Rosensein [12], the application by Mierswa [50], or the speech emotion classification by Schuller et al. [51]). This indicates that more than frameworks is needed to build effective features, namely, heuristics.

We introduce explicit heuristics to guide the search. These heuristics represent know-how about signal processing operators. Heuristics can promote a priori interesting functions, or eliminate obviously noninteresting ones. They are a vital component of EDS, as they were in AM. A heuristic in EDS is a function that gives a score to an AF, ranging from 0 (forbidden) to 10 (very recommended). The score is determined prior to the AF's effective construction and integration into a population. These scores are systematically used when EDS builds a new AF, to select the candidates from all of the possible operations. In the current version of EDS, these heuristics were designed and implemented manually. The most interesting and powerful heuristics are the following.

(i) *Control the Structure of the Functions.* AFs can in principle have arbitrarily complex forms, including the form for the arguments of operators (see AF (2) in Section 2.2). However, it is rare that a very complex function is needed to compute the value of a scalar argument (e.g., the cut-off frequency for a high-pass filter). A heuristic can be used instead, where x is the input signal, and *Branch* represents the sub-AF of the argument of a *HpFilter* operator, considered as a potential argument for *HpFilter*

$$!\_a(\text{HpFilter}(!\_t : a(x), \text{Branch}) => \text{Max}(0, 5 - \text{Size}(\text{Branch})).$$

The resulting AF will be scored 5 if *Branch* is a constant operator, 4 if its length is 1, and so on.

(ii) *Avoid Bad Combination of Operations.* There are specific combinations of operators that we know are not of interest.

For instance, multiple high-pass filters can be avoided using the heuristic

$$!\_a(\text{Split}(\text{Split}(!\_t : a(x), !\_a))) => 1.$$

This heuristic considers two consecutive Split operations to be a bad composition. Note that this heuristic differs from rewriting rules, which will simply combine filters.

(iii) *Range Constant Parameter Values.* Some heuristics control the range of parameter values for some operators. For example, the following heuristic

$!\_a(\text{Envelope}(!\ -a(x), \text{Cst} < 50\,\text{frames}) => 1$ governs the size of the window when computing an envelope (Cst represents a constant value), and

$!\_a(\text{Hpfilter}(x, \text{Cst} < 100\,\text{Hz})) => 1$ governs the cut-off frequency value of a filter.

(iv) *Avoid Too Many Operator Repetitions.* It is frequently useful to compute the spectral representation of a signal (*FFT* in concrete cases). In signal processing, it is also common for an operation to be repeated twice. For instance, *MFCC* coefficients can be seen as a "double *FFT*" of a signal. However, it seems unlikely that three consecutive applications of the *FFT* could generate interesting data. This idea is easily represented as a heuristic, and can be programmed explicitly using the number of occurrences of a given operator (e.g., *FFT*). The pattern can be written as: $!\_a(\text{FFT}(!\_t : a(\text{FFT}(!\_f : a(\text{FFT}(!\_t : a(x)))))) => 1$.

(v) *Avoid Too Many Type Repetitions.* The ideas above can be applied to types instead of concrete operators. In particular, we wish to disregard compositions containing several (i.e., more than 3) operators of the same type, in particular when this type is scalar. For instance, expressions like (Abs(Sqrt(Log(Sqrt(... contain many repetitions of operators of type "*a*", and are probably less interesting to explore than AFs with a more balanced distribution of types:

$$?\_a(?\_a(?\_a(!\_t : a(x)))) => 1.$$

(vi) *Favor Particular Compositions.* There are recommended compositions of operators; for example, it is useful to apply a windowing function to each frame after a Split:

$$!\_a(\text{Split}\,(\text{Hanning}(!\_t : a(x)))) => 8.$$

Note that this heuristics can be generalized to any operator which does not alter the type of its input (usually a signal, "*t:a*").

All of these heuristics can be considered as a manual bootstrap that makes the system operational. Ideally, these heuristics could be learned automatically by a self-analysis of the system. Some work has begun in this domain (see Section 4).

2.5.4. Rewriting Rules. Rewriting rules simplify functions prior to their evaluation and speed up the search. Rewriting rules are rudimentary representations of DSP theorems. Unlike heuristics, they are not used by the genetic algorithm to favor combinations, but they do impact the search by

(i) avoiding the need to compute a function multiple times with different but equivalent forms. For example,

$$\text{Correlation}(x, x) => \text{Autocorrelation}(x), \text{ or}$$
$$\text{HpFilter}(\text{HpFilter}(x, a), b) => \text{HpFilter}(x, \max(a, b));$$

(ii) reducing the computational cost. For example, Parseval's equality ([71], see below) avoids computing a possibly costly *FFT* of a signal.

The rules are triggered iteratively using a fixed-point algorithm until a normal form is obtained [72]. The confluence of our rule set has not been proven, but is likely to occur as symmetry was avoided. Here are the most useful rewriting rules used by EDS:

(i) $\text{Abs}(\text{Abs}(X)) => \text{Abs}(X)$,

(ii) $\text{Hpfilter}\ (X, 0) = X$,

(iii) $\text{Envelope}\ (X, 1) = X$,

(iv) $\text{Envelope}(\text{Abs}(X), w) => \text{Envelope}(X, w)$,

(v) $\text{Abs}(\text{FFT}(X)) => \text{FFT}(X)$,

(vi) $\text{Abs}(\text{Square}(X)) => \text{Square}(X)$,

(vii) (Parceval) $\text{Mean}(\text{FFT}(\text{Square}(X)) = \text{Sum}(\text{Square}(X))$,

(viii) $\text{Var(X)} = \text{Mean}(\text{Square}(X - \text{Mean}(X))) = \text{Mean}(\text{Square}(X) - \text{Square}(\text{Mean}(X))$,

(x) $\text{Max}(\text{Autocorr}(X)) = \text{Sum}(\text{Square}(X)) = \text{Autocorr}(X)(0) = E(X)$.

2.5.5. Computation of AFs. In contrast to most approaches in feature generation, AFs are not computed in isolation, but are computed globally for a whole population. This approach reduces the number of computations by exploiting the redundancies between features in each population. When a population is generated, a tree representing the set of all AFs is also generated. Each subsequent set of operators is then computed once (for each sample in the training set), using a bottom-up evaluation of the population tree. The increase in computational efficiency ranges from a factor of 2 to 10, depending on the patterns used and the population variability and size. This computation can be easily distributed to several processors.

2.6. Basic Operators: The "General" Library. The design of the primitive operators is essential, but this topic has yet not been addressed in the feature generation literature. Currently this choice must be done by the user, for example, by the use of a grammar. This choice is complex because many of the features generated by traditional techniques can now

be produced systematically by kernel-based classifiers. For instance, features generated by Boolean or simple mathematical expressions can be easily reconstructed using Boolean kernels [73]. Conversely, the representational density principle suggests that generated features should be kept as simple as possible, analytically. Operators should therefore embody signal characteristics that are useful for general purpose feature construction. For instance, a "filter" operator that applies to any signal will be useful. Conversely, too specific operators will not lead to interesting constructions. An operator such as "log-attack-time" is not relevant outside of specific attack detection applications, so it was not included. A mathematical "plus" operator is also problematic, as it will generate an exponential growth in the feature space size and will be difficult to control.

Here follows a description of operators in the basic version of EDS. Note that a signal is considered as a vector of values (following the time/amplitude representation). In the following descriptions, we generally use the term "vector of values". We use "signal" only when it makes the description clearer.

