

Low Dimensional Mapping with Self-Organizing Maps of Genetically Extracted Features from Digital Audio Corpora

Thesis

for the

Degree of Bachelor of Science

in

Cognitive Science

at the University of Osnabrueck

October, 2006

Examinee:

Bastian Tenbergen
btenberg @ uos.de

Supervisors:

Prof. Dr. Kai-Uwe Kühnberger
Dr. habil. Peter Geibel
both at the Institute of Cognitive Science,
Department of Artificial Intelligence
University of Osnabrueck
{ pgeibel | kkuehnbe } @ uos.de

Table of Contents

1. Introduction	3
2. Genetic Algorithms	6
3. Self-Organizing Map	10
4. Clustering Music Files	13
1. Related Work	13
2. Similarity Clustering using GAs and SOMs	19
3. Implementation	21
5. Experiments	24
1. Experimental Set-up	24
2. Results	28
6. Discussion	35
7. Future Work	37
8. References	38

Appendix A

Manual to the Plug-In

Appendix B

Code Documentation

Appendix C

Media

Chapter 1

Introduction

During the last few years, the channels of acquiring musical data have increased rapidly. Due to the growing importance of broad-band network connections, users are no longer limited to radio or television broadcasting services or other non-computer aided methods to acquire copies of musical pieces. More or less legal peer-to-peer networks, online music stores and personalized Internet radio stations provide the user with a constant and never-ending flow of musical information. The choice seems to be endless. On the one hand this large supply makes it possible to get any piece of music any time with a minimal amount of costs. On the other hand, this gives rise to a new, challenging problem: The need of retrieving musical information in large corpora. Keeping the overview over a large collection becomes more difficult and costly with the size of the collection. Unnamed files or files with an unknown content are very difficult to identify and classify manually, which makes a system desirable that is able to accomplish this task automatically.

Many different research teams have addressed this problem in different ways. Research is being conducted on the basis of non-real-world audio data ([Unal2004], [Uitdenbogers2004], [Pardo2004]). That is data of musical information that need to be interpreted in one form or another. Among this kind of musical information belong both score representations of pieces as well as music stored MIDI files. Although research in musical information retrieval based on non-real-world data sets reveals very promising results, the problem is that those kinds of data sets are usually not comparable to the collections that are stored on a user's hard disc or on broadcasting station servers. It is more likely that a form of compressed – either lossy or not, such as MP3 or ogg-vobis files – or uncompressed real-world audio data (as in compact discs or live performances) can be found in a collector's database (throughout this work, the term "user" will be used to refer to any kind of individuals that possess audio data – no matter if it is a home user, a

radio or television station, a web-radio service or a music store, either online or not). Because of this, a system that can deal with real-world audio data is of a much higher public interest than a system that can deal with pre-interpreted score- or MIDI-like representations instead. Processing real-world audio data for music information retrieval purposes can generally be done using two prominent approaches: a statistical or mathematical approach ([Mierswa2005]) and a more neurophysiological approach ([Tzanetakis2001]), both possibly involving neural networks ([Schedl2004]). Since the ability to retrieve musical information from real-world audio data is a very impressive capability of the human brain, especially the application of artificial neural networks similar to neuronal interconnection in the brain provides distinguished results. Both approaches in dealing with real-world data make it necessary to extract certain audio features from the data before hand.

This research presents a way to assist users in the task of finding music pieces in a cluster of similar songs. This is done by presenting the user a set of similar songs according to an input query. This way, a possibly very large collection of song files can be narrowed down to a choice of much fewer songs. This allows for lowering the complexity in comparing songs manually, since a global comparison has already been performed. To accomplish this task, a plug-in for the Machine Learning Environment Yale has been developed. The plug-in makes use of a Self-Organizing Map ([Kohonen1982], [Kohonen2001]) to classify audio files and to create similarity clusters. The files need to be processed in a feature extraction algorithm that extracts certain features ([Mierswa2005]) from the underlying corpus. The ideal feature extraction algorithm is retrieved in terms of evolutionary programming ([Koza1997]).

Classifying songs into similarity clusters allows for a variety of different tasks. As an example, meta information for unnamed song files can be retrieved. Unidentified files can easily be compared with songs in a cluster containing much fewer elements than the original corpus (which might be excessively large). In addition, the suggested system can help automating user preference analysis to individualize the offer of online music stores and other personalized music services.

This paper is divided into three major parts. The first part consisting of sections two and three conveys background information on the two most important approaches in Machine Learning that are used in this research, namely Genetic Programming and Self-Organizing Maps.

The second part (section four) refers to previous work in the realm of Music Information Retrieval and helps to get an insight on how this field has been approached in the last couple of years. In addition, it explains the new approach in real-world music data classification at hand and the plug-in for the Machine Learning Environment Yale is explained.

The third part comprises the sections five, six and seven. It contains detailed description of the experimental design and the results of the experiments are displayed and explained. The results are discussed and the outcome. Some application possibilities are discussed and future work is suggested in the last section of this paper.

Chapter 2

Genetic Algorithms

Approaches in Machine Learning generally try to find an optimal hypothesis, e.g. a way to solve a given task. The scope of tasks can range from decision making as in decision tree algorithms over clustering with neural networks to theorem proving with systems involving inductive logic programming. The domain of genetic or evolutionary programming has proven to be of high applicability to explore a large search space for an optimal hypothesis. Especially in cases, where a lot of possible solutions to a given problem are to be evaluated, evolutionary approaches present a good way to find a way through the hypothesis space and render an optimal solution. In the following, the terms evolutionary programming (EP) and genetic programming (GP) and the terms evolutionary algorithm (EA) and genetic algorithms (GA) are considered to be equivalent.

Although genetic programming has high demands with regard to runtime and memory, solutions to a certain problem can be found with high accuracy in a blind parallel random search-like approach [Koza1997]. The core idea is to develop hypotheses that have an optimal fitness with regard to a given problem. Genetic programming can hence be thought of as an optimization problem, since the fitness is optimized. This is motivated by biological and evolutionary models. The fundamental biological principle is the one explained in Darwin's Theory of Evolution. In this theory, only the fittest individuals of a population survive and get hence the chance to prosper and proliferate. Weak, unfit individuals will not be able to pass their genetic material into the new generation.

Although, this is a very narrowed-down view on the processes that occur between two populations that represent different levels of the food-chain, this principle of natural selection can be applied perfectly on evolutionary programming approaches. Only the fittest hypothesis within the realm of hypotheses that provide a possible solution are kept

and a new generation of hypotheses is based upon the fittest ones of the prior generation. In terms of evolutionary programming, hypotheses are referred to as individuals, a set of possible hypotheses is called a population and the set of all possible hypotheses represents the hypothesis space (i.e. the space containing an optimal solution that needs to be explored). Successive populations within an evolutionary process are called generations.

Individuals are usually represented as executable program code or other kinds of machine readable information, e.g. vector representations or alike. The essential property of any genetic algorithm is a fitness function. The fitness is computed for every individual with regard to a certain task. This determines the likelihood for an individual to be copied into the next generation. Besides the fitness, there are some additional biologically inspired properties that find application in genetic algorithms. These properties are mutation and cross-over. In contrast to measuring the fitness, mutation and cross-over perform direct manipulations on the individual's DNA, that is, the individual's code or alike. Mutation takes a random individual as an argument and returns another individual that differs from the input individual by one single bit (this bit might be a number in vector-like individuals or whole sub-trees in tree-structured individuals). This manipulation can be a simple deletion of such a bit, insertion or alternation. The cross-over operator takes two input individuals and creates two new individuals on the basis of the two parent individuals. During cross-over, sub-trees from the parent individuals are exchanged to form the new individuals.

Both the mutation and the cross-over operator might create syntactically invalid individuals. This means that an individual might have been created that does not execute correctly on a certain task. A concrete example: in the evolutionary algorithm used in this approach, individuals are method trees for feature extraction from music files. A bit in the individual can consist of a sub-tree (possibly consisting of only the root node). The method trees are processed in a depth-first manner, so that a sub-tree's last element's output is the input of the next element in the higher level. If a sub-tree is syntactically invalid, i.e. malformed, so that the next bit cannot process the input data properly, the whole method tree is syntactically invalid. Refer to sections 4.1. and 4.2. for details on the GA approach used in this research project.

The genetic algorithm or the surrounding environment (in this case: the learning environment Yale) hence need to have the ability to check if an individual is syntactically correct. However, this depends as well as the fitness function on the task that is to be learned, since it determines the architecture of the individuals. In other words: The fitness function and the syntactical validator need to be designed with regard to the task that is to be learned, which makes evolutionary programming a highly domain-independent, yet problem specific approach in machine learning.

