**University of Alberta**

# Using Multiple Detectors for Artist Classification

by

**Qiongyun Zhang** ©

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science.**

Department of Computing Science

Edmonton, Alberta
Spring 2005

0-494-08181-3

Library and
Archives Canada

Published Heritage
Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et
Archives Canada

Direction du
Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

*Your file Votre référence*
*ISBN:*
*Our file Notre reterence*
*ISBN:*

# Abstract

The digitization of audio media has brought with it the questions of how to store and catalog it, what to do with the information and how to protect its authorization. Music Information Retrieval is concerned with using retrieval techniques to perform searches on music media. An important problem in music information retrieval is the classification of music. Automatic classification at a coarse level, such as distinguishing among classical, rock and jazz music, is not a difficult problem, but more fine-grained distinctions among musical pieces sharing similar characteristics are more difficult to establish. In the realm of Music Information Retrieval (MIR), there is a burgeoning interest in automatic song and artist identification from audio data. Such work would obviously be useful for anyone who wants to ascertain the performing or composing artists of a new piece of music. It could be also used for automatic music categorization, for a digital library database or the data management for Karaoke according to artists. It could aid preference-based search and recommendations for music. Another area where artist identification may benefit is copyright protection and enforcement. The artist identification problem can be divided into two parts: how to select promising audio features and what kind of machine learning techniques are suitable for the specific task. We implement a prototype system which can analyze features and predict the performing artist of a music piece using machine learning algorithms. We experiment with features and efficient 1-class learning techniques to build customized classifiers for the specific task of artist identification.

# Acknowledgements

I would like to thank:

- My parents for supporting me all the time through my ups and downs.
- My supervisors, Dr. Randy Goebel and Dr. Robert Holte for their understanding, continuous support and helpful guidence in my research.
- Dr. Michael Frishkopf for being my external examiner and helpful feedback.
- All other fellow students who shared the great time in my graduate life.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

This dissertation describes a machine learning approach to the problem of identifying a performing artist in a piece of music.

## 1.1 Introduction

The digitization of audio media has brought with it the problem of what to do with the information; how to store it, catalog it, and make sure that only those authorized can inspect it. In the realm of Music Information Retrieval and Music Digital Library research communities, there is growing interest and work on automatic song and artist identification from audio data. Such systems would be useful for anyone who wants to ascertain the performing or composing artists of a piece of music. It could also be used for automatic music categorization, for digital library database or the data management for Karaoke according to artists. It could also aid preference-based search and recommendation for music. Another area where artist identification may benefit is copyright protection and enforcement [BEL02]. Depending on requirements for an application one wants to develop, the identification can be based on either a whole piece of music, or a short segment. For example, a system that works like a radio scanner to find specific artist's music is supposed to be able to identify the artist, given a fairly short segment. For other applications such as automatic CD categorization, etc, the identification can be on the whole piece level.

Analogous to human listening procedure (Figure 1.1), an artist identification listening system generally has three parts: signal processing, learning and classifying. With a little training, if a human is familiar with a particular artist's voice and style, one can usually recognize that artist's work, even when hearing a song from the first time. The problem of artist identification is to automatically establish the identity of an artist using audio features extracted from songs in a

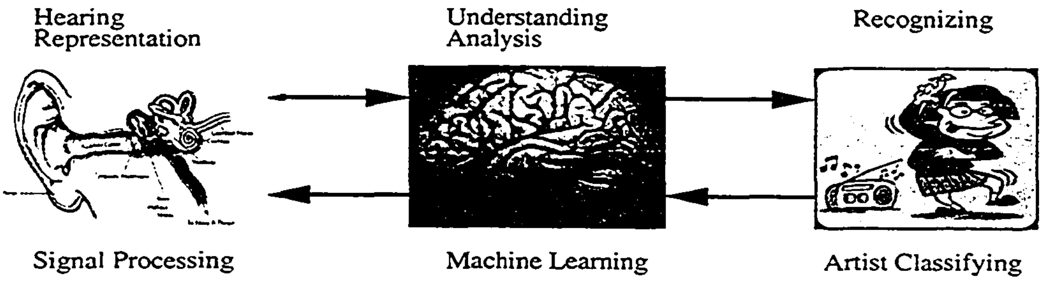


Figure 1.1: Machine Listening Procedure

dataset of music pieces. It is a typical classification problem that arranges objects into categories or classes according to established criteria. For artist identification, each artist is considered as a class. It must be noted that "artist" is in ambiguous, since the artist a music piece features could be the composer, the performers, the instrument players, the producer, and so on. On the other hand, in some cases, like in rock and pop music, the performer, singer and the composer is often the same. In our work, we treat the performing musician as the artist.