Abs: the absolute value

Arcsin: the arc-sinus

AttackTime: the duration of the attack of a signal does a low-pass filter at 50 Hz and then computes the duration in seconds of the attack (the attack is for amplitudes between 20% and 80% of the max value of the signal)

Autocorrelation: the autocorrelation of a signal computed using the FFT

Bandwidth: the width of the highest peak at a percent threshold of the peak; the threshold is a parameter of this operator

BarkBands: computes the sums of amplitudes in filtered Bark bands covering the whole spectrum (all bands have the same size in terms of Bark pitches)

Bartlett: applies a Bartlett filter

Blackman: applies a Blackman filter

BpFilter: band-pass filter

Centroid: the Centroid of a vector of values

Chroma: computes the average of amplitudes of a signal in filtered pitch-bands covering the whole spectrum; all bands have the size of one pitch, and all pitches are wrapped onto a single octave

Correlation: correlation computed using FFT

dB: dB scale: $20 \times \log_{10}(x)$

Differentiation: the first derivative of a vector of values: the vector containing the differences of successive elements in the input vector

Division: divides every value by a constant parameter

Envelope: the outline of the signal. It is a signal that connects all the peaks of the input signal. We calculate it using the Hilbert transform

FFT: the discrete Fourier transform of a signal, based on FFTW

FilterBank: bandpass filters the signal across n band of equals width

Flatness: measures the flatness of a signal

Hamming: applies a Hamming filter

Hann: applies a Hann filter

Hanning: applies a Hanning filter

HarmSpectralCentroid: the Harmonic spectral Centroid is computed as the average over of the *instantaneous* harmonic spectral Centroid within a running window. The instantaneous spectral Centroid is computed as the amplitude (linear scale) weighted mean of the harmonic peaks of the spectrum

HarmSpectralDeviation: the Harmonic Spectral Deviation is the average of the *instantaneous* harmonic spectral deviation within a running window. The instantaneous harmonic spectral deviation is the spectral deviation of amplitude (logarithmic scale) components from a global spectral envelope

HarmSpectralSpread: the harmonic spectral spread is the average of the *instantaneous* harmonic spectral spread within a running window. The instantaneous harmonic spectral spread is the amplitude (linear scale) weighted standard deviation of the harmonic peaks of the spectrum, normalized by the instantaneous harmonic spectral centroid

HarmSpectralVariation: the harmonic spectral variation is the mean of the *instantaneous* harmonic spectral variation. The instantaneous harmonic spectral variation is the normalized correlation between the amplitude (linear scale) of the harmonic peaks of two adjacent frames

HFC: the high-frequency Content: taken across a signal spectrum. It characterizes the amount of high-frequency content in the signal (the magnitudes of the spectral bins are added together, after mutliplying each magnitude by the bin "position" (proportional to the frequency))

HpFilter: high-pass filter

Integration: the cumulated sum of values of a vector

Inverse: $1/x$

IQR: the Interquartile range of a vector of values: the difference between the percentile at 75% and the percentile at 25%

Length: the number of items in a vector

Log10: $\log_{10}(x)$

LpFilter: low-pass filter

Max: the maximal value of any vector of values

MaxPos: the index of the maximal value of a vector

Mean: the mean value of a vector

Median: the median value of a vector

MelBands: computes the sums of amplitudes in filtered Mel bands covering the whole spectrum (all bands have the same size in terms of Mel pitches)

Min: the minimal value of a vector

Mfcc0: the Mel-Frequency Cepstral coefficients of a signal, including the coefficient 0 (the energy)

Mfcc: the Mel-Frequency Cepstral coefficients of a signal

Multiplication: multiplication by a constant

Normalize: normalizes a signal

***N*th:** the value of the nth item in a vector

PeakPos: returns the positions of all the peaks of a signal

Percentile: for a vector v of values, of length l , and a parameter $p \in [0, 1]$, sorts the values of v in increasing order, and returns the value at position $p \times l$ (rounded if $p \times l$ is not an integer)

Pitch: the pitch of a signal computed using the spectral multiplication method

PitchBands: computes the sums of amplitudes in filtered pitch bands covering the whole spectrum

Power: for a parameter n , returns x^n

Range: difference between the max and min values of a vector of values

RemoveSilentFrames: takes a list of frames and returns the list without the silent frames (frames with only 0s)

RHF: the ratios of High-Frequencies of a signal: the sum of the squares of the high frequency half of the spectrum divided by the sum of the squares of the low frequency half of the spectrum

RMS: Root Mean Square, the mean energy of the signal

SpectralCentroid: the first moment of the spectrum: the brightness of the sound

SpectralDecrease: measures the decrease of the spectrum energy with perceptive criteria

SpectralFlatness: measures the similarity of the spectrum to a white noise

SpectralKurtosis: smoothness of the spectrum compared to a Gaussian distribution (4th moment)

SpectralRolloff: frequency under which most of the energy is found in the spectrum (rough approximation of the cut frequency between harmonic signal and noise)

SpectralSkewness: the dissymmetry of the sprectrum compared to a Gaussian distribution (3rd moment)

SpectralSpread: the spread of the spectral energy around the Centroid

Split: splits the input signal in constant-size frames (size is specified as an argument); returns the list of frames

SplitOverlap: idem *Split*, except with overlapping; the overlapping factor is a parameter

Sqrt: the squared-root of a value

Square: the square of a value

Sum: the sum of the values in a vector

Triangle: applies a triangle filter to a signal

Variance: the first statistical moment

ZCR: the zero-crossing rate of a signal

Harmonicity: the degree of acoustic periodicity, also called harmonics-to-noise ratio (HNR); see Praat, [74]

LTAS: the long-term average spectrum; see Praat, [74].

3. Experiments

Many experiments were conducted with the EDS system to evaluate the extent that interesting AFs are found for difficult audio classification tasks. Several systematic studies are reported by Zils [46], including the detection of a singing voice in polyphonic recordings, the classification of musical instruments, and the classic problem of music genre classification. Music genre classification has received a lot of attention in music information retrieval research. Although it has been shown that existing large-scale music taxonomies are inconsistent with each other [75], several works have addressed the problem of automatic classification for small-scale genre taxonomies based on the acoustic properties of music signals (see, e.g., [33]). As are most music classification systems, published music genre classifiers are based on statistical aggregation of short-term features of signals. Because of its relative simplicity, four-genre classification has become a textbook case for signal classification research. The problem also provides a typical example in which AFs are useful. Zils [46] reports an improvement due to the use of AFs over known BOF features in four-class genre classification; the four easily distinguishable musical genres are Dance, Hip-Hop, Symphonic Classical, and Pop/Rock (see Table 1). Similarly, Mierswa [50] reports good results, although no comparison was made with other approaches on the same data set.

Table 1: Performance of AFs generated for a four-class music genre problem [46].

		Perf train	Perf test	Feature
Best features	REF	58%	53%	*Variance(Bandwidth(Envelope(FFT(Split(x, 250)), 50), 10))*
	EDS	61%	64%	*Log(Range(SpectralSpread(Hann(Split(Blackman(x), 153)))))*
Best classifiers	REF	79%	80%	*Multilayer perceptron (23 features, 3 layers)*
	EDS	82%	80%	*kNN (25 features, 8 neighbors)*

However, it should be noted that in all cases, the features found by Zils [46] or by Mierswa [50] fall into the category of BOF features. We do not think that music genre classification is an interesting example for deriving more complex features, because current systems (including our own approach) already achieve about 80% performance; the remaining error is mostly due to the difficulty of defining the "ground truth" precisely.

Applications in less studied domains were also performed. Interestingly, initial attempts to apply EDS to chord recognition problems showed reasonable performance, but without improvement over existing approaches [76]. Subsequent comparative studies using an improved version of the basic operator library were performed on *f0* estimation and chord recognition problems [77]. These studies showed that EDS is able to match and occasionally surpass traditional ad hoc approaches which are known to be particularly efficient and difficult to improve. A study concerning urban sounds analysis considered the use of AFs for hierarchical classification [78], and similarly concluded that there are advantages of AFs over standard methods. For movie remastering, Monceaux et al. [79] showed that perceptive descriptors built with AF could be used to automate the tedious task of audio *upmixing*. All of these studies reported substantial improvements in classification using AFs over state-of-the-art audio features.

In the following sections, we describe the results obtained with AFs on several difficult audio classification tasks, and report on the most salient aspects of these studies.

3.1. Artificial Experiments. In this section, we describe two artificial experiments aimed at situating AFs empirically in the space of "known" features. The first problem demonstrates that a single AF can perform better than a hundred known, generic features. The second experiment shows the reverse, that AFs can approximate a known feature.

3.1.1. The Pink Noise Experiment. The problem we consider here, briefly sketched by Pachet and Zils [80], consists of detecting a sinus waveform in a given frequency range (say, 0–1000 Hz), mixed with a powerful colored noise in a separate frequency range (1000–2000 Hz). Since the colored noise is the most predominant characteristic of the signal, we show that generic features are unable to detect the isolated sinus. For instance, examining the spectrum of a 650 Hz sinus mixed with a 1000–2000 Hz colored noise (Figure 4), the peak of the sinus is visible but not predominant; thus it is hard to extract automatically with general spectral features.