Due to this fact, evolutionary algorithms have a variety of different parameters that need to be adjusted. The most important parameters are the amount of individuals per generation, the percentage of maintained individuals from the previous generation, mutation and cross-over likelihood as well as the maximum number of generations. The number of maintained individuals is the number of individuals of the previous generation that are copied into the next generation, either mutated or not – depending on the probability of a mutation on an individual to be performed. An ordinary setting for this parameter is a number of n individuals with the highest fitness or just a random selection of n individuals. The cross-over likelihood determines the probability of an individual to be an argument for the cross-over operator. The output individuals of the cross-over operation are also subject to mutation.

The maximum number of generations essentially denotes how many evolution steps are made. Throughout every successive generation, the average fitness of the individuals increases. Hence, the last generation contains the fittest individuals. So, the more generations are performed, the fitter is the fittest individual of the last generation.

These parameters are generally only restricted by the amount of memory and runtime that is available. However, all of the above parameters have a direct impact on the time needed for the algorithm to converge and the quality of the resulting solution. It might hence be the case that in very large search spaces, an evolutionary algorithm can take several days of runtime to converge until a satisfying solution is found.

Below in Listing 1, you can find a brief, abstract evolutionary algorithm in pseudo code.

```

program ea()
  pop = start_pop()
  do *until max number of generations reached*
    pop = next_gen(pop)
  end
end

function start_pop()
  return *a set of randomly generated and syntactically correct
        individuals*
end

function next_gen(curpop)
  newpop = empty_pop()
  copy_inds(curpop, newpop)
  perform_xover(curpop, newpop)
end

function empty_pop()
  return *an empty population*
end

function copy_inds(curpop, newpop)
  do *number of individuals to be maintained*
    ind = find_individual(curpop)
    ind = *maybe mutate this individual*
    add(ind, newpop)
  end
end

function perform_xover(curpop, newpop)
  do *number of individuals to be crossed over*
    mother = find_individual(curpop)
    father = find_individual(curpop)
    ind = cross(mother, father)
    ind = *maybe mutate this individual*
    validate_ind(ind)
    add(ind, newpop)
  end
end

function validate_ind(curind)
  *check if an individual is syntactically correct*
end

function find_individual(curpop)
  ind = *find individual with highest fitness*
  if (delete_used_individuals) delete(ind, curpop)
  return ind
end

```

Listing 1. A generic evolutionary algorithm in pseudo code.

Chapter 3

Self-Organizing Maps

The Self-Organizing Map (SOM) is a neural network structure that helps to map high dimensional data onto a low dimensional representation. A Self-Organizing Map is basically a Multi-Layer Perceptron with one input layer with n neurons, no hidden layer and one output layer with m neurons. The input is a d dimensional vector, $x \in \mathbb{R}^d$, the output is $y \in \{0,1\}$. The activation of the input layer is equal to the input. Self-Organizing Maps are winner-takes-all-networks, where only one neuron of the output layer is active. All others are silent. Every output neuron computes the scalar product of the input vector and the weight vector. The winning neuron i^* , i.e. the only neuron that is active in the output layer, is the one neuron that maximizes net activation, that is, the neuron with the highest scalar product. The winner neuron's weight vector is updated according to the rule below.

$$\mathbf{w}_i(t+1) = \begin{cases} \mathbf{w}_i(t) + \epsilon (\mathbf{x} - \mathbf{w}_i(t)) & , \quad \text{if } i = i^* \\ \mathbf{w}_i(t) & , \quad \text{else} \end{cases}$$

$\epsilon :=$ learning rate

Equation 1. SOM update rule.

In this rule, w_i is the weight vector of the i -th neuron in the output layer, t is a iteration step, x the input vector and ϵ the learning rate.

In other words, the neuron with the weight vector that is closest to the input pattern is "pulled" into the direction of the input pattern by updating the weight vector according to

the learning rule. This means that throughout the iteration of all training steps of the network, the neurons will move to the areas of the input space, where sets of similar input patterns can be found. The neurons become prototypes of a input cluster and structure the input space. This is illustrated in Figure 1 below. In Figure 1, the colored spots represent input patterns that exist in clusters in the input space. The dashed vectors represent the weight vectors of the neurons in the SOM before all iteration steps have been performed. The un-dashed vectors illustrate the weight vectors after the classification. Please note that this diagram is only a rough scatch.

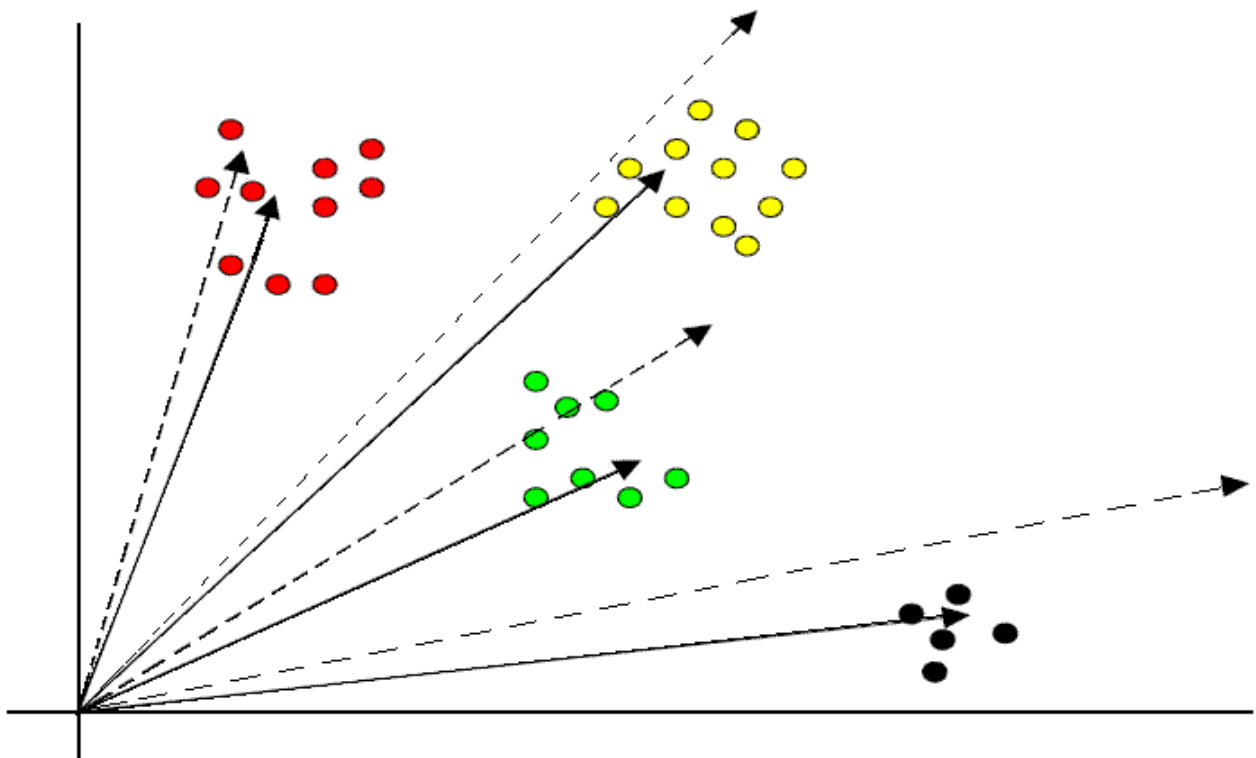


Figure 1. Neurons in a SOM become prototypes of clusters in the input space.

By assigning a neighbor relation between two adjacent neurons, a non-linear topological mapping of the high dimensional input patterns onto a usually two dimensional grid can be realized [Kohonen2001]. The most common topologies of neurons are rectangular and hexagonal structures. In the latter structure, neurons are aligned in hexagons, whereas in rectangular structures, neurons are aligned in an $n \times m$ matrix. Other topologies are row-like and circular structures, which are an $n \times 1$ matrices, and differ from each other in the relation from the first and last neuron. These are bidirectionally connected in a circular structure.

Previous research [Kohonen1982] has proven that rectangular and hexagonal topologies provide the best results in clustering the input space.

Assigning neighbor relations and hence creating topologies in a SOM can therefore help to illustrate the input space in a much less dimensional representation. This can be achieved by mapping the input onto the SOM or mapping the Self-Organizing Map onto the input space.

Self-Organizing Maps have been successfully applied in many different fields of research. SOMs have proven to be valuable tools not only in data mining and clustering tasks, also pattern recognition can be easily achieved by making use of this special type of neural network.

In this research, very high dimensional feature vectors are extracted from digital audio files. Since the aim is to find similar songs, the Self-Organizing Map seems to be predestinated to accomplish this task. A low dimensional mapping of the high dimensional feature vectors is the ideal way to represent similar audio files.