## 1.2 Research Goal

The goal of the research described in the dissertation is to build an artist identification system using machine learning techniques, in order to perform classification among various artists that the system has been trained to recognize. Considering the vast space of existing artists, it is not practical to build a system that can handle all artists. Therefore our system is designed to be able to distinguish unknown artists from the ones it knows as well.

## 1.3 Contributions

Our contributions include the design, implementation, and experimentation with a new artist classification system. Our design is based on using four different machine learning methods to create a collection of 1-class "detectors", and then combining this collection of 1-class detectors using a meta-learning approach, to create a complete classifier. Our approach is different than others in that we build the classifer incrementally, by training a 1-class detector for each class and then combining them linearly. Designed in this way, our classifier can handle the problem of lack of training data from unknown classes, and it is also possible to be further developed to reject unknown artists.

## 1.4 Summary

This chapter has introduced and motivated the artist classification problem in music information retrieval. It has also stated the goal of our research.

Chapter 2 provides the background knowledge of machine learning and artist identification. It introduces challenges of feature extraction and selection, the concept of 1-class classification and multi-class classification, classifier building, and the overall strategy of classification process and classifier evaluation. Following that, the current state of art of the artist identification problem is described. Chapter 3 discusses our strategies for extracting features from audio data, to be used for our machine learning algorithms. In Chapter 4, several machine learning algorithms and our customized classifier combination technique are introduced. Chapter 5 gives a detailed description of the measurements we use to evaluate classifiers. Chapter 6 describes the design of our experiments, and presents and analyzes the results. Chapter 7 provides some concluding remarks and discussion on limitations as well as ideas for future work.

# Chapter 2

# Background on Machine Learning and Artist Identification

This chapter describes the basic concepts of machine learning including feature extraction and selection, how to build classifiers using these features, how a constructed classifier is used to identify new instances and the evaluation of classifier performance. More specifically, we explain the difference between 1-Class classification and Multi-Class classification. Following the introduction of the background knowledge of machine learning, the current state of the art of artist identification is described in term of features and learning algorithms used in previous work.

## 2.1 Features

The basic building blocks of any machine learning system are a collection of features about things we would like to learn to make predictions about. Features are also called attributes or properties. A collection of features with their values forms a dataset in which each item describes an instance or example. Table 2.1 is an example of feature-based data.

| **Hair** | **Height** | **Weight** | **Lotion** | **Result** |
|---|---|---|---|---|
| blond | average | light | no | sunburned |
| blond | tall | average | yes | none |
| brown | short | average | yes | none |
| brown | short | average | no | sunburned |
| red | average | heavy | no | sunburned |
| brown | tall | heavy | no | none |
| brown | short | light | yes | none |

Table 2.1: An example of feature-based data

One learning task is to find a classifier or function that generalizes from the training data so that the classifier is able to predict the class for a new instance. As a fundamental unit of machine learning, features describe the data and influence the nature of learning algorithms that can be used. Figure 2.1 shows the diagram of the transformation from data to features and then to knowledge discovered by learning algorithms.

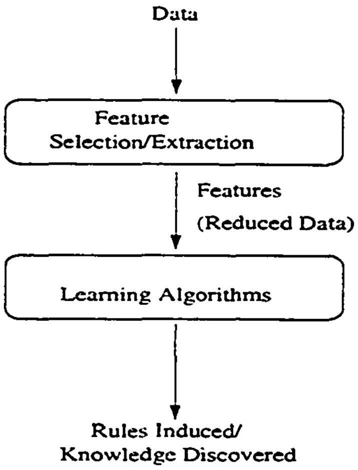

Data
Feature Selection/Extraction
Features (Reduced Data)
Learning Algorithms
Rules Induced/ Knowledge Discovered

Figure 2.1: A data flow diagram: relations between data, features and knowledge discovery

While machine learning techniques such as classification attempt to summarize data into a concise form that can be understood and reused, appropriate feature selection is required to make the learning more efficient and effective [LM98]. Robust and relevant features can benefit learning by providing an accurate description of the data. On the other hand, redundant and irrelevant features can degrade learning and the quality of the classifier. Therefore, the selection of features is a key to learning and the quality of a classifier. The features extracted from the audio data in our work and feature selection strategies will be described in detail in Chapter 3.

## 2.2 Classification

In general, the process of classification is completed in three major steps: learning a classifier from training data, classifying new unknown data using the classifier, and measuring the classifier's performance. A learning algorithm defines the actual process to build a classifier model from training data and its performance on testing data is then measured by certain evaluation criteria to determine how close the model is to the data.