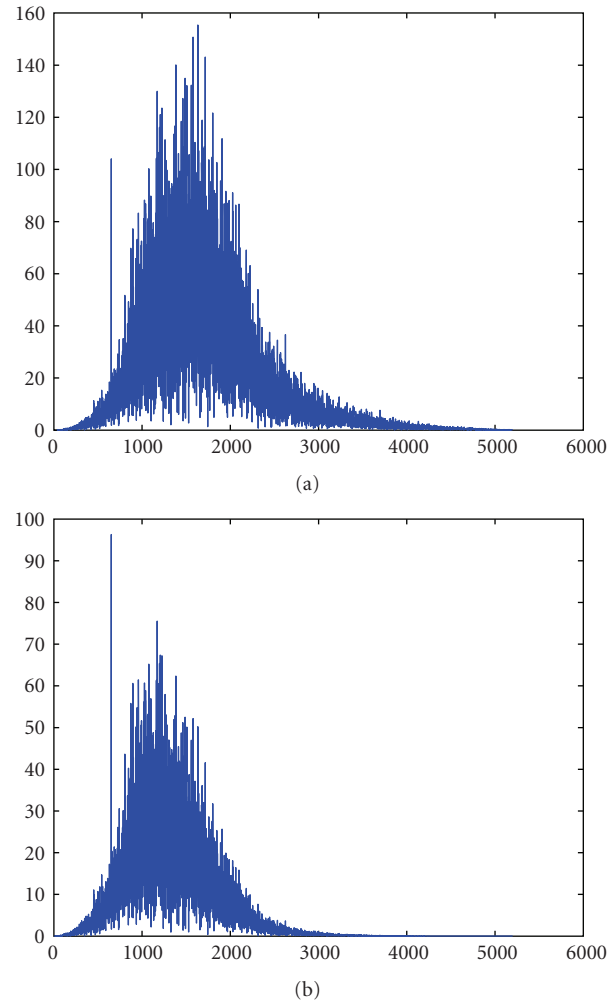

Figure 4: (a) A spectrum of a 650 Hz sinus mixed with 1000–2000 Hz colored noise. (b) A spectrum of a 650 Hz sinus mixed with 1000–2000 Hz colored noise, prefiltered by a 1000 Hz low-pass filter.

Of course, this problem is easy to solve manually with knowledge of how these signals are constructed. Prefiltering can be applied to cut off most of the colored noise frequencies, allowing the sinus to emerge from the spectrum. Figure 4 shows the sinus peak emerging when the signal is low-pass filtered, thus it becomes easier to extract automatically.

Although simple, this problem cannot be adequately solved using generic features. This is demonstrated in the following experiment. We build a database of sinus plus colored

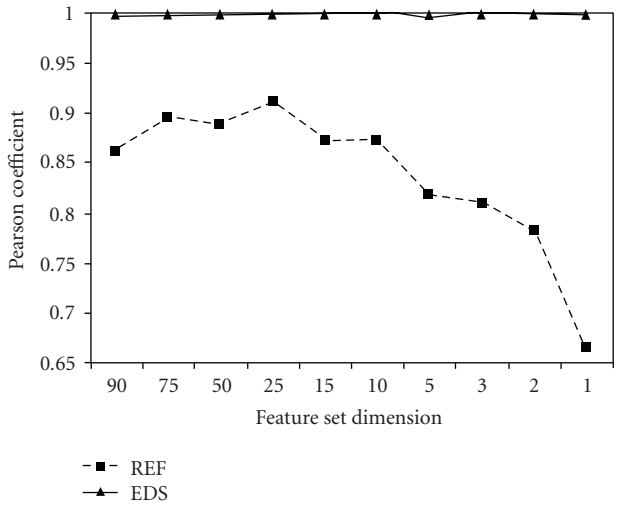

Figure 5: Results of the Pink noise experiment. The plain line is the performance of AFs, and the dotted line is the reference features. One AF is enough to solve the problem perfectly, whereas a feature set of dimension 100 with generic features cannot. The difference is even more drastic for smaller feature sets.

noise with varying values for the sinus (from 0 to 1000 Hz). Each signal is labeled with the corresponding frequency of the sinus. We then train two regression classifiers for this task. The first one (called *REF*) is trained with a "generic feature set" and contains features taken from the Mpeg-7 audio standards; it has dimensions varying from 1 to 100 (details in [61]). The second one (called *EDS*) is trained with AFs built specifically for this problem. EDS explored about 40 000 AFs to find an almost optimal AF for this problem, using the most general pattern "!_*a*". The result is illustrated in Figure 5. This shows clearly that one analytical feature (dimension 1) can solve the problem almost perfectly, whereas a full set of generic features (dimension 100) cannot. The best analytical feature found by EDS is the following:

$$\text{(A)}\quad \text{MaxPos}(\text{FFT}(\text{Integration}(\text{BpFilter}(\text{Hamming}(x), 10, 500)))).$$

This AF (A) is easy to interpret, and does almost exactly what should be done in this case, although probably not in the simplest way. Its fitness (Pearson coefficient) is .99. The most likely human feature, based on knowledge of the database creation process, would be something like

$$\text{(H)}\quad \text{MaxPos}(\text{FFT}(\text{LpFilter}(x, 1000))).$$

This theoretically perfect feature is also good (fitness = 0.87), but it has a slightly lower fitness than (A). This shows incidentally that the best features are in practice not always the most justified, mathematically.

3.1.2. Approximating a Known Feature. The second experiment assesses how well AFs can approximate a known feature, which is not expressible directly, using the operator library. To this end, we chose an audio feature called "spectral compactness", with an implementation available from the *jAudio* feature extraction library [81]. This feature is described as follows (p. 603, *ibid.*): "*Compactness is closely related to Spectral Smoothness as defined by McAdams [82]. The difference is that instead of summing over partials, compactness sums over frequency bins of an FFT. This provides an indication of the noisiness of the signal.*"

We do not know the details of the implementation of this feature, nor the feature's precise definition, but they are not needed here. The aim of this experiment is to see how "reachable" this feature is, using the set of generic features and using AFs. Because spectral features are known to be correlated, and because *MFCC* coefficients are known to well represent spectral information in general, we conducted several experiments to show that spectral compactness is not easily reached from other known sets of generic features. We used a database of 2500 percussion sounds described by Roy et al. [83]. We used patterns "!_*a*", "!_f(!_Vf(!_f : a(!_t : $a(x)$))))", and the usual BOF pattern "?_a(?_Va(Hanning(Split(x, 512, 0.5))))". The results are illustrated in Figure 6.

We can see that the performance gain of analytical features increases as the size of the feature set decreases, with a spectacular result from single features (84% versus 43%). Additionally, Figure 6 shows that AFs outperform all the specific feature sets considered here: general spectral features, MFCC, and the entire set of general features! EDS produced the following analytical feature, an instantiation of the pattern "!_f(!_Vf(!_f : a(!_t : $a(x)$))))":

$$\text{Abs}_{\_f}(\text{Sum}_{\_f}(\text{Integration}_{\_Vf}(\text{PeakPos}_{\_Vf}(\text{FFT}_{\_f:a}(\text{HpFilter}_{\_t:a}(\text{Derivation}_{\_t:a}(x_{\_t:a}), 3969)))))).$$

This AF yields a correlation coefficient of 0.79. This is an interesting result, since it contains operators that explicitly appear in the definition of spectral compactness (see above). However, EDS produced an even better analytical feature,

$$\text{(EDS)}\quad \text{Abs}(\text{Centroid}(\text{Hanning}(\text{Variance}(\text{Derivation}(\text{Split}(x, 882)))))),$$

whose correlation with spectral compactness is over 0.84. As a comparison, the best reference feature identified is

$$\text{(BestRef)}\quad \text{HarmonicSpectralDeviation}(\text{Hanning}(x)).$$

3.2. Perceived Intensity. One of the first problems addressed by EDS was the *perceived energy* problem (also called the *subjective intensity* problem). Perceived energy relates to the relative energy levels that listeners intuitively attribute to different songs played at the same volume level. For example, a punchy punk-rock song with loud saturated guitars and screaming voices sounds relatively "more energetic" than, say, an acoustic guitar ballad with a soft voice. To model this subjective impression, users were asked to label musical extracts of various genres with the "energy felt", independent of the listening volume [84]. We collected more than 2600