Chapter 4

Clustering Music Files

The aim of this research is to assist a user to manage large music collections or retrieve similar songs from a very large music database. A system is suggested that helps to find a particular song or a set of songs amongst many by reducing the search space (i.e. the user's music collection). The search space reduction is performed by filtering out those songs which are not similar to an input song and are hence unlikely to be amongst the set of songs the user is looking for. In order to accomplish this rather challenging task, a system is needed that works on the basis of real-world audio data. In the past couple of years, research on Music Information Retrieval has been conducted with an ever growing diligence, possibly due to the growing importance of digital music distribution and the therefore increasing size of private music collections. Research focuses on both real-world and non-real-world audio data.

In the next section, this research is briefly reviewed. An overview is given on different aspects of Music Information Retrieval and the most important work with regard to this research project is described in more detail. The second section deals with the aim of this research and describes the motivation behind it before the last section explains the implementation of the software that has been developed within the scope of this research.

4.1. Related Work

Classifying audio data is a challenging task keeping in mind that raw audio files are not machine readable in a sense that an excerpt of polyphonic musical information can be decomposed into its constituents. A piece of musical information is considered polyphonic if more than one tone is played at the same time, i.e. when waveforms of different tones

are superimposed to form a new, single waveform and are hence conceivable at the same time index. Without complicated heuristics, a computer is not likely to be able to differentiate certain instruments used in the piece or to distinguish the bass-line from the melody (keeping in mind that the concept of "melody" is a rather obscure one, see [Uitdenbogers2004]) or even tell the difference between the first and second voice. These abilities seem to be of human nature only; especially the last one seems to be reserved for professional musicians or very talented humans only. However, mathematically speaking, music data – either compressed or not – can be considered a time series, or – more general – a value series that can be analyzed using several mathematical techniques ([Mierswa2005]). These mathematical techniques can extract certain values from the time series and will therefore be referred to as feature extraction methods in the following passages.

Many attempts have been made to computationally analyze music and build frameworks for music classification and music information retrieval to assist humans in the organization of large music corpora. The variety of investigations ranges from Query-by-Humming Systems ([Unal2004] and [Pardo2004]) over full-text indexing ([Knuth2001]) to metadata-based approaches as described in [Pachet2001]. In this paragraph, related work with the largest impact on the research at hand is briefly discussed.

A lot of research has been conducted on the basis of non-real-world audio data. Music that is represented in scores or a MIDI-like format is without further interpretation not perceivable by a human listener and is hence only to a limited extent adequate to structure a user's collection of MP3 files or such. However, these researches have provided quite promising results in the realm of Music Information Retrieval.

Pardo et al in [Pardo2004], have conducted a pilot study in matching a sung query to a database of stored MIDI files. The authors make use of a similarity ranker in form of String-Alignment methods as well as hidden Markov Models. They especially addressed the problem of erroneous queries due to a low humming performance of the user. The outcomes of the experiments were that specifically designed human-based error models provide much better results in handling a mismatch between query and target songs than the complete absence of any error model. Yet, synthetic error models provide equally good

results than data-driven, singer-based error models.

MIDI file-based Query-by-Humming Systems have been investigated by Unal et al, too, [Unal2004]. In contrast to [Pardo2004] however, Unal et al focused on handling query errors on the basis of psychoacoustic considerations. This approach is closer to neurological functions in the human brain than the approach by Pardo et al, but shares the limitation that only a very narrow amount of features from the songs are actually used.

Another approach based on MIDI-files has been conducted by Uitdenbogerd and Zobel, [Uitdenbogers2004]. The researchers paid special attention to the polyphonic music events and variations in key and pitch in order to present an efficient way to return similar songs in clusters. They focused on the song's melody, pointing out, however, the subjective nature of the concept of "melody". Uitdenbogerd and Zobel found out that both the similarity measurement techniques used – String matching or n-gram matching – provide good results in similarity measurement of musical data, but are dependent on the actual task – querying a large corpus or comparing two songs. The authors propose a very efficient system to retrieve similar songs from large corpora but mention that a system handling real-world audio formats instead of MIDI representation is desirable.

A very naturalistic research was conducted by Pachet and Laigre, [Pachet2001]. In contrast of approaching the task of Music Information Retrieval, the authors focus only on the environmental metadata (i.e. data retrieved from the context, music files can be found in instead of metadata such as ID3-Tags of MP3 files, see [Hacker2000]). In order to structure a – possibly very large – set of audio files of any format, the file name as well as the cluster (e.g. directory) the file occurs in are taken into account. The authors present a number of heuristics to retrieve information like the song title or artist name and hence tag the files in the corpus. This approach has the advantage that it is easy to implement into other systems. It helps to avoid multiple entries of files as well as it provides a way to disambiguate a corpus by homogenizing the file names.

Content-based approaches on real-world data promise good results in the realm of Music Information Retrieval. Allamanche et al, [Allamanche2001], investigated similarity metrics between audio data using MPEG-7 low-level descriptions. They did so by extracting "audio

fingerprints", i.e. some audio features, from the files and comparing them using a nearest neighbor rule. A vector quantization algorithm is used to determine classes. The authors paid special attention to the robustness of their system against a variety of distortions and to the applicability on standard home computers. The proposed system shows good results in identifying music on a content-based level.

Liu, Wang and Chen, [Liu1998], adopted a slightly different approach for classification. They used low-level features derived from audio content of a video scene to determine the semantic context and hence classify a video scene. The authors have extracted certain features from the audio content, a neural network classifier, i.e. a generic Multi-Layer Perceptron, is used to cluster the video scenes into different types. This is an approach in musical genre categorization by feature extraction and neural network classification, although the goal was to discriminate different TV-scene types. It is to note, however, that the application of a Multi-Layer Perceptron requires a-priori assumptions concerning the number of classes, the corpus is to be clustered in. Since there is a very large amount of possible classes for audio (and, as in this case, video) material so that the a-posteriori findings are always restricted on the assumptions.

Tzanetakis et al have contributed an important milestone in musical genre classification, see [Tzanetakis2001]. The authors argue that human listeners can classify musical content by referring only to small excerpts of a piece, e.g. a few fractions of a second. Hence, human listeners must rely on musical surface features instead of complex (mathematical) interaction between waveforms of the piece. So, Tzanetakis et al focus on the extraction of surface and rhythm properties and propose a set of own features. Using a Gaussian classifier, they were able to classify a variety of different musical genres. The authors proposed two different graphical user interfaces (GUIs) that visualize a song's membership to a certain genre. Since the borderlines of musical genres are rather fuzzy, a nice side effect of the GUIs is that not only the genre of a song is displayed but rather a number of other genres, the song shows characteristics of are visualized.

Kurth and Clausen [Knuth2001] present a way to retrieve exact matches of audio excerpts from very large audio databases. Using certain algebraic indexing techniques and the extraction of surface features, the authors present a system that returns all occurrences of

a given query. Not only the song name itself is returned, but the system also returns the exact time index in which the query can be found. This is done by storing pattern doubles of the form $((time_i, classification_i)) \quad i \in I$. The authors hence do not structure the search space by songs, rather by time indexes, which allows for the retrieval of exact matches. The authors emphasize that retrieving matches from audio databases can be subdivided into two aspects: Finding exact matches and finding a set of matches that are similar to the query. This research focuses on the second aspect and is therefore complementary to the work in [Knuth2001].

Schedl et al [Schedl2004] made use of a topology preserving neural network structure, i.e. a Self-Organizing Map, to map similarities in musical pieces on a perceptual level. In their paper, the authors give a short historical overview on three visualization user interfaces that they developed in the past couple of years. In their latest visualization system, the authors include metadata such as directory contents and ID3-Tags in order to give an even more thorough structuring of the search space. The authors however use only a limited amount of features. The advantage of the author's system is that no assumptions need to be made for the clusters, the music files are classified in. The system is hence robust against a large variety of distortions as well as large, complex heterogeneous corpora.

Up to this point, all researchers who extract features from digital audio data mostly presented their own features. This leads to a large variety of features that can be extracted from audio data. However, a unifying framework that measures the feature's applicability was missing. Mierswa and Morik, [Mierswa2005], were the first ones to address this problem. In order to accomplish this task, Mierswa and Morik made use of method trees, built of feature extraction algorithms that are combined sequentially so that data can flow from algorithm to algorithm. At the end, the output consists of the extracted features from the time series that is input to the method tree, i.e. the music piece that features need to be extracted from. The elements (feature extraction methods) in the method trees (the authors call these sequences "chains", though their definition is slightly more complex) can consist of chains themselves. Hence, a tree structure is created, which gives the name earlier mentioned: method trees. The data in the tree flows in a depth-first-manner from root to the first child, then to the second child, and so on until the n th child, that delivers

the output. If a child itself is a root of a sub-tree, the last child of that sub-tree delivers the data back to the root, so that the sequence can continue in the next higher tree level. A method tree is generated by randomly combining the feature extraction methods.