### 2.2.1 Building and Using a Classifier

In general, building and using a classifier involves a two-step process, as shown in Figures 2.2 and 2.3: training (learning) and identification (classification). In the training step, a classifier is built using a machine learning algorithm that analyzes a set of training data, each labeled with the name of its class. In the prediction phase, the classifier built in the previous step is used to predict the class of unknown examples.

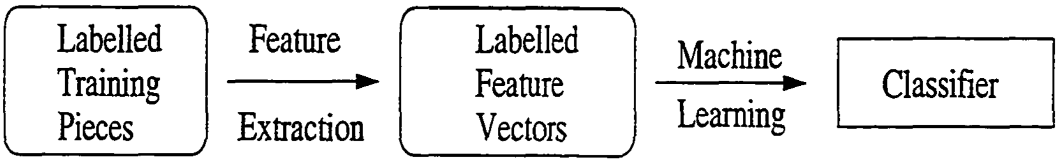


Figure 2.2: Building a classifier using machine learning algorithm

A common technique for building a classifier is to apply a machine learning algorithm to a set of labelled training examples. Each training example has a set of features and a class label. The values of features in each training item (in our case, double-precision floating point values) along with the label (artist name) are used to build a classifier. More specifically, each of our training examples has features derived from a segment of a complete performance and the name of its singer. For each audio piece, we extract a set of features via a signal processing program. Chapter 3 gives a detailed description of the feature extraction strategies.

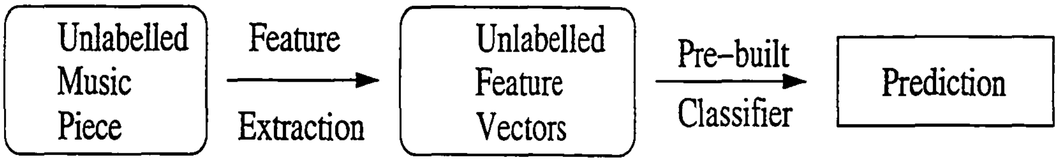


Figure 2.3: Identifying the artist using pre-built classifier

Once the classifier is built, it can take a piece of audio data with an unspecified artist and uses the feature values extracted from the data to identify the artist. Figure 2.3 shows the procedure.

### 2.2.2 1-Class Classification and $n$-Class Classification

A classification task is typically viewed as placing an object in the appropriate class, from a choice of $n$ classes. Usually by learning from training data, a classifier is able to recognize examples from all the classes it has seen in training and then categorize new examples in testing. A classifier can be either generative or discriminative. Generative classifiers learn a model of the conditional probability $p(x|y)$ of the inputs $x$ and label $y$, and make their predictions by using Bayes rules [Mit97] to calculate $p(y|x)$ for each class label $y$, and then picking the most likely label $y$ [NJ02]. *Naive Bayes* [Mit97] is a typical generative learning method. Discriminative classifiers model the posterior probability $p(y|x)$ directly, or learn a direct map from inputs $x$ to the class labels. Discrinimative methods attempt to maximize the discriminability between classes without estimating class-conditional densities $p(x|y)$ or assuming anything about the input distribution $p(x)$. A good example of a discriminative classifier is a Support Vector Machine (SVM) [SPSTS01]. A widely held belief is that discriminative classifiers are almost always to be preferred to generative classifiers [NJ02], and a compelling reason is that one should solve the classification problem directly and never solve a more general problem as an intermediate step (such as modeling $p(x|y)$) [NJ02].

1-class classification tries to identify a class of target data and to distinguish it from all other possible classes. In the training step of a 1-class classifier, only positive examples of a class are fed to the learning algorithm. In the testing step, examples from that specific class are considered as positive and are supposed to be recognized by the classifier, while examples from any other classes are supposed to be classified as negative. 1-class classification has been suitable for applications where there is only information available regarding one class for training [HGR04]. For example, to build a classifier to verify authenticity of a bank cheque, there is usually only examples from legal customers for training as it is difficult to acquire sufficient counterfeit examples. Therefore the learning task can be viewed as a 1-class classification problem. It is also practical to build an ensemble of 1-class classification which can be combined to perform multi-class classification.

There are some 1-class classification methods existing in the field of machine learning, such as 1-class SVM [SPSTS01], and Auto-encoder Neural Network [JMG95]. Many other machine learning algorithms can be customized to perform 1-class classification as well. For example, by using training data from only one class and setting a distance threshold, a 1-class $k$-Nearest Neighbors classifier can be easily constructed. Likewise, a 1-class Naive Bayes classifier can be built with slight modification.

In our work, we build a 1-class classifier for each individual artist in the training data, using both existing and customized 1-class learning methods. The details about these learning methods will be presented in Chapter 4.