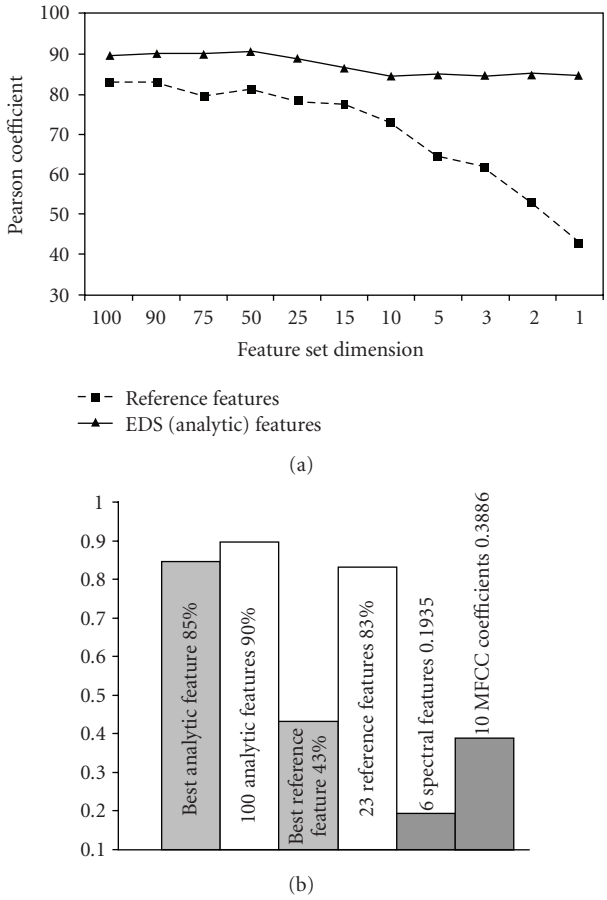

Figure 6: (a) A performance comparison of analytical versus reference features, for various feature set dimensions. AFs perform better, and the improvement is more important for feature sets of small dimension. (b) A comparison of various regression classifiers to approximate the "spectral compactness" feature. The best AF yields a performance of 85%. It outperforms (1) the best reference feature (43%); (2) a feature set with 6 general spectral features (19%); (3) the 10 first MFCC (38%); and (4) the whole set of reference features of dimension 100 (83%). A set with the 100 best AFs further improves the correlation up to 90%.

answers, about 12-13 answers per title. A consensus for the perceived intensity value was found for each title, using a process that eliminated extreme values, assigned a numeric value to each energy level (normalized between 0 for "Low" and 1 for "Very High"), and computed the mean value from this normalized data. To evaluate the relevance of this intensity value, we computed the standard deviation of the results for all of the listeners. For 200 titles (98%), the standard deviation was less than the distance between two successive categories (0.33), so the mean value was assumed to be a correct evaluation of the intensity. The remaining 2% of titles were removed from the database, as their intensity was considered to be too indefinite to be evaluated automatically. The resulting intensity values were used as ground truth to train EDS and to perform feature selection. We proceeded the same way on a second series of perceptive tests, building a 200-title database to test the AFs and classifiers.

A statistical analysis of these tests showed that musical energy is a consensual concept, that is, most users perceive the same energy for the same songs, with a 10% statistical variance. We then built two labeled databases of 200 signals, each of length 5 s and at 11025 Hz; one database was used for training and the other was used for testing. The results are shown in Table 2.

We then proceeded to compare three different feature sets. The first set was obtained from 30 low-level Mpeg-7 descriptors, extracted from Herrera et al. [85]. The second set was composed of manually derived AFs based on our "intuition" about perceived intensity, in which the intensity is related to the tempo, or more generally, to energy variations of the signal. Several new features were built out of intuitive concepts. Using the template of Scheirer's tempo extractor [52], we manually designed AFs describing the signal energy variations. We then correlated these features with the perceptive tests. The strongest correlated AF was found (by hand) to be

$$\text{(hand)}\quad \text{Log}(\text{Variance}(\text{Derivation}(\text{Energy}(x)))) \tag{}$$

with a correlation of 0.57 with the perceptive tests. This AF relates to the amplitude and frequency of the signal's raw energy. As a comparison, the tempo that was extracted automatically with Scheirer's method yielded a correlation of 0.55.

The third set of AFs was derived automatically by EDS using the AF BOF pattern (see Section 2.5.1). After evolving over 22 populations, 100% of the AFs created by EDS were found to perform better than the best Mpeg-7 features on the training database. They also yielded 91% accuracy on the test database.

The best AF found by EDS was:

$$\text{(BestEDS)}\quad \text{Square}\,(\text{Log10}\,(\text{Mean}\,(\text{Min}\,(\text{FFT}\,(\text{Split}\,(x, 4009)))))) \tag{}$$

with a fitness (Pearson correlation coefficient) of 0.744 on *train*, and 0.812 on *test*.

It is interesting to examine how EDS yielded its best AFs. At first, EDS found AFs close to the Mpeg7 low-level descriptors, but it eventually improved upon them with various types of preprocessing. For instance, EDS found:

$$\text{(Mpeg7)}\quad \text{Mean}(\text{SpectralSkewness}(\text{Split}(x, 4410))), \tag{}$$

which yielded a fitness (Pearson correlation) of 0.60. Then, EDS found AFs which were close to the empirically discovered Scheirer-inspired feature, although they had been improved with additional operators. The AF with the strongest correlation was:

$$\text{(Empirical)}\quad \text{Mean}(\text{Log}(\text{Variance}(\text{Split}(\text{Derivation}\,(\text{Square}(x)),1)))), \tag{}$$

with a correlation coefficient of 0.64. Finally, EDS found really novel features, such as:

$$\text{(newEds)} \quad \text{Sqrt(Min(Sum(FFT(Split}(x, 1))))),$$

which yielded a fitness of 0.69, as well as the best feature shown above (BestEDS).

Interestingly, these findings (notably feature *Mpeg7*) were confirmed by a larger-scale manual study by Sandvold and Herrera [86], using a traditional feature selection approach (by definition,they could not find the other features found by EDS). To conclude our study, we evaluated the performances of classifiers built with the optimal combination of the resulting AFs. After running a final selection, EDS retained 24 features to use in the final energy classifier. The best method found was a kNN that yielded a score (accuracy) of 78.2% on *train*, and 85.1% on *test*. It combined both Mpeg7 and the best AFs found by EDS. The model error of this best classifier was 11.3%. Considering the 10% variance on the perceptive tests, we believe that EDS found a virtually optimal model for this problem. The performances of these classifiers are compared in Table 2.

The study of perceived energy was an interesting one, since it demonstrated that EDS can outperform any available general-purpose feature set, and because the results were confirmed by a later manual study.

3.3. Classification of Pandeiro Sounds. Analytical audio features can naturally be used to classify instrument sounds. Some small-scale studies were reported by Zils [46] for instrument databases, again demonstrating improvements using AFs compared with general-purpose features. However, the limitation to these kinds of study is the difficulty of building reference ground truth databases [87]. As a consequence, any deficiencies in these classifiers cannot be attributed with certainty to their features. Some studies have addressed the sub-problem of drum sound classification, for which the *ground truth* issue is less salient; however, since the use of traditional feature sets results in excellent performance, this suggests that the problem is not a very difficult one [38].

Again, the difficulty becomes apparent when we consider lower-level abstractions. For instance, tabla transcription [88] appears to be more difficult than general drum sound identification. An interesting study was performed on the focused but difficult problem of pandeiro sound classification [83, 89].

This study was motivated by the design of interactive music systems that are controlled in real time by percussive instruments. The pandeiro is a Brazilian percussion instrument that is able to produce six categories of sounds (see Figure 7). The problem addressed in our study was to classify pandeiro sounds in each of these six categories, using only the attack portion of the signal. Because the playing modes of the pandeiro, and therefore the six categories of sound, are well known, the ground truth problem becomes irrelevant and we can focus on the feature generation.

Figure 7: The gestures that produce the six basic Pandeiro sounds (tung, ting, tchi, tr, PA, pa).