The universe of all possible method trees (hypothesis space) is explored in order to find the ideal method trees and hence the ideal way to extract features from a certain type of audio file. The authors apply a Genetic Programming approach to explore the hypothesis space. In the GA, the individuals are the method trees. A set of method trees is a population and successive populations are generations. The mutation operator simply inserts a feature extraction method from the method tree, deletes it or substitutes it by another one that provides the same syntactical structure of the method tree. The cross-over operator replaces one method tree's sub-tree by some other method tree's sub-tree respecting syntactic validity.

This GA approach is somewhat different from ordinary evolutionary programming paradigms, since instead of the individuals being represented as binary vectors or executable program code, the individuals are method trees. The fitness metric hence cannot be merely a function but is the performance measured by another learning algorithm. An individual's fitness is measured by applying an excerpt of the source directory on the individual to extract some features. A Support Vector Machine classifies the audio files, from which the individual has extracted the features, into one of two classes (each class is a genre – either Pop/Techno, Pop/HipHop, Pop/Classic; see below for details). The error of the SVM classification is estimated by a k-fold cross validation. The mean accuracy, recall and precision determine the fitness of the individual at hand with respect to the audio data it has been applied on.

The selection of method trees that are copied into the succeeding generation and of method trees undergoing the cross-over operator is based on the individual's portion on a proportionally segmented circle, i.e. fitness parts out of 360 degrees. Individuals with a larger portion are considered fitter than ones with a lower portion and have a higher chance to be chosen for the next generation and cross-over.

Due to the advantages of a Genetic Programming approach in exploring very large hypothesis spaces, it becomes obvious why such an approach is used to build a unifying framework for feature extraction methods. Mierswa and Morik used this evolutionary algorithm in order to learn method trees that have an ideal performance to classify audio data into genres. Three test corpora were compiled and consisted of music files that belong to either the Pop or Techno genre, the Pop or Hip hop genre or to Pop or Classic music. The files in every corpus needed to be classified in either of the two genres in the corpus.

To learn more about the features used in the approach by Mierswa and Morik, please refer to [Mierswa2005], [Mierswa2006], [MierswaV3.3] and [Fischer2002].

4.2. Similarity Clustering using GAs and SOMs

The goal of this research is to present a framework that classifies an arbitrarily large corpus consisting of music files stored in the generic MP3 format into an arbitrary number of similarity clusters. This framework will assist users in searching for a particular song or set of similar songs in the private collection by filtering out those entries that are not similar to a queried file, the search is based on. This aim is complementary to the focus of the research in [Knuth2001], since Knuth and Clausen aimed at finding exact matches to a queried file.

It is necessary, to make the system robust against a wide range of musical styles. Therefore, two important considerations need to be fulfilled. First, the more inhomogeneous the corpus is, e.g. the larger the variety in genre and style amongst the songs is, the larger can be the variety of songs' features. Hence, different features might be necessary to be considered for different songs. Second, the number of similarity clusters is not known prior to the classification. It can therefore be assumed that the number of similarity clusters increases with an increasing broadness of the collection. The rest of this section will address both considerations and will explain which approaches have been chosen to meet the above requirements.

There is a large variety of different feature extraction methods. The first repository has been organized by Mierswa and Morik in [Mierswa2005]. Since the aim of this research was not the exploration of methods to analyze raw audio data, details about what feature methods exist are omitted here. Instead, this research uses these methods to imitate the human ability to compare songs on a perceptual level in order to cluster them according to their similarities.

Essentially, the work done by Mierswa and Morik is very close to the idea of the present research. Although, the crucial difference is that this research does not aim at determining either one of two genres for a corpus consisting of songs from the two genres. Instead, it is the aim to cluster songs according to their similarity. The test corpus is not pre-structured into rough similarities. In addition, the problem of determining if a set of songs is similar or not presents itself as an even more subjective and hence harder task than to pinpoint a song's genre, especially if there is only a choice of two genres it can be a part of (although the boundaries between certain genres are somewhat fuzzy).

Since both, the approach of Mierswa and Morik and the present research need to extract the best possible feature set from a corpus of digital music data, and since the evolutionary algorithm in the work of Mierswa and Morik provided excellent results, the present research makes use of the same idea to extract features from audio files. The genetic algorithm is assumed to find the ideal feature extraction method with respect to the underlying corpus and hence takes into account the – possibly very large – variety of different song-styles and genres. Since the method trees resulting from the evolutionary algorithm in the above approach do not extract features themselves, it is necessary to apply those method trees on the raw data again so that the features can be extracted. However, feature extraction does not entail classification. Hence, a framework is needed to take care of the classification process.

In the realm of Machine Learning, there is quite a choice of algorithms that can classify patterns (e.g. music files) into a certain amount of classes. Although decision tree algorithms like C4.5 or standard neural networks like Multi-Layer Perceptrons provide very promising results in doing so, it is however necessary that there is a fix amount of classes, the input patterns are classified to. Only Support Vector Machines and self-organizing

Kohonen networks seem to accomplish the task of classifying the input space into a unknown number of clusters and therefore meeting the second consideration mentioned above.

In previous research the extracted features have been used to classify songs into musical genres. The algorithms that extract the features from the music files have been designed by the original authors ([MierswaV3.3]) to provide values that comply with a wide range of different operators for the learning environment Yale. It can therefore be inferred that the extracted features comply with a SVM classification process as well as a SOM clustering technique.

The extracted feature vectors are of potentially very high dimensionality. High dimensional data however are quite unhandy when it comes to a user-friendly classification, so a paradigm is desirable that is able to handle high dimensional data in a user-friendly way. SOMs have proven to provide excellent results in classification tasks, which can be seen from [Schedl2004]. They can also cope with very high dimensional input by lowering the dimensionality to a usually two-dimensional representation, which has the nice side-effect of making similarity relations between clusters "visualizable". Self-Organizing Maps are hence the ideal paradigm to classify the input patterns.

4.3. Implementation

The genetic algorithm used in this approach to find an ideal feature extraction method and the feature extraction process itself have already been implemented for the Machine Learning Environment Yale. See [Mierswa2006] and [Fischer2002] for details.

Hence, a software framework was needed to fulfill the task of classifying the patterns into similarity clusters. In order to accomplish that task, a plug-in for the Machine Learning Environment Yale has been developed. Special emphasis was placed on an easy integration into Yale and the adaptation to many different corpora. In the following, the implementation of the plug-in is described.

For a broader insight, please refer to Appendix A.

The plug-in contains a simple Self-Organizing Map operator. It can be found in the group “Kohonen Networks“ in Yale's operator list after the plug-in has been installed properly. The operator can be used as a standalone experiment or as part of a larger experiment framework. Please refer to [Fischer2002] for details.

When the operator is applied, the parameter settings are retrieved from Yale's user interface. After that, a SOMFileHandler-object is invoked. This object takes the path to a feature vector file as set in the corresponding parameter as one argument and transforms the training patterns (i.e. feature vectors) from a string representation into double representations. The result is a feature pattern matrix containing the extracted features. In addition, the SOMFileHandler-object stores a song list containing the songs that have been processed from the source directory. This allows for a more user friendly representation of the clusters. As soon as the training patterns are retrieved, the Self-Organizing Map-object is invoked according to the parameter settings for the number of neurons and topology, the learning rates and the amount of classification runs. The Self-Organizing Map runs the classification given the feature vectors. When the classification finished successfully, the distance between the input feature vectors and the final weight vectors of the SOM need to be measured. This is done by making use of a Distance-object. This object looks for a distance \hat{d} that minimize the functions

$$d(\mathbf{x}, \mathbf{y}) = \sqrt[\lambda]{\sum_{i=1}^n |x_i - y_i|^\lambda} \quad \lambda \in \mathbb{R}$$

Equation 2. Minkowski Distance.

With $\lambda = 1$: Manhattan Metric, $\lambda = 2$: Euclidean Distance.

or

$$d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^n (1 + m) \quad m = \begin{cases} 0 & x_j = y_j \\ 1 & \text{else} \end{cases}$$

Equation 3. Nominal Distance.

respectively (x is a pattern, y is a neuron and n the number of features). The measurement is performed with regard to the parameter setting for the metric that is supposed to be used. It can be found in [Schedl2005] that no superiority of different metrics exist to measure the distance between vectors. Hence, three rather simple metrics have been implemented, i.e. Euclidean distance, Manhattan metric and Nominal distance, although the experiments focus on the Euclidean distance (see section 5.1).