The ultimate goal of our work is to build an $n$-class classifier that can identify a set of different artists. However when there are a considerable number of classes in a learning task, the performance of an $n$-class classifier can degrade because of the large number of classes. An alternative is to build an $n$-class classifier by making use of an ensemble of different classifiers. For example, 1-class classifiers can be combined to do $n$-class classification based on the decision of each individual classifier.

In our system, we combine four different machine learning methods in various ways to provide the basis for our $n$-class classification experiments. The combination methodology will be discussed in Chapter 4.

### 2.2.3 Evaluation

To evaluate the performance of classifiers, it is important to define an appropriate evaluation criteria. We will use a standard machine learning technique called *cross validation* [Mit97]. Each set of labelled training instances is partitioned into $n$ groups of training instances $G_1 ... G_n$, each of them having approximately the same number of training instances. Then $n$ different classifiers are constructed $C_1 ... C_n$, where $C_i$ uses all of the groups as training instances except $G_i$. Following that, a *confusion matrix* is computed for each of the $n$ classifiers, $C_i$ using $G_i$ as testing data. A confusion matrix [Mit97] is simply a table that records the classification results of a classifier. Table 2.2 gives an example of a $2 * 2$ confusion matrix in which there are two classes of data, positive and negative. The number in row $i$ and column $j$ denotes the number of examples from class $i$ being classified as class $j$. In the example shown in Table 2.2, there are 100 positive examples and 100 negative examples in the testing set. The classifier recognizes 90 positive examples out of 100, and incorrectly classifies the other 10 positive examples; it also correctly classifies 80 negative examples, while misclassifying the remaining 20 negative examples.

| **Predicted⇒**<br>**Actual⇓** | **Positive** | **Negative** |
|---|---|---|
| **Positive** | 90 | 10 |
| **Negative** | 20 | 80 |

Table 2.2: A 2*2 Confusion Matrix

Various statistics [vR79] can be computed from the confusion matrix to evaluate the performance of a classifier. Based on these statistics, ROC Curves [Mar03] and Cost Curves [DH00] can be constructed to evaluate our classifiers. Further details on the concepts of these evaluation tools will be presented in Chapter 5.

## 2.3 Related Work

A number of systems have been developed over the past years that support classification of artists using various prediction algorithms, based on different features. All these works slice a piece of music into small frames, extract features from each of these short frames and use these feature vectors as the data for training and testing. The classification results for all the frames in a piece vote together to decide the classification for the complete piece. Therefore, there are two accuracies, at the frame level and at the entire song level. Four typical systems related to our work are as follows:

- An singer identification system Minnow-match is developed and described in [WFL01]. The dataset for this work is a 21 artists dataset, which includes data for 16 vocal singers and 5 non-vocal artists. In this work, the author divides each piece into short slices and then automatically extracts various features and makes classifications using a neural network and an SVM. Features are computed using Fast Fourier Transform (FFT) [Coo87] and Mel Frequency Cepstral Coefficient (MFCC) [Log00] on each of these short segments. The classifier is trained by a multi-layer ANN and SVM respectively. To solve the scaling problem of neural networks, a third approach is to combine the SVM with the neural network by feeding the resulting output of each SVM to a new neural network. The system classifies in a one-in-$n$ singer space correctly around 50% for 21 singers at the entire song level.

- For the same task, Adam et al. use singing segments instead of using the entire track for singer classification [BEL02]. A multi-layer neural network classifier is trained, using Linear Prediction Coefficient (LPC) [Har01] for the task of locating segments of music that are dominated by singing voice. The classification is also performed by a neural network classifier. The inputs to the neural network are MFCCs calculated from each short time segment. Experimenting with the same dataset as in [WFL01], the classification is reported to be improved by about 15% compared with [WFL01] and comes to 65% for 21 singers at entire song level. Also the classification performance on 16 singers increases to 68%. At the segment level, the best resulted reported is about 36% for 16 singers and 32% for 21 singers.

- In another system by Kim and Whitman [KW02], voice coding features are computed by LPC instead of FFT and MFCC. Gaussian Mixture Model (GMM) [DHS00] and SVM classifiers are built to identify different singers. The author selected a subset of the dataset used in [WFL01] which contains the data for the 16 vocal artists. They added data for another singing artist, making a total of 17 artists in their new dataset. The best results of song classification using their approach is 41.5% for 17 vocal artists and a frame accuracy of 31%.