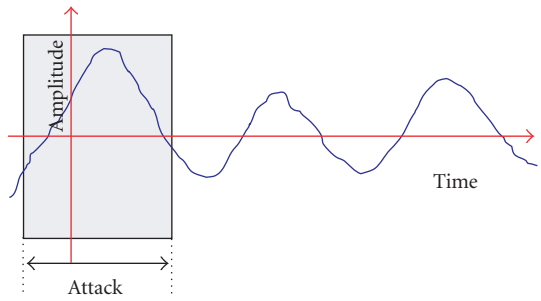

Figure 8: A full sound and its attack portion. The attack is defined by an amplitude threshold: when the amplitude goes above the threshold, the attack begins, and when it eventually goes under, the attack ends. In practice, we consider 128 samples after the beginning of the attack, that is, about 3 milliseconds.

The classification of complete sounds was reported by Roy et al. [83] and was shown to be a relatively easy task. However, the application targeted is an interactive music system. Therefore, we need to classify pandeiro sound in real-time, or at least within a few milliseconds, so no interruption can be perceived during the interaction. As a consequence, we cannot use complete sounds to classify a stroke, as they last approximately 150 milliseconds. Instead, we try to classify from the first 3 milliseconds (or 128 samples, at 44 100 Hz) of each sound.

For this study, we recorded a database of 2448 complete, isolated Pandeiro sounds that were produced by the same instrument during different sessions. The 2448 samples were equally distributed (408 per class).

We extracted the attack portion of the sounds so that the attack lies in the middle of the considered sample (see Figure 8, [89]).

To assess the efficiency of AFs, we compared them to results obtained with a reference feature set (a complete list of the reference features is given by [83]). This set includes features commonly used in audio signal classification tasks, notably the Mpeg-7 audio list, and *Chroma*, which is often used for music analysis [90]. We systematically evaluated the performance of two classifiers: one built with the reference set, the other built with the AFs found by EDS using the set of basic operators defined in Section 2.6.

Table 2: The performances for the perceived intensity classifiers using various feature sets.

Method	Correlation	Model error
Random (mean value for all titles)	0.18	21%
Best MPEG7 feature [(SpectralSkewness (x)]	0.56	16.9% ± 1.5%
MPEG7 feature combination 21 features selected	0.87	12.1% ± 1.9%
Best EDS feature [Sqrt(Min(Sum(FFT(Split(x, 1 sec)]	0.68	14.5% ± 1.8%
EDS + MPEG7 combination 18 features selected	0.89	11.3% ± 1.8%

Each evaluation experiment was divided into two parts. In the first part, classifiers were trained with the training database and tested with the test database. The databases were divided so that two-thirds were used for training and one-third for testing. The samples were chosen at random to avoid artifacts, such as those caused by the evolution of the membrane during the recording session, or by small variations in the player gestures. In the second part of the experiment, classifiers were trained and tested on the test database only, using 10-fold cross-validation in which each classifier was trained on 9/10 of the test database, and its precision was evaluated on the remaining 1/10. This operation was repeated 10 times using random draws. The objective of this double experiment was to demonstrate that the advantages of using AFs do not depend on the conditions of the experiments. The use of cross-validation using only the test database is further supported since EDS uses the training database to evaluate the analytical features; reusing it to train the classifiers could introduce biases.

Finally, we also evaluated the signal itself (128 samples) as a feature (this is possible here only because these attack signals are very short). This was done to confirm that the raw signal is not a good feature, and also to provide a point of comparison for the other experiments.

We used an SVM with a default polynomial kernel. EDS was used fully automated to create and select AFs. The patterns used included the following simple patterns (we deliberately avoided the *Split* operator, since the samples are very short):

(i) "!_a(!_t : $a(x)$)",

(ii) "!_a(!_$V$$a$(!_$t$: $a(x)$))".

We ran the genetic search until the improvements in the feature fitness converged. EDS evaluated about 50,000 features. Because attacks are small signals, only 17 reference features could be computed and evaluated, with a total dimension of the feature set equal to 90. We therefore selected 90 AFs out of the AFs found by EDS on the attack. We present results obtained for various sizes of the feature sets, ranging from 1 to 90. This is an important aspect for real time applications. As seen in Figure 9, EDS finds not only better features, but also feature sets with a smaller dimension.

Again, AFs found by EDS improve the classification performance. AFs are superior to general features, in particular for small feature sets; three AFs perform as well as the 50 best general features.

For the experiments reported in Table 3, the feature selection algorithm used is based on syntactical properties of analytical features. Other feature selection schemes, such as Information Gain Ratio (IGR), yield to the same results. However, for feature sets of small dimensions, the performance of IGR decreases substantially (this result is discussed in [91]).

The performance gain due to analytical features for small feature sets is advantageous, especially for real time applications. Table 3 shows that three AF features yield a better precision than 50 reference features. These features are:

Abs (Log10 (Percentile (Square (BpFilter(x,764,3087)), 64)))
Centroid (MelBands (Derivation (HpFilter
(Power (Normalize(x), 3), 100)), 6))
Abs (Sum (Arcsin (Mfcc (Hann(HpFilter(x,19845)), 20)))).

3.4. Classification of Animal Sounds. A pioneering large-scale study was conducted concerning the automatic classification of companion dog barks, using machine learning algorithms [92]. The goal of this study was to determine whether specific but speaker-independent (so-to-speak) acoustic features of barks were encoded in different "barking situations". Previous studies have showed that humans have the ability to categorize various barks that they hear and to associate them with the appropriate emotional content [93]. They also showed that humans with different dog experience levels show similar trends in categorization of the possible inner state of the given barking dog. Another study showed that human perception of the motivational state of dogs is influenced by acoustic parameters in the barks [94]. In contrast, humans show only modest accuracy in discriminating between individual dogs by hearing their barks [95].

The database consists of barks of the *Mudi* breed individuals recorded from 14 individuals. The total sample size of barks is 6646, categorized in 6 different *situations*: "stranger", "fight", "walk", "alone, "ball", and "play". AFs were investigated as to their performance for training and testing six-category classifiers on this database. An interesting aspect of this study is that we used an external library [74] to complement the basic operator library. The recognition of dog barks requires the analysis of specific signal features, similar to the features used routinely in human speech recognition. Spectral operators used in speech recognition, such as MFCCs, are included in the basic EDS library, but others are not, such as formants, which are specific to

Table 3: Results obtained for the attack problem. *Reference* refers to a reference feature set of MPEG7 audio features. *AF* refers to analytical features generated with EDS. Feature sets of smaller dimensions are subsets of the feature sets of dimension 90, constructed using our feature selection algorithm based on AF syntactical structure. *Signal* refers to the signal used as a feature set, for comparison.

Feature	Feature set dimension									
	90	75	50	25	15	10	5	3	2	1
Reference	[33, 91]	[33, 69]	[31, 37]	[15, 51]	[31, 43]	[20, 91]	[1]	[7, 69]	[12, 16]	[56]
AFs	94,5	94	93,3	91,4	91,4	89	89,5	88	80,1	69,2
Signal	77.7	76.9	73.3	64.1	64.2	60	59.2	58.1	57.5	44

(i) *SpectralRolloff (derivation(x))*
(ii) *SpectralFlatness (square(x))*
(iii) *Sqrt(RHF(derivation(Abs(derivation(x)))))*
(iv) *Rms(SpectralSkewness(split(x, 512)))*
(v) *Abs(max(SpectralFlatness(split(x, 256))))*
(vi) *Mean (formant (1, x))*
(vii) *Deviation (harmonicity (x))*

Figure 9: The best AFs found for dog bark classification, using operators borrowed from speech recognition. Note their relative syntactic simplicity, and the use of different window sizes.

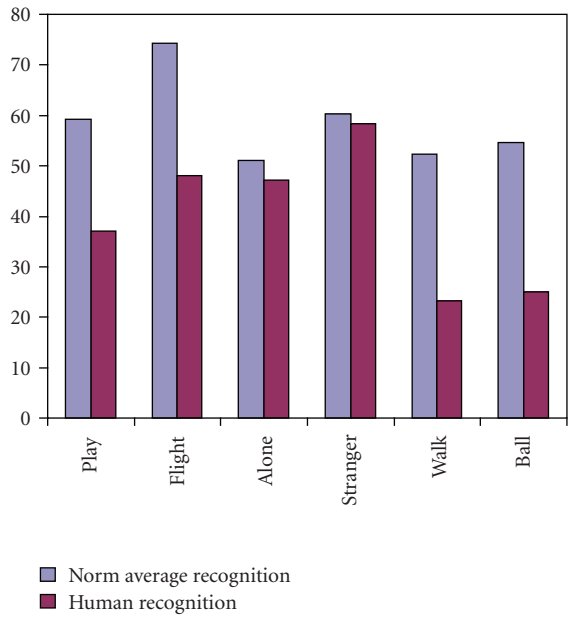

Figure 10: An AF based classifier outperform humans at dog-independent bark recognition tasks.

speech. We used the BOF patterns "!_a(!_t : a(Split(x)))" and "!_a(!_t : a(x))". With this extension of the basic operator library, the best AFs found by EDS were the (admittedly relatively simple) following Figure 7.