The result of this distance measurement is an Integer-array containing the index of the neuron that is closest to a given pattern. The patterns are represented by the index of the array itself. So, the term $pos[5] = 3$ means that neuron number three is the closest neuron to pattern number five. As soon as the distances are measured, the file clusters are created with respect to the distances between patterns and neurons relations and the song list that is retrieved from the SOMFileHandler-object. The result is a number of Cluster-objects, each of which consists of an individual ID and the list of songs, i.e. music files in the source directory, whose feature vectors are similar to one another. These Cluster-objects can then be scanned for the file name of the queried file. The cluster containing this file name is then returned to the user. Please refer to Figure 2 below for a detailed schematic representation of the inner architecture of the Self-Organizing Map plug-in

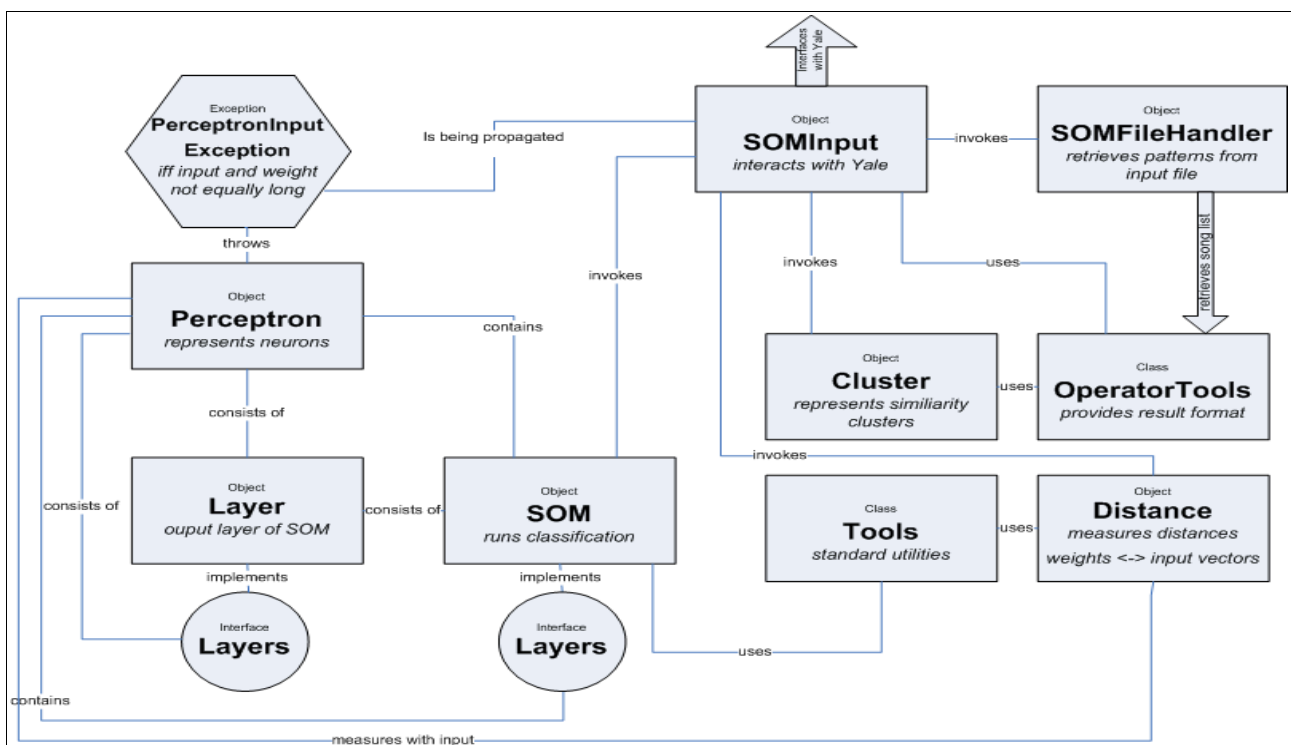


Figure 2. A detailed schematic representation of the SOM-Plug-in's inner architecture.

Chapter 5

Experiments

Classifying raw audio data into similarity clusters is basically a three step process. It requires multiple experiments in the Machine Learning Environment Yale to be run and evaluated accordingly. These experiments come as xml-files in a separate package or in the full package together with the plug-in. In order to learn how to use the sample experiments and to get an overview on how to install and use the plug-in, please refer to Appendix A. You can obtain a copy of the plug-in by also referring to Appendix A or take it directly from Appendix C. In the next section, the experimental design is explained. The test corpora are described and most important experimental parameters are defined. In addition, the experimental procedure is explained. The second section of this chapter displays the results of the experiments.

5.1. Experimental Set-up

The underlying requirement of all experiments is that the audio corpus on which the experiments are performed consists of generic MPEG-1 Layer III files with a sampling rate of 44.1KHz [Hacker2000]. Such MP3 files are the most common way to distribute music amongst computers, so this research focuses the attention on them. The proposed classification framework is robust against corpora consisting of files encoded with different bitrates.

The primary test corpus consisted of 360 audio files, all differing in length, bitrate and genre. These files were subdivided into six sub-corpora (i.e. subdirectories): three small ones, each containing approximately 20 files, and three medium ones, each containing 100 files. The files in the small sub-corpora were taken from a private collection and vary in

style and genre. The medium corpora contained the (unofficial) top 100 chart songs from the years 1990, 2000 and 2005, respectively. To test the proposed system for applicability in very large collections, a secondary corpus consisting of 1,554 files, i.e. the (unofficial) charts from the years 1990 to 2005 has been compiled and processed. Although an effort has been made to take songs from as many different genres as possible, this research focuses only on modern popular music, ranging from early pop stages to fairly recent modern publications. Recapitulating, the overall music corpus consisted of approximately 8.07 gigabytes in 1,914 files summing up to over 160 hours of total playtime. The files in the source directory are generally not structured although recursive processing of subdirectories is supported. Similarity clusters are found on the bases of a query file. This file is located within the source directory and is hence processed in terms of feature extraction too.

The basic experimental procedure has been divided into three steps. In the first step, the evolutionary algorithm takes the raw data from the local hard disc and learns an ideal feature extraction method with regard to the test corpus. In the second step, the experiment file containing the feature extraction method returned by the GA is used to transform the raw data into a feature vector representation. The output of the feature extraction step is a file containing the feature vectors. In the third step, this file is used by the classification operator to train the Self-Organizing Map. In addition, the classification operator retrieves the file name of the query file from the raw audio data. After the classification is done, this file name is retrieved from the similarity clusters that have emerged from the classification. The system's output is a string representation of all file names of the songs that are in the same cluster as the query file. Below in Figure 3 the principle experimental layout is displayed.

For the evolutionary algorithm, an iteration size of a maximum of 50 generations was chosen. However, an early stopping criterion was introduced by limiting the number of generations without improvement of the average fitness to a fifth of the maximum generation size. The number of individuals per generation was set to three times the size of the respective test corpus, however maintaining a minimum of 100 individuals. The