- The fourth application on artist classification is different than the aforesaid three in that this system learns to recognize six famous pianists playing the same set of piano concert pieces based on their performing style, instead of singers. The system extracts a number of low-level features related to tempo and loudness from the original audio CD recording by these pianists, and applies 6 machine learning algorithms to the task of learning classifiers based on these features. These learning methods include Naive Bayes, k-Nearest Neighbors (kNN), Decision Tree, Classification via Regression, Logistic Regression and a rule learning algorithm [WF00]. Instead of doing $n$-class classification directly, the author converted it into $n(n-1)/2$ pairwise discrimination problems. The overall prediction accuracy is around 70%.

| **Authors** | **Data Pre-processing** | **Use Voice Segment** | **Features** | **Learner** |
|---|---|---|---|---|
| Whitman et al. | 1 out of every k slices of 0.8-1s is taken as an example. | No | MFCC, FFT | ANN, SVM, and SVM+ANN |
| Kim et al. | Example frames are of 1024 samples ( 100msec) taken every 512 samples ( 50msec) | Yes | LPC | GMM, SVM |
| Berenzweig et al. | Example frames are of 32 msec long,<br>taken every 16msec. | Yes | LPC for vocal segmentation;<br>MFCC for singer classification | Multi-layer neural network |
| Widmer et al. | Example frames are of the length of two bars | N/A | tempo and loudness features | Naive Bayes, kNN, Decision Tree, Rule Learner, Classification via Regression and Logistic Regression. |

Table 2.3: Approaches Used in Different Artist Identification Systems

Table 2.3 lists the different approaches used in the four previous research works. Table 2.4 lists and compares the best classification results obtained by the three singer identification systems since they are tested on the same dataset.

| Authors | 16 artists (frame) | 16 artists (song) | 21-artists (frame) | 21 artists (song) |
|---|---|---|---|---|
| Whitman et. al | N/A | N/A | N/A | .50 |
| Kim et. al | .307 | .415 | N/A | N/A |
| Berenzweig et. al | .356 | .683 | .318 | .649 |

Table 2.4: Comparison of Classification Accuracy on 21-Artists Dataset

As revealed in the related work, the singer identification systems depend on the voice features in music more than features from other various aspects of music. In terms of classification accuracy, the performance of these classifiers, especially at frame level, is still relatively low [KW02]. Therefore, in our work, we use a different and wider repertoire of features. Also, instead of using existing machine learning techniques to build an $n$-class classifier, we build a custom $n$-class classifier using different 1-class machine learning algorithms. By using these different features and learning approaches, we hope for some improvement on the artist identification problem.

## 2.4 Summary

In this chapter, background knowledge on machine learning related to our work was briefly introduced, including the major steps of learning and classification, selection and the importance of features, classifier construction and evaluation. In particular, 1-class and n-class classification were discussed. Also four different artist classification systems in the literature were described and compared regarding the features and learning methods used.

# Chapter 3

# Feature Extraction

As previously introduced, audio features are the heart of the process of both classifier building and artist identification. This section introduces some basic elements of music and the strategies we used to extract audio features corresponding to these music elements.

## 3.1 Basic Elements of music

Music contains complex structures of many interrelated elements. Generally speaking, elements that can be measured directly using scientific instruments comprise the ***physical properties*** of music. On the other hand, those that a human listener associates with the music sound are the *perceptual attributes* [Sch00]. For some perceived attributes of sounds, it is relatively easy to understand the physical correlates of the attributes. For example, as the frequency of a sine tone varies, the pitch of the sound varies in a simple ways. However, for other sounds and properties, the correlation between the physical and the perceptual can be difficult to understand.

### 3.1.1 Pitch

In a piece of western music, multiple notes group sequentially in time into melodies, simultaneously into chords, and in both directions into structures such as "harmonies" and "key". Each music note, when turned into sound, is perceived to have a pitch. The pitch of a sound is an attribute which may be positioned on a continuum from "low" to "high".

By definition, the pitch of a sound means the frequency of a fixed-amplitude sine tone that most closely matches the sound [Sch00]. But this does not mean that sounds have a "correct" pitch that listeners can be asked to "identify". The pitch of a sound is simply what a particular listener, in a particular context, feels that it is. The physical correlate of pitch is *frequency*. As the frequency of a sound changes, the pitch of the sound is perceived to change [Sch00].

### 3.1.2 Loudness

The loudness of a sound is a perceptual attribute that can be felt as "quiet" or "loud". Loudness can be defined operationally as the amount of energy in a sine tone. Loudness is typically correlated with the physical property *intensity*. The relationship between loudness and intensity is similar to that between pitch and frequency. Pitch and loudness are the two most well-studied attributes of sounds [Sch00].