The most interesting conclusion of this study was that the automatic classifier could reliably discriminate between individual dogs while humans could not [95], as shown in Figure 10. This was the main difference between the performances of the machines and humans.

This supports the concept that there are individual differences in dog barks, but humans are not able to recognize them easily. Incidentally, this study (discussed in detail in Molnár et al. [92]) provides a new perspective for understanding animal communication. There may be "hidden" acoustic features in animal sounds that could eventually be identified. Similar issues have been raised regularly by behavioral scientists of all sorts, for example in the study of bird songs [96].

4. Discussion

The traditional view of feature generation, inspired by supervised concept induction, is that the performance of classifiers can be boosted if the "natural" features are not expressive enough. In this context, feature generation is an additional method for improving classifier performance, along with feature selection, kernel parameter tuning, and boosting. The situation is different when the items to classify are raw signals, as no "natural" features are known *a priori*. However, there is a huge body of technical knowledge about particular signals types (e.g., audio) and particular operators. For the most part, the link between signals and features for a particular classification task is still performed heuristically and manually in the signal processing research community. Here we introduce *analytical features*, in association with a feature generation scheme that produces feature sets adequate for arbitrary supervised audio classification problems. For efficiency, the generation of AFs exploits explicit knowledge representation constructs, notably heuristics, rewriting rules, and function patterns, all based on the AF dimensional typing system. We have shown that AFs improve classifier performance on concrete and difficult classification problems. More importantly, they fill the gap between the body of scientific knowledge about audio signals, and the feature identification problem inherent to all signal classification tasks. In this respect, AFs represent the knowledge-based counterpart of feature generation studies.

However, the use of analytical features raises several open issues, related to the very nature of signals.

(i) The *operators* considered so far were chosen by studying existing and efficient features designed manually. However, it cannot be assumed that these operators are the best ones. More cognitively plausible operators could substantially improve the performance of generated features. One possibility is to design operators by taking inspiration from the model of the cochlea, as suggested by Pressnitzer

and Gnansia [18], as a way to evolve more biologically grounded features. Another possibility is to manipulate lower-level, sample-based operators, for instance, by using data flow models of signal processing functions [97]. Finally, for applications dealing with psychologically high-level classes like musical genre, operators explicitly dealing with *long-term* properties of the signal that are not expressible through statistical moments should be investigated in more detail. Finally, the intrinsic limitation of AFs compared with more general *programs* can be overcome by allowing the system to create new operators on the fly. For instance, a classifier achieving good performance on different problems could be abstracted into a new operator, and then reused on other problems.

(ii) We do not know much about the *topology* of the AF space. How can we define similarity measures for AFs? Are there clusters or hubs in this space? How can we infer feature properties (fitness) from their syntactic representation? Preliminary work has started to address these issues [69], but much remains to be done before the space can be better understood mathematically, and before we can possibly predict the usefulness of generated features.

(iii) To date, the set of *heuristics* (and rewriting rules) is hand-coded. Ideally, this knowledge about signal processing operators and functions would be gained automatically by the system, similar to AM's Mathematical inventions. To do so, the system needs to be able to design experiments, assess their interestingness, and infer general properties from them. This would allow the system to constantly update its body of knowledge autonomously. Such an autotelic automated researcher is currently being investigated, bootstrapping off the problems and results found from the study of EDS.

(iv) As predicted by Lenat's observations [54], it appears that EDS, like other feature generation systems, eventually reaches the *fringe* of the AF space, although it certainly does not explore all of it. This phenomenon (most AFs are similar after a certain point) may be related to the intrinsic brittleness of systems built upon a finite set of basic operators. A way to increase the potential performance of feature generation is to open the system so that it exploits information or knowledge produced "outside", for example, knowledge available online, in an *e-science* perspective. Technically, this requires a different design of the operator library and of the type system onto which everything is grafted. The use of *tags* as substitutes for operator *types* is an interesting avenue of research. Tags have already proven to be efficient for indexing knowledge implicitly expressed in online scientific papers, most of which are not fully exploited (or even read).

Whereas supervised concept induction is primarily used as a tool for optimizing the performance of problem solvers (starting with Samuel's checkers), the AF construct allows the transition of symbolic problems to signal-based problems. This transition offers a practical way to investigate two fundamental questions of human perception: (1) to what extent is human perceptual categorization of items based on objective features of those items, and (2) in those cases, can we identify these features explicitly? Analytical features do not provide a definitive answer to these two questions in general. However, by performing a systematic search in large function spaces, this approach lessens the impact of arbitrarily chosen feature sets on classifier performance. As such, they provide a promising avenue to address these issues, as has been illustrated in the audio domain, but also, we think, in all other domains of human categorization.

References

[1] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 211–229, 1959.

[2] M. Buro, "From simple features to sophisticated evaluation functions," in *Proceedings of the 1st International Conference on Computers and Games (CG '98)*, vol. 1558 of *Lecture Notes in Computer Science*, pp. 126–145, Springer, London, UK, 1998.

[3] T. E. Fawcett and P. E. Utgoff, "Automatic feature generation for problem solving systems," in *Proceedings of the 9th International Workshop on Machine Learning (ML '92)*, D. Sleeman and P. Edwards, Eds., pp. 144–153, Morgan-Kaufmann, Aberdeen, Scotland, July 1992.

[4] R. S. Michalski, "A theory and methodology of inductive learning," in *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonnell, and T. M. Mitchell, Eds., chapter 4, pp. 83–134, Morgan Kaufmann, San Mateo, Calif, USA, 1983.

[5] K. A. De Jong, W. M. Spears, and D. F. Gordon, "Using genetic algorithms for concept learning," *Machine Learning*, vol. 13, no. 2-3, pp. 161–188, 1993.

[6] T. E. Fawcett, "Knowledge-based feature discovery for evaluation functions," *Computational Intelligence*, vol. 12, no. 1, pp. 42–64, 1996.

[7] G. Bagallo and D. Haussler, "Boolean feature discovery in empirical learning," *Machine Learning*, vol. 5, no. 1, pp. 71–99, 1990.

[8] S. Salzberg, "Improving classification methods via feature selection," Tech. Rep. JHU-92/12, Department of Computer Science, Johns Hopkins University, Baltimore, Md, USA, 1992.

[9] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *The Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.

[10] Z. Zhao and H. Liu, "Searching for interacting features," in *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI '07)*, pp. 1156–1161, Hyderabad, India, January 2007.

[11] D. Yang, L. A. Rendell, and G. Blix, "A scheme for feature construction and a comparison of empirical methods," in *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI '91)*, pp. 699–704, Sydney, Australia, August 1991.

[12] S. Markovitch and D. Rosenstein, "Feature generation using general constructor functions," *Machine Learning*, vol. 49, no. 1, pp. 59–98, 2002.

[13] E. Gabrilovich and S. Markovitch, "Feature generation for text categorization using world knowledge," in *Proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI '05)*, pp. 1048–1053, Edinburgh, UK, August 2005.

[14] K. Krawiec, "Genetic programming-based construction of features for machine learning and knowledge discovery tasks," *Genetic Programming and Evolvable Machines*, vol. 3, no. 4, pp. 329–343, 2002.

[15] A. Asuncion and D. J. Newman, "UC Irvine Machine Learning Repository," University of California, School of Information and Computer Science, Irvine, Calif, USA, 2007, http://archive.ics.uci.edu/ml.

[16] R. E. Bellman, *Dynamic Programming*, Princeton University Press, Princeton, NJ, USA, 1957.

[17] J.-C. Risset, "Timbre analysis by synthesis: representations, imitations, and variants for musical composition," in *Representation of Musical Signals*, chapter 1, pp. 7–43, MIT Press, Cambridge, Mass, USA, 1991.