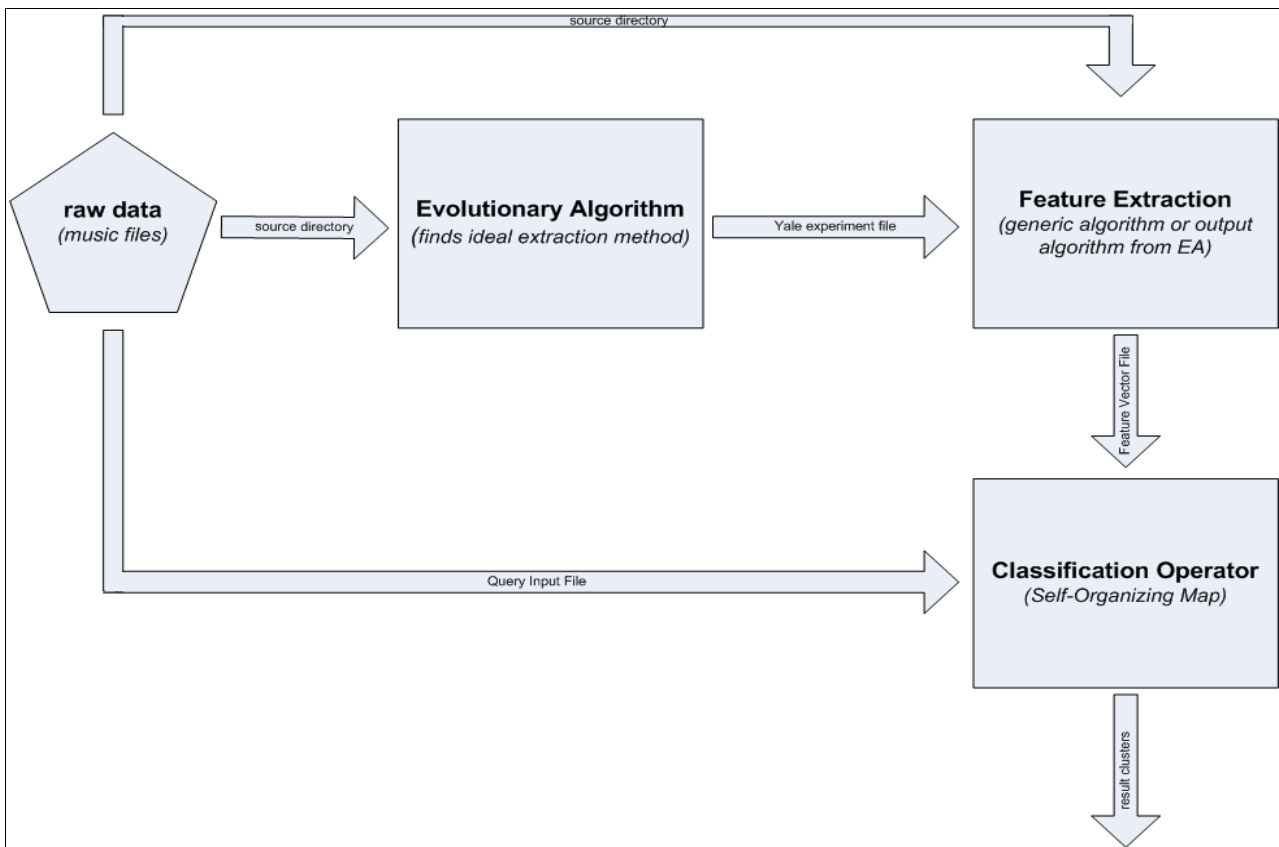


Figure 3. Experimental Layout.

probabilities for an individual undergoing cross-over and mutation were set to 0.5 and 0.2 respectively. An effort has been made to copy the fittest individual of a population into the succeeding population and hence ensuring the survival of that individual. In the feature extraction step, a maximum of 65,536 samples per batch has been chosen to limit the amount of memory needed by the algorithm. The remaining parameters for the evolutionary algorithm and the feature extraction step were set to default values.

In order to test the clustering performance, the Self-Organizing Map was applied on the extracted features with a variety of different parameter settings for the overall number of neurons. Since the number of clusters cannot exceed the number of neurons in the SOM, but increasing the number of neurons negatively influences the time needed to finish the classification, an ideal number of neurons that allows for a maximum of clusters needs to be found – a trade-off. All test runs of the SOM had the following parameters in common. The network topology was always set to an “open” $n \times m$ matrix, with $n, m > 1$, i.e. a rectangular structure with open topology. The number of training runs (iteration steps) was set to 1,000 and the learning rates for the winner neuron and its neighboring neurons was

set to 1 and 0.25 respectively. According to [Schedl2005], no superiority of any distance metric over another metric exists. Hence, all test runs have been performed using the Euclidean distance as the metric.

To test, which features are most important for the classification task, test runs were performed with normalized and filtered patterns. The patterns have been normalized by dividing each pattern $x(i, j)$ – with i being the i -th pattern (i.e. i -th song from the source directory) and j being the j -th feature of the i -th pattern – by the root mean square deviation of the respective column in the pattern matrix. A filtering of the patterns was done by setting every pattern with a very small absolute value – $|x(i, j)| < 1$ – to zero.

All three small sub-corpora and all three medium sub-corpora have been tested in the same way by first applying the evolutionary algorithm on the corpora. After that, the features are then extracted using the fittest individual from the GA (i.e. the ideal feature extraction method with regard to the underlying corpus). Since the purpose of the secondary corpus was to ensure the system's robustness on very large corpora consisting of over 1,500 files, the genetic algorithm was not used on the secondary corpus. This was meant to reduce the time and space complexity of the test run. Instead, a generic feature set that has proven good results in a variety of tasks has been extracted from all files in the large corpus. All extracted feature sets, either generic or evolutionarily retrieved, are classified with the Self-Organizing Map using different settings to find an optimal clustering result.

From each corpus, one file was chosen to be the query file. Since the query file is the file that users will enter as a sample of their musical taste (for instance), the evaluation focuses on the cluster containing this file to ensure that the suggested similar songs are indeed similar. To evaluate the similarity of the songs in the cluster containing the query file, the cluster was transposed into a play list and evaluated by a human user. The user was asked to rate each song on a scale from zero to five (0: no similarity; 1: little similarity; 2: medium to little similarity; 3: medium to high similarity; 4: high similarity; 5: very similar/close to equal) using the query file as the prototype.

The user was asked to consider two songs a and b similar, if the

- musical style of both songs is subjectively similar,
- musical themes can be described as subjectively similar, and
- the pace and/or rhythms of the songs are subjectively similar.

Two songs can also be considered similar, if they are not members of the same genre.

The user was a twenty-two year old male German student, with no background in music theory and music science. The participation in this study was on a voluntary basis and the user did not receive a reward. Prior the ranking of the songs, the user was naive to the purpose of this study to avoid a user bias. Although the judgment of the similarities cannot lead to significant findings because only one single user was asked to rate the songs, it is still a good way to compare the system's findings to an ordinary user's musical judgment.

5.2 Results

While conducting the experiments, special attention has been paid to the performance of the genetic algorithm and on the clustering performance of the Self-Organizing Map.

Summarizing, there are two major outcomes that can be found amongst the results. Firstly, the genetic algorithm did not provide feature extraction methods that allow for high clustering performances. Contrastingly, applying a *generic* feature extraction method enabled the Self-Organizing Map to classify the music files in the source directory more accurately into similarity clusters. The second major outcome is that the Self-Organizing Map was able to find similarity clusters among the music files and hence provided good results in filtering the source directory. These results are explained in the rest of this section in more detail. To get an idea of the clustering performance, Tables 1 to 3 show some example similarity clusters.

One very important finding in the present research is that applying a genetic algorithm to find the best possible feature extraction method for an arbitrary set of music files does not provide optimal results if the music files are afterwards clustered on the basis of extracted features with a Self-Organizing Map. This can be seen from the fact that the genetic algorithm finds exceptionally short individuals (i.e. feature extraction methods) for rather small corpora. The genetic algorithm was applied on all three small and all three medium corpora. The GA finds optimal individuals for all six test corpora, however the length of the respective optimal individuals of each new generation decreased.

<i>Similar Songs in Cluster (ranking):</i>	<i>Similar songs not in cluster (ranking):</i>
Nick Kamen I promised myself (5)	Enigma Sadness (5)
Sandra Hiroshima (5)	Matthias Reim Verdammt, ich
Technotronic Get up (4)	lieb' Dich (4)
FPI Project Rich in Paradise (4)	Roxette It must Have
Roxette Dangerous (4)	been Love (4)
Billy Joel We Didn't Start	49ers Touch Me (4)
the Fire (4)	Guru Josh Infinity (3)
David Hasselhoff	P.M. Sampson & Double Key
Crazy for you (4)	We Love to Love (3)
Bombalurina Itsy Bitsy Teeny	Chocolate Ritmo de la Noche (3)
Weeny Yellow Polka (4)	
Matthias Reim Ich hab geträumt	
von Dir (3)	
MC Hammer U Can't Touch This (3)	
Gianni Nannini Un'estate	
Italiana (3)	
Beats International Dub Be Good	
to Me (3)	
Black Box I Don't Know Anybody	Corpus Size : 100
Else (3)	Neurons : 10x8
Wilson Phillips Hold On (3)	Filtering : No
Torfrock Beinhart (3)	Normalization : No
Alannah Myles Black Velvet (2)	Total # of Clusters : 20
Real McCoy & MC Sar	Similar files : 23
It's On you (2)	
IceMC Scream (2)	Target Song:
Was Not Was Papa Was	Depeche Mode Enjoy the Silence
a Rolling Stone (1)	
Eros Ramazotti Se Bastasse Una	
Canzone (1)	
Paula Abdul Opposites Attract (1)	
Norbert und die Feiglinge	
Manta (0)	
Billy Joel Leningrad (0)	

Table 1. Similarity Cluster for "Enjoy the Silence" by Depeche Mode.