### 3.1.3 Timbre

In music, timbre is the quality of a musical note which distinguishes sound of identical pitch and loudness. For example, consider a note of the same pitch and loudness played on a piano and a flute. While they have the same frequency, the human ear hears them as different, and with a little practice can identify the instruments involved. However, there is no simple set of physical properties that correspond to timbre, and there is no clear operational definition [Sch00]. Timbre can be roughly defined as all those qualities of a sound that allow a listener to distinguish sound of same pitch and loudness. These qualities might include:

- Spectrum: the aggregate of simpler waveforms (usually sine waves) that make up what we recognize as a particular sound. This is what Fourier analysis gives us [BPR+].
- Envelope: the attack, sustain, and decay portions of a sound (often referred to as transients) [BPR+].

Envelope and spectra are complex concepts, and include a lot of sub-categories. For example, spectral features are very important, as there are many different ways that the spectral aggregates can be organized statistically, in terms of shape and form. For example the relative "noisiness" of a sound is a result, in large part, of its spectral relationships. Facets of envelope (onset time, harmonic decay, spectral evolution, steady-state modulations) are not simply explained by just looking at the envelope of a sound [BPR+].

### 3.1.4 Tempo

Tempo identifies the rate of the beat of the music, and is measured by the number of beats per minute. The tempo of music is a perceptual sense that the sound is recurrent in time at regular intervals. A good operational definition of tempo would be the frequency of a click-track adjusted to have the same perceived speed as the stimulus. Like pitch and loudness, tempo is a perceptual attribute that cannot be measured directly from a sound. A sound does not have a *real tempo* that a listener might judge "incorrectly". The tempo of a sound to a listener is just whatever the listener thinks it is. Even though good musicians can agree on timing and tempo for the performance of music, it is not presently well-understood what the physical correlate of tempo is [Sch00].

### 3.1.5 Rhythm

Rhythm as a musical concept is intuitively easy to understand, but somewhat difficult to define. There is no ground truth for rhythm to be found in simple measurements of an acoustic signal. This places rhythm and tempo into the domain of perceptual attributes of sound.

## 3.2 Dataset Choices

In our work we use two different datasets, one is the 21 vocal and non-vocal artists dataset as used in [WFL01], [KW02] and [BEL02]; the other is a dataset made up of performance features of 6 famous piano players [WZ04].

In the 21-artists dataset each artist has around 10 MPEG 1 Layer 3 (MP3) files. While using MP3 files as our source data, we need to decompress them first. A whole MP3 audio file is sliced into non-overlapping 30 second chunks. Each of these chunks is decompressed into a 22050Hz, 16-bit mono audio file, to prepare for later extraction of audio features. When extracting features, these audio chunks will be further sliced into windows, each of which will have a statistical processing program applied, such as the *Fast Fourier Transform*. The feature extraction strategy will be described in the next section. The segmentation procedure is shown in Figure 3.1. After the pre-processing, features extracted from those 30 second audio chunks make up the dataset used to train and test our system.

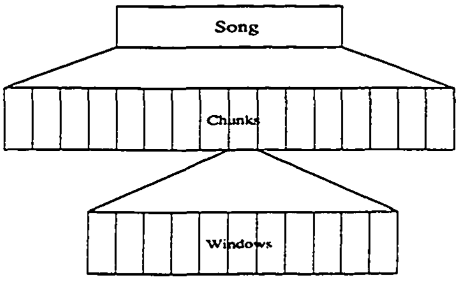


Figure 3.1: Song, Chunks and Windows

In the 6-pianist dataset, there are 12 piano sonata movements, and every player has recordings on all of the 12 pieces, which results in 72 recordings. Each piece is sliced into windows of two bars; each window has a set of features related to tempo and loudness. Thus, the raw data with this dataset for our experiments is tempo and overall loudness values measured for windows. Altogether, this procedure produces about 23,000 instances for the 6 players [WZ04].

## 3.3 Feature Extraction Strategy

The selection of descriptive features for a specific application is one of the main challenges in building machine learning systems. Based on available digital processing techniques and previous research work, there are a wide variety of audio features that can be extracted from an audio file. These correspond to several major music elements—timbre, loudness, beat/rhythm, pitch. Due to the availability of efficient digital signal processing software, we make use of an audio data processing program called Marsyas [Tza00] to extract features for the 21-artists dataset. The features extracted by Marsyas are presented below. For the pianist dataset, as we are provided with the dataset which can be used directly, we need no extra work on feature extraction. The feature extraction strategy used in [WZ04] for the dataset will be briefly described as well in section 3.3.4.

### 3.3.1 Timbre Texture Features

Features to represent timbre texture are calculated from Short Time Fourier Transform (STFT) [Roy] that can be efficiently calculated using the FFT algorithm and MFCCs.