[18] D. Pressnitzer and D. Gnansia, "Real-time auditory models," in *Proceedings of the International Computer Music Conference (ICMC '05)*, pp. 295–298, Barcelona, Spain, September 2005.

[19] S. Theodoridis and K. Koutroumbas, *Pattern Recognition*, Academic Press, San Diego, Calif, USA, 3rd edition, 2006.

[20] O. Ritthoff, R. Klinkenberg, S. Fischer, and I. Mierswa, "A hybrid approach to feature selection and generation using an evolutionary algorithm," in *Proceedings of the UK Workshop on Computational Intelligence (UKCI '02)*, J. Bullinaria, Ed., pp. 147–154, Birmingham, UK, September 2002.

[21] S. H. Lim, L.-L. Wang, and G. DeJong, "Explanation-based feature construction," in *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI '07)*, pp. 931–936, Hyderabad, India, January 2007.

[22] H. Guo, L. B. Jack, and A. K. Nandi, "Feature generation using genetic programming with application to fault classification," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 35, no. 1, pp. 89–99, 2005.

[23] H. Guo and A. K. Nandi, "Breast cancer diagnosis using genetic programming generated feature," *Pattern Recognition*, vol. 39, no. 5, pp. 980–987, 2006.

[24] R. I. Dogan, L. Getoor, and W. J. Wilbur, "A feature generation algorithm for sequences with application to splice-site prediction," in *Proceedings of the 10th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD '06)*, vol. 4213 of *Lecture Notes in Computer Science*, pp. 553–560, Berlin, Germany, September 2006.

[25] T. Solorio, O. Fuentes, N. Ward, and Y. A. Bayyari, "Prosodic feature generation for back-channel prediction," in *Proceedings of the 9th International Conference on Spoken Language Processing (INTERSPEECH-ICSLP '06)*, vol. 5, pp. 2398–2401, Pittsburgh, Pa, USA, September 2006.

[26] D. Eads, K. Glocer, S. Perkins, and J. Theiler, "Grammar-guided feature extraction for time series classification," in *Proceedings of the 9th Annual Conference on Neural Information Processing Systems (NIPS '05)*, Vancouver, Canada, December 2005.

[27] D. R. Eads, D. Hill, S. Davis, et al., "Genetic algorithms and support vector machines for time series classification," in *Applications and Science of Neural Networks, Fuzzy Systems, and Evolutionary Computation V*, vol. 4787 of *Proceedings of SPIE*, pp. 74–85, Seattle, Wash, USA, July 2002.

[28] K. Krawiec and B. Bhanu, "Visual learning by coevolutionary feature synthesis," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 35, no. 3, pp. 409–425, 2005.

[29] G. Peeters, "A large set of audio features for sound description (similarity and classification) in the CUIDADO project," 2003, http://www.ircam.fr/anasyn/peeters/ARTICLES/Peeters_2003_cuidadoaudiofeatures.pdf.

[30] K. West and S. Cox, "Features and classifiers for the automatic classification of musical audio signals," in *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR '04)*, pp. 531–536, Barcelona, Spain, October 2004.

[31] J.-J. Aucouturier and F. Pachet, "Improving timbre similarity: how high's the sky?" *Journal of Negative Results in Speech and Audio Sciences*, vol. 1, no. 11, pp. 1–11, 2004.

[32] J.-J. Aucouturier, *Ten experiments on the modeling of polyphonic timbre*, Ph.D. thesis, University of Paris 6, Paris, France, June 2006.

[33] G. Tzanetakis and P. Cook, "Musical genre classification of audio signals," *IEEE Transactions on Speech and Audio Processing*, vol. 10, no. 5, pp. 293–302, 2002.

[34] C. McKay and I. Fujinaga, "Automatic genre classification using large high-level musical feature sets," in *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR '04)*, pp. 525–530, Barcelona, Spain, October 2004.

[35] A. Eronen and A. Klapuri, "Musical instrument recognition using cepstral coefficients and temporal features," in *Proceedings of the IEEE Interntional Conference on Acoustics, Speech, and Signal Processing (ICASSP '00)*, vol. 2, pp. 753–756, Istanbul, Turkey, June 2000.

[36] G. Peeters, "Automatic classification of large musical instrument databases using hierarchical classifiers with inertia ratio maximization," in *Proceedings of the 115th Convention of the Audio Engineering Society (AES '03)*, New York, NY, USA, October 2003.

[37] A. R. Tindale, A. Kapur, G. Tzanetakis, and I. Fujinaga, "Retrieval of percussion gestures using timbre classification techniques," in *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR '04)*, pp. 541–545, Barcelona, Spain, October 2004.

[38] P. Herrera, A. Yeterian, and F. Gouyon, "Automatic classification of drum sounds: a comparison of feature selection methods and classification techniques," in *Proceedings of the 2nd International Conference on Music and Artificial Intelligence (ICMAI '02)*, pp. 69–80, Springer, Edinburgh, UK, September 2002.

[39] P. Cano, E. Batlle, T. Kalker, and J. Haitsma, "A review of audio fingerprinting," *Journal of VLSI Signal Processing*, vol. 41, no. 3, pp. 271–284, 2005.

[40] P. Hanna, N. Louis, M. Desainte-Catherine, and J. Benois-Pineau, "Audio features for noisy sound segmentation," in *Proceedings of the 5th International Symposium on Music Information Retrieval (ISMIR '04)*, pp. 120–123, Barcelona, Spain, October 2004.

[41] T. L. Nwe and Y. Wang, "Automatic detection of vocal segments in popular songs," in *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR '04)*, pp. 138–145, Barcelona, Spain, October 2004.

[42] L. Lu, D. Liu, and H.-J. Zhang, "Automatic mood detection and tracking of music audio signals," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 14, no. 1, pp. 5–18, 2006.

[43] E. Scheirer and M. Slaney, "Construction and evaluation of a robust multifeature speech/music discriminator," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '97)*, vol. 2, pp. 1331–1334, Munich, Germany, April 1997.

[44] J.-J. Aucouturier and F. Pachet, "Tools and architecture for the evaluation of similarity measures: case study of timbre similarity," in *Proceedings of the International Symposium on Music Information Retrieval (ISMIR '04)*, pp. 198–203, Barcelona, Spain, October 2004.

[45] F. Pachet and A. Zils, "Evolving automatically high-level music descriptors from acoustic signals," in *Proceedings of the International Symposium on Computer Music Modeling and*

Retrieval (CMMR '03), vol. 2771 of *Lecture Notes in Computer Science*, pp. 42–53, Montpellier, France, May 2004.

[46] A. Zils, *Extraction de descripteurs musicaux: une approche évolutionniste*, Ph.D. thesis, University of Paris 6, Paris, France, September 2004, http://aymeric.zils.free.fr/papiers/these04.pdf.

[47] F. Mörchen, A. Ultsch, M. Thies, and I. Löhken, "Modeling timbre distance with temporal statistics from polyphonic music," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 14, no. 1, pp. 81–90, 2006.

[48] F. Mörchen, I. Mierswa, and A. Ultsch, "Understandable models of music collections based on exhaustive feature generation with temporal statistics," in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '06)*, pp. 882–891, Philadelphia, Pa, USA, August 2006.

[49] I. Mierswa and K. Morik, "Automatic feature extraction for classifying audio data," *Machine Learning*, vol. 58, no. 2-3, pp. 127–149, 2005.

[50] I. Mierswa, "Automatic feature extraction from large time series," in *Classification—The Ubiquitous Challenge: Proceedings of the 28th Annual Conference of the Gesellschaft für Klassifikation e.V.*, C. Weihs and W. Gaul, Eds., pp. 600–607, Springer, Berlin, Germanu, 2004.

[51] B. Schuller, S. Reiter, and G. Rigoll, "Evolutionary feature generation in speech emotion recognition," in *Proceedings of IEEE International Conference on Multimedia and Expo (ICME '06)*, pp. 5–8, Toronto, Canada, July 2006.

[52] E. D. Scheirer, "Tempo and beat analysis of acoustic musical signals," *Journal of the Acoustical Society of America*, vol. 103, no. 1, pp. 588–601, 1998.

[53] D. B. Lenat, *AM: an artificial intelligence approach to discovery in mathematics as heuristic search*, Ph.D. thesis, Stanford University, Stanford, Calif, USA, 1976.