Surprisingly, none of the individuals that the evolutionary algorithm has found to be the fittest one on a specific corpus was able to provide a feature set that allowed the Self-Organizing Map to reach good clustering performance. Applying the SOM on a feature set of one of those individuals caused the neural network to find only one or two clusters, in rare cases three clusters, where two of the three clusters only contained one file. To ensure that this finding does not depend on the properties of the genetic algorithm, the EA

<i>Similar Songs in Cluster (ranking):</i>	<i>Similar songs not in cluster (ranking):</i>
Band Ohne Namen Take my Heart (5)	Highland Bella Stella (5)
Die 3. Generation Ich will, dass Du mich liebste (5)	Echt Weinst Du (5)
Laura Immer wieder (5)	Reamonn Supergirl (4)
Die Ärzte Wie Es Geht (4)	Orange Blue She's Got That Light (4)
Sisqo Thong Song (4)	R Kelly If I could Turn back Hands of Time (4)
Madonna Music (3)	Rednex Spirit of The Hawk (3)
Anastasia I'm Outa Love (3)	Santana Maria Maria (3)
Manu Chao Bongo Bong (3)	Madonna American Pie (3)
Music Instructor feat. Dean Super Fly (3)	Sting Desert Rose (3)
Gigi D'Agostino The Riddle (0)	Gabrielle Rise (3)
Limp Bizkit Take a Look Around (0)	Corpus Size : 100
ATC My Heart Beats like a Drum (0)	Neurons : 15x15
Mauro Picotto Komodo (0)	Filtering : No
Vanessa Amorosi Absolutely Everybody (0)	Normalization : No
Alex Ich will nur Dich (0)	Total # of Clusters : 7
	Similar files : 15
	Target Song: Metallica Nothing else Matters

Table 2. Similarity Cluster for "Nothing else Matters" by Metallica.

has been re-applied on a medium test corpus with a variety of different settings other than those mentioned in section 5.1. Yet, no resulting individual provided a feature set suitable for the SOM classification.

Since the retrieved feature extraction methods did not provide satisfying results, a generic feature set has been used to extract features from all test corpora. The Self-Organizing Map has been applied on the features to cluster the files into similarities. A variety of different settings has been used to determine the optimal clustering performance.

In the small test corpora, only a very limited number of clusters could be found. Increasing the number of neurons did not influence the number of clusters. Accordingly, the optimal number of neurons for the medium test corpora does not depend on the number of found similarity clusters as well, as long as the SOM contained more neurons than the maximum

<i>Similar Songs in Cluster (ranking):</i>	<i>Similar songs not in cluster (ranking):</i>
Elton John & George Michael Don't Let The Sun Go Down on Me (5)	Band Ohne Namen Take my Heart (5)
Garland Jeffreys Hail Hail Rock 'n Roll (5)	Highland Bella Stella (5) Echt Weinst Du (5) Die Ärzte Wie Es Geht (5)
Michael Jackson Heal the World (5)	Die 3. Generation Ich will, dass Du mich liebst (5)
Youssou Ndour & Neneh Cherry 7 Seconds (5)	Laura Immer wieder (5) Reamonn Supergirl (4)
Meat Loaf I'd Do Anything for Love (5)	Orange Blue She's Got That Light (4)
Young Deenay Walk On By (5)	R Kelly If I could Turn back Hands of Time (4)
Thomas D Liebesbrief (5)	Sisqo The Thong Song (3)
Christina Aguilera Beautiful (5)	Rednex Spirit of The Hawk (3)
Coldplay Speed Of Sound (5)	Santana Maria Maria (3)
Hypertraxx The Darkside (4)	Madonna Music (3)
Nelly Furtado I'm Like a Bird (4)	Madonna American Pie (3)
Atomic Kitten It's OK (4)	Sting Desert Rose (3)
Daniel Bedingfield If You're Not The One (4)	Gabrielle Rise (3)
Nomad I wanna give you devotion (4)	Anastasia I'm Outa Love (3)
Salt 'n Pepa Lets Talk about Sex (3)	Manu Chao Bongo Bong (3)
Ace of Base Don't Turn Around (3)	Music Instructor feat. Dean Super Fly (3)
Dune Hardcore Vibes (3)	(results truncated)
Chris Brown Run It (3)	
Dru Hill How Deep is Your Love (2)	Corpus Size : 1,554
J-Kwon Tipsy (2)	Neurons : 15x15
Color Me Badd I Wanna Sex You Up (1)	Filtering : No
Cher Believe (1)	Normalization : No
Interactive Living Without Your Love (0)	Total # of Clusters : 143
Wolfgang Petry Die Längste Single der Welt (0)	Similar files : 24
	Target Song: Metallica Nothing else Matters

Table 3. Similarity Cluster for "Nothing else Matters" by Metallica.

number of clusters that can be found with respect to a test corpus. For every test run, an average of 12 clusters could be found. Optimal clustering performance was provided by setting the number of neurons to approximately 30% of the number of files in the test corpus. As it was expected, a very large number of clusters was found in the large test corpus. However, the number of found clusters did not exceed 25% of the number of files in the corpus. For no test run, a very large number of neurons (i.e. in cases in which the number of neurons exceeded the number by a multiple) influenced the clustering performance negatively.

Normalizing the feature patterns negatively influenced the clustering performance. In all test runs, only two clusters could be found – independent from the chosen target file. Optimal results were provided in test runs in which normalizing of the patterns did not take place. It is to note that the extracted feature vectors have a very wide scope, i.e. some features have a very small absolute value, while others have a very large absolute value. Filtering the feature patterns in every test run did not influence the clustering performance negatively. In most cases, the clusters found with the filtering option activated were the same as the similarity clusters without filtering having taken place. Listing 2 displays the generic feature set. Filtered values (i.e. those that are unimportant for a classification task) are marked by an italic font. It can be seen that the majority of extracted values are frequency spectrum features and rhythm features.

Generally speaking, applying the SOM on the features extracted from the small corpora resulted in two or three clusters that are found amongst the input files. Although filtering did not influence any of the other test corpora, in the first small corpus, finding similar files was highly dependent on whether the feature patterns are filtered or not. Without filtering, the determined clusters only contained two files, both of which were rated to be very not similar to the target song (target song: “Paddy's Sicknote“ by “The Dubliners“, Songs in cluster: Marilyn Manson – “Tainted Love” and Frank Boeijen Groep – “Kronenburg Park”). Filtering the patterns caused 14 songs to be in the similarity cluster of the target song, none of which were rated to be similar. The user judged the similarity of “Paddy's Sicknote” and it became obvious that only “Yellow Submarine” by “The Beatles” was considered to be slightly similar to the target song (it received a ranking of one).

```

<feature name="Abs-Loudness">
<feature name="Branching">
  <feature name="Tempo">
    <feature name="Correlation">
      <feature name="TempoFinder">
        <feature name="CorrelationAverage">
      </feature>
    </feature>
  </feature>
<feature name="ExtremaDifference">
  <feature name="Characteristica_1">
  <feature name="Characteristica_2">
  <feature name="DisplacementDiff">
  <feature name="DifferenceAverage">
</feature>
</feature>
<feature name="ZeroCrossings">
  <feature name="ZeroCrossingFilter">
    <feature name="DisplacementDifferences">
      <feature name="ZeroCrossingsAverage">
    </feature>
  </feature>
  <feature name="PST_Angles">
  <feature name="PhaseSpaceTransformation_Angles">
  <feature name="Angles">
  <feature name="AngleAverage">
</feature>
</feature>
<feature name="PST_Distances">
  <feature name="PhaseSpaceTransformation_Distances">
  <feature name="Distances">
  <feature name="DistanceAverage">
</feature>
</feature>
<feature name="FFT_Complete">
  <feature name="Hanning_complete">
  <feature name="FFT_complete">
  <feature name="Frequency2Bark">
  <feature name="FrequencyPeaks">
  <feature name="LinearRegressionFunction">
  <feature name="SpectralFlatnessMeasure">
  <feature name="SpecGeomAverage">
  <feature name="SpecArithAverage">
</feature>
</feature>
<feature name="SpectralCrestFactor">
  <feature name="SpecMax">
  <feature name="SpecArithAvg">
</feature>
</feature>
<feature name="FFT_Windowing">
  <feature name="Windowing">
  <feature name="WindowingChain">
  <feature name="Hanning">
  <feature name="FFT">
  <feature name="Frequency2Bark (2) ">
  <feature name="MaxFrequency">
</feature>
</feature>
<feature name="FrequencyWindowAverage">
<feature name="KMeansIntervals">
<feature name="Interval2SingleValues">

```

Listing 2. A generic generic feature set.

Repeating the test runs on this small corpus with another query file did not reproduce these results. Instead, the results were comparable to the results of test runs on the other corpora.

The classification provided good results in structuring the input space into similarity clusters. In all test corpora, similarity clusters could be determined with regard to a specific target file. In average, 23.7 similar files could be found in each cluster. These files were rated by a user to judge the subjective similarity between the files. The majority of the files in the similarity clusters were considered to be similar to the target file. In each similarity cluster, there was an average of 14.7 similar files. These files achieved at least a ranking of three on the scale between zero to five (0: no similarity; 1: little similarity; 2: medium to little similarity; 3: medium to high similarity; 4: high similarity; 5: very similar/close to equal). Hence, in average, 65.7% of the files in the retrieved similarity clusters are considered to be similar to the query file. Only a few files in the found similarity clusters were not considered similar by the user (average: 9 files, that is 34.3%); these files received a ranking of two or less.