Fourier transform, in essence, decomposes a waveform or function into sinusoids of different frequency which sum to the original waveform. It identifies the different frequency sinusoids and their respective amplitudes [Wei]. FFT is an efficient algorithm to compute the discrete Fourier transform (DFT) and its inverse. It transforms a signal between the time domain and the frequency domain. Figure 3.2 shows the result of applying FFT on a simple sine wave. Adapted from the Fourier transform, STFT analyzes only a small section of the signal at a time using a technique called windowing the signal, and maps a signal into a two-dimensional function of time and frequency. In the discrete time case, the signal to be transformed is broken up into chunks, each chunk is Fourier transformed and the complex result is added to a matrix which records magnitude and phase for each point in time and frequency [Wei].

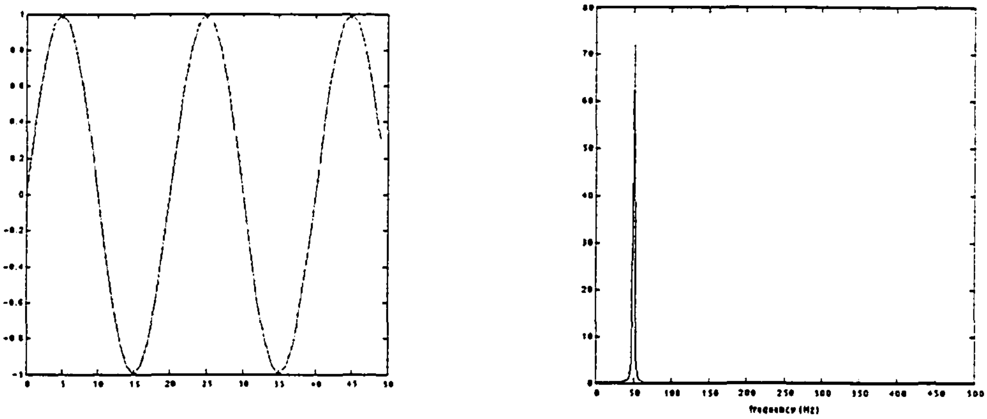


Figure 3.2: Fast Fourier Transform on a sine wave

Features calculated from STFT include spectral centroid, spectral roll-off, spectral flux, zero-crossing, MFCC and low-energy feature [Tza00].

- Spectral Centroid: it is the centroid of the short-time Fourier transform magnitude. The centroid is the balancing point of the spectrum—the frequency where the energy of all frequencies below that frequency is equal to the energy of all frequencies above that frequency. It is a measurement of "brightness". The brighter the music is, the higher the centroid value.
- Spectral Roll-off: it is defined as the frequency below which 85% of the Fourier transform magnitude distribution is concentrated.
- Spectral Flux: defined as the squared difference between the normalized magnitudes of successive spectral distribution. It is a measure of the amount of local spectral change.
- Zero-crossings: computed as the number of signal sign changes in a frame.
- Low-energy: the percentage of windows that have less *Root Mean Square* (RMS) energy than the average RMS energy across a frame. As it measures signal energy, it could be used for a measure of loudness.

MFCCs carry important information about the music's instrumentation and its timbres, the quality of a singer's voice, and production effects. MFCCs have also been used as dominant features

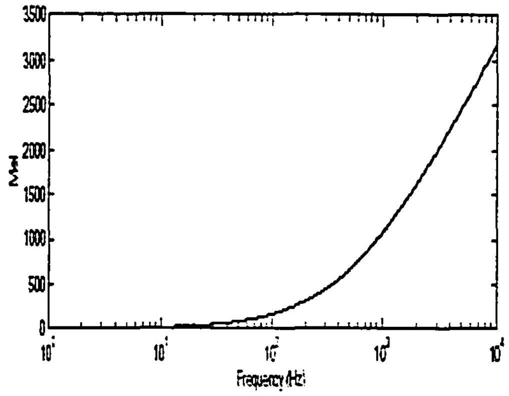


Figure 3.3: The Mel scale function [Log00].