[54] D. B. Lenat and E. A. Feigenbaum, "On the thresholds of knowledge," *Artificial Intelligence*, vol. 47, no. 1–3, pp. 185–250, 1991.

[55] D. B. Lenat, "When will machines learn?" *Machine Learning*, vol. 4, no. 3-4, pp. 255–257, 1989.

[56] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, Mass, USA, 1992.

[57] D. E. Goldberg, *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison-Wesley, Reading, Mass, USA, 1989.

[58] K. W. Haase Jr., *Invention and exploration in discovery*, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, Mass, USA, 1990.

[59] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*, Cambridge University Press, Cambridge, UK, 2000.

[60] Q. Sun and G. DeJong, "Feature kernel functions: improving SVMs using high-level knowledge," in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '05)*, vol. 2, pp. 177–183, San Diego, Calif, USA, June 2005.

[61] F. Pachet and P. Roy, "Exploring billions of audio features," in *Proceedings of the International Workshop on Content-Based Multimedia Indexing (CBMI '07)*, pp. 227–235, Bordeaux, France, June 2007.

[62] D. J. Montana, "Strongly typed genetic programming," *Evolutionary Computation*, vol. 3, no. 2, pp. 199–230, 1995.

[63] B. Pierce, *Types and Programming Languages*, chapter 22, MIT Press, Cambridge, Mass, USA, 2002.

[64] R. A. Fisher, "The use of multiple measurements in taxonometric problems," *Annals of Eugenics*, vol. 7, pp. 179–186, 1936.

[65] R. Kohavi and G. H. John, "The wrapper approach," in *Feature Selection for Knowledge Discovery and Data Mining*, H. Liu and H. Motoda, Eds., pp. 33–50, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998.

[66] C. J. van Rijsbergen, *Information Retrieval*, Butterworths, London, UK, 1979.

[67] A. L. Edwards, "The correlation coefficient," in *An Introduction to Linear Regression and Correlation*, chapter 4, pp. 33–46, W. H. Freeman, San Francisco, Calif, USA, 1976.

[68] J. R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann, San Francisco, Calif, USA, 1993.

[69] G. Barbieri, M. Degli Esposti, F. Pachet, and P. Roy, "Complexity of the fitness landscape for analytical features," Tech. Rep. CSL 09-1, Sony Computer Science Laboratory-Paris, Paris, France, January 2009.

[70] K. Zhang and D. Shasha, "Simple fast algorithms for the editing distance between trees and related problems," *SIAM Journal on Computing*, vol. 18, no. 6, pp. 1245–1262, 1989.

[71] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Prentice Hall, Upper Saddle River, NJ, USA, 1999.

[72] N. Dershowitz and J.-P. Jouannaud, "Rewrite systems," in *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, chapter 6, pp. 243–320, Elsevier, Amsterdam, The Netherlands, 1990.

[73] R. Khardon and R. A. Servedio, "Maximum margin algorithms with Boolean kernels," *The Journal of Machine Learning Research*, vol. 6, pp. 1405–1429, 2005.

[74] P. Boersma and D. Weenink, "PRAAT: a system for doing phonetics by computer," Tech. Rep. 132, Institute for Phonetic Sciences of the University of Amsterdam, Amsterdam, The Netherlands, 1996, http://www.praat.org.

[75] F. Pachet and D. Cazaly, "A taxonomy of musical genre," in *Proceedings of Content-Based Multimedia Information Access (RIAO) Conference*, vol. 2, pp. 1238–1246, Collège de France, Paris, France, April 2000.

[76] G. Cabral, F. Pachet, and J.-P. Briot, "Recognizing chords with EDS: part one," in *Proceedings of the 3rd International Symposium on Computer Music Modeling and Retrieval (CMMR '06)*, vol. 3902 of *Lecture Notes in Computer Science*, pp. 185–195, Pisa, Italy, September 2006.

[77] G. Cabral, J.-P. Briot, S. Krakowski, L. Velho, F. Pachet, and P. Roy, "Some case studies in automatic descriptor extraction," in *Proceddings of the Brazilian Symposium on Computer Music (SBCM '07)*, Sao Paulo, Brazil, May 2007.

[78] B. Defréville, P. Roy, C. Rosin, and F. Pachet, "Automatic recognition of urban sound sources," in *Proceedings of the 120th AES Conference*, Paris, France, May 2006.

[79] J. Monceaux, F. Pachet, F. Amadu, P. Roy, and A. Zils, "Descriptor-based spatialization," in *Proceedings of the 118th Convention of the Audio Engineering Society (AES '05)*, pp. 1–8, Barcelona, Spain, May 2005.

[80] F. Pachet and A. Zils, "Automatic extraction of music descriptors from acoustic signals," in *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR '04)*, pp. 353–356, Barcelona, Spain, October 2004.

[81] D. McEnnis, C. McKay, I. Fujinaga, and P. Depalle, "Jaudio: a feature extraction library," in *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR '05)*, pp. 600–603, London, UK, September 2005.

[82] S. McAdams, "Perspectives on the contribution of timbre to musical structure," *Computer Music Journal*, vol. 23, no. 3, pp. 85–102, 1999.

[83] P. Roy, F. Pachet, and S. Krakowski, "Improving the classification of percussive sounds with analytical features: a case study," in *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR '07)*, pp. 229–232, Vienna, Austria, September 2007.

[84] A. Zils and F. Pachet, "Extracting automatically the perceived intensity of music titles," in *Proceedings of the 6th Conference on Digital Audio Effects (DAFx '03)*, pp. 180–183, London, UK, September 2003.

[85] P. Herrera, X. Serra, and G. Peeters, "Audio descriptors and descriptors schemes in the context of MPEG-7," in *Proceedings of the International Computer Music Conference (ICMC '99)*, Beijing, China, October 1999.

[86] V. Sandvold and P. Herrera, "Towards a semantic descriptor of subjective intensity in music," in *Proceedings of the International Computer Music Conference (ICMC '05)*, Barcelona, Spain, September 2005.

[87] A. Livshin and X. Rodet, "The importance of cross database evaluation in musical instrument sound classification: a critical approach," in *Proceedings of the International Symposium on Music Information Retrieval (ISMIR '03)*, pp. 241–242, Baltimore, Ma, USA, October 2003.

[88] P. Chordia, "Segmentation and recognition of Tabla strokes," in *Proceedings of the 11th International Symposium on Music Information Retrieval (ISMIR '05)*, pp. 107–114, London, UK, September 2005.

[89] P. Roy, F. Pachet, and S. Krakowski, "Analytical features for the classification of percussive sounds: the case of the Pandeiro," in *Proceedings of the 10th International Conference on Digital Audio Effects (DAFx '07)*, pp. 213–220, Bordeaux, France, September 2007.

[90] E. Gómez, *Tonal description of music audio signals*, Ph.D. thesis, Universitat Pompeu Fabra, Barcelona, Spain, 2006.

[91] A. L. Blum and P. Langley, "Selection of relevant features and examples in machine learning," *Artificial Intelligence*, vol. 97, no. 1-2, pp. 245–271, 1997.

[92] C. Molnár, F. Kaplan, P. Roy, et al., "Classification of dog barks: a machine learning approach," *Animal Cognition*, vol. 11, no. 3, pp. 389–400, 2008.

[93] P. Pongrácz, C. Molnár, Á. Miklósi, and V. Csányi, "Human listeners are able to classify dog (Canis familiaris) barks recorded in different situations," *Journal of Comparative Psychology*, vol. 119, no. 2, pp. 136–144, 2005.

[94] P. Pongrácz, C. Molnár, and Á. Miklósi, "Acoustic parameters of dog barks carry emotional information for humans," *Applied Animal Behavior Science*, vol. 100, no. 3-4, pp. 228–240, 2006.

[95] C. Molnár, P. Pongrácz, A. Dóka, and Á. Miklósi, "Can humans discriminate between dogs on the base of the acoustic parameters of barks?" *Behavioural Processes*, vol. 73, no. 1, pp. 76–83, 2006.

[96] T. Draganoiu, M. Pasteau, and L. Nagle, "Do different elements of Black redstart song have different threat values?" *Journal of Ornithology*, vol. 147, pp. 158–159, 2006.

[97] Y. Orlarey, D. Fober, and S. Letz, "Syntactical and semantical aspects of Faust," *Soft Computing*, vol. 8, no. 9, pp. 623–632, 2004.