More interestingly, most of the songs that were not enclosed in the similarity cluster of the query file (i.e. those songs that the system considers not similar to the query file) were not considered similar by the human user, too. Only 17% received a ranking of three or above and can be hence thought of as “forgotten” by the system.

Another important finding is the fact that the accuracy of the classification was very high in the large corpus. For instance, there were 24 songs in the similarity cluster for the song “Nothing else Matters” by Metallica, 75% of which received a ranking of at least three (18 files), and 54% received a ranking of four or higher (13 files). However, it is to note that a lot of songs that were present in this song's similarity cluster during other test runs (on one of the medium test corpora) were not included in the similarity cluster in the test run on the large corpus.

Chapter 6

Discussion

Although the clustering accuracy of the suggested system is not as high as initially desired, classifying music files into similarity clusters is a good way to structure the search space (i.e. a very large music corpus). The suggested system presents itself as a capable tool to help the user to find music pieces according to the personal taste from large digital music databases. Nevertheless, there are a few things to note about the results from the previous section.

Applying a Genetic Programming approach on the task to find the ideal features to extract from a directory of music files in order to cluster the files into their similarities did not turn out to be a good strategy in accomplishing the task of feature extraction. This can be seen from the fact that normalizing the feature vectors results in distorted clustering performance. In addition, the individual-size seems to correlate with the size of the test corpus. The smaller the test corpus was, the shorter was the fittest individual for that corpus. Due to the fact that extracting a generic feature set from every test corpus resulted in a good clustering performance, evolutionary programming can hence be excluded from the suggested framework without losing applicability and validity on different test corpora. The reduction of runtime and memory complexity that can be expected is another argument in favor of excluding genetic algorithms from the approach at hand.

Filtering the feature patterns in every test run did generally not influence the clustering performance, which proves that features with a very small absolute value do not influence the classification and only play a minor role. These features can hence be omitted and do not need to be extracted from the source files. Therefore, the generic feature set can be designed more stream-lined, i.e. specifically for the task of feature extraction for similarity clustering using a Self-Organizing Map.

The clustering performance is directly connected with the amount of files in the underlying directory. The more files are in the source directory, the higher the clustering performance, regarding the number of retrieved clusters. This, of course, is not surprising since common sense suggests that the more songs are included, the higher the diversity will be among those songs. However, it is to note that the number of files falsely included into the similarity cluster of a target song is much lower in larger corpora than in smaller corpora. This can be explained with the fact that it is more likely to find truly similar files with regard to a certain target song in a larger corpus than in a smaller one. In the smaller corpus, the matches that are just “relatively” similar are included because these files are more similar to the target song than to any other song in the corpus – there are just no pieces that are more similar to the target song. However, much more similar songs might be present in the larger corpus. This also explains, why music pieces that were considered similar in a smaller cluster and hence included in a similarity cluster of a certain song are not included in the similarity cluster of the same song in a larger corpus (see the example of the song “Nothing else Matters” in the previous section). Songs, previously considered similar are simply more similar to another song than to the target song in the larger corpus.

Recapitulating it can be said that the very challenging task to structure large music databases in perceptually similar clusters and hence helping the user to find similar files more easily can be accomplished by the system at hand. The Self-Organizing Map Plug-in for the Machine Learning Environment Yale provided good results in finding similarity clusters on different test corpora and is a competent framework to structure digital music databases automatically. The system however can not only be used to help a user finding more music according to his/her personal taste, but it can be used by owners of very large music libraries to manage their databases. This might help in designing offers for customers of music online stores or Internet radio services. It is also imaginable that music pieces can be compared on a topological level, for example by music historians, to visualize musical correlation and similarity.

Chapter 7

Future Work

The present research presents the first steps to develop a system that can automatically structure music databases on a perceptual level. Hence, of course, a lot of work still needs to be done.

Since the random sample of the user judgment was very small in this study, more thorough user surveys need to be conducted with regard to the similarity judgment in successive research to answer the question if the presented system is applicable in wider user communities.

Future work will address the design of a more streamlined feature extraction method to provide a feature set that allows for more accurate classification tasks. In addition, the presented system can be tested on a larger variety of music files. The system has mainly been tested on fairly modern and recent music but should be tested on older music and/or classic music. Future research should answer the question of the importance of corpus pre-structuring as well as genre distribution among the test files.

Another interesting idea is to develop a standalone system out of the present plug-in. A piece of software is desirable that works independently from Yale and is able to provide home users with an intelligent structuring of their personal music archives. It is imaginable to include the possibility to share extracted features and store them in a centralized server, which will provide the opportunity to search remote databases for similar songs.

Connecting online music stores with that database can make it possible for the users to quickly obtain legal copies of music files that are similar to one's private collection for a minimum of financial costs.

Chapter 8

References

- [Allamanche2001] *Allamanche, E., Herre, J., Hellmuth, O., Fröba, B., Kastner, T. & Cremer, M.* **Content-based identification of audio material using MPEG-7 low level description.** In J. S. Downie and D. Bainbridge (Eds.), Proceedings of the Second Annual International Symposium on Music Information Retrieval (ISMIR). Pp 197-204, 2001.
- [Fischer2002] *Fischer, S., Klinkenberg, R., Mierswa, I., and Ritthoff, O.* **YALE: Yet Another Learning Environment - Tutorial.** No. CI-136/02, Collaborative Research Center 531, University of Dortmund, Dortmund, Germany, 2002.
- [Hacker2000] *Hacker, S.* **MP3, the definitive guide.** O'Reilly & Associates, Inc., 2000.
- [Kohonwn1982] *Kohonen, T.* **Self-Organized Formation of Topologically Correct Feature Maps.** Biological Cybernetics 43. pp 59-69, 1982.
- [Kohonen2001] *Kohonen, T.* **Self-Organizing Maps.** Springer-Verlag, New York, Inc., 2001.
- [Knuth2001] *Kurth, F., and Clausen, M.* **Full-Text Indexing of Very Large Audio Data Bases.** Presented at the 110th Audio Engineering Society Convention, 2001.
- [Koza1997] *Koza, J. R.* **Genetic Programming.** Encyclopedia for Computer Science and Technology. 1997.

- [Liu1998] *Liu, Z., Wang, Yao., Chen, T.* **Audio Feature Extraction and Analysis for Scene Segmentation and Classification.** Journal of VLSI Signal Processing 20, pp 61-79, 1998.
- [Mierswa2006] *Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., and Euler, T.* **YALE: Rapid Prototyping for Complex Data Mining Tasks.** In Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2006), ACM Press, 2006.
- [Mierswa2005] *Mierswa, I., and Morik, K.* **Automatic Feature Extraction for Classifying Audio Data.** In Machine Learning Journal, Vol. 58. pp 127-149, 2005.
- [MierswaV3.3] *Mierswa, I.* **Value Series Processing with Yale.** Version 3.3.
- [Pachet2001] *Pachet, F., and Laigre, D.* **A Naturalist Approach to Music File Name Analysis.** In Proceedings of 2nd International Symposium on Music Information Retrieval, 2001.
- [Pardo2004] *Pardo, B., Shlfrin, J., Birmingham, W.* **Name That Tune: A Pilot Study in Finding a Melody From a Sung Query.** Journal of the American Society for Information Science and Technology 55(4). 2004
- [Schedl2004] *Schedl, M., Pampalk, E., Widmer, G.* **Intelligent Structuring and Exploration of Digital Music Collections.** Austrian Research Institute for Artificial Intelligence (ÖFAI), Vienna, Austria and Department of Computational Perception Johannes Kepler Universität (JKU) Linz, Austria. 2004
- [Tzanetakis2001] *Tzanetakis, G., Essl, G., and Cook, P.* **Automatic Musical Genre Classification Of Audio Signals.** In Proceedings of the Int. Symposium on Music Information Retrieval (ISMIR). Pp 205-210, 2001.

[Uitdenbogers2004] *Uitdenbogerd, A. L., Zobel, J. An Architecture for Effective Music Information Retrieval.* Journal of the American Society for Information Science and Technology 55(12), pp 1053-1057, 2004.

[Unal2004] *Unal, E., Narayanan, S. S., Chew E. A Statistical Approach to Retrieval under User-dependent Uncertainty in Query-by-Humming Systems.* International Multimedia Conference, Proceedings of the 6th ACM SIGMM international workshop on Multimedia information retrieval. pp 113-118, 2004.