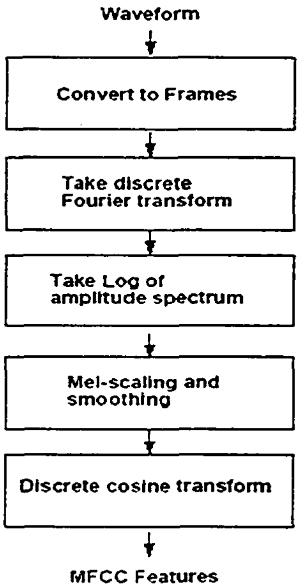

Waveform
Convert to Frames
Take discrete Fourier transform
Take Log of amplitude spectrum
Mel-scaling and smoothing
Discrete cosine transform
MFCC Features

Figure 3.4: Process to create MFCC features [Log00]

for speech recognition [Log00]. They are perceptually motivated features that are also based on Fourier transforms [TEC01]. After taking the log-amplitude of the magnitude spectrum, the FFT bins are grouped and smoothed according to Mel frequency scaling. The Mel scale is based on a mapping between actual frequency and perceived pitch as apparently the human auditory system does not perceive pitch in a linear manner [Log00]. The mapping is approximately linear below $1kHz$ and logarithmic above. Figure 3.3 shows the Mel function. Finally, a discrete cosine transform is applied [Log00]. Figure 3.4 shows the process of creating MFCC features.

### 3.3.2 Rhythmic Content Features

The calculation of features for representing the rhythmic structure of music is based on the *Wavelet Transform* (WT) and the feature set is based on detecting the most salient periodicities of the signal [Tza00]. A beat histogram is built by applying WT algorithms for an excerpt. Figure 3.5 is an example beat histogram.

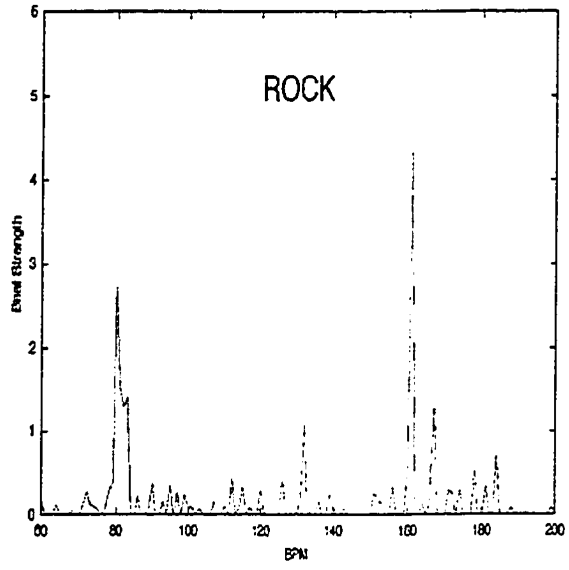


Figure 3.5: A Beat Histogram Example [TEC01].

A set of features are calculated in order to represent rhythmic content. These are:

- A0, A1: amplitude of the first and second histogram peak;
- RA: ratio of the amplitude of the second peak divided by the amplitude of the first peak;

- P1, P2: period of the first, second peak in bpm(beats-per-minute);
- SUM: overall sum of the histogram (indication of beat strength).

### 3.3.3 Pitch Content Features

The pitch content feature set is determined by multiple pitch detection techniques [Tza00]. Similar to rhythm content analysis, a pitch histogram is constructed first. Two versions of the pitch histogram are created: a *Folded Pitch Histogram* (FPH) and an *Unfolded Pitch Histogram* (UPH). Figure 3.6 shows the UPHs for two songs. The difference between FPH and UPH is that in the folded case, all musical notes are mapped to a single octave [Tza02]. The FPH carries information regarding the pitch classes or harmonic content of the music whereas the UPH contains information about the pitch range of the piece [Tza02].

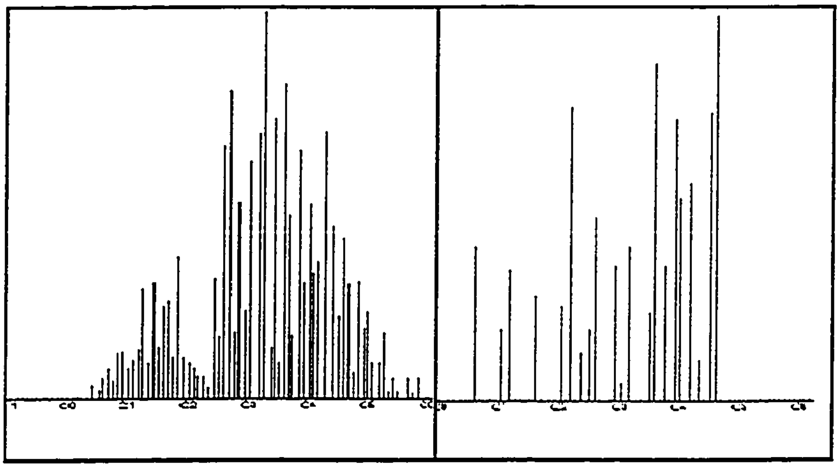


Figure 3.6: Pitch Histograms of a Jazz song (left) and an Irish folk song (right) [TEC01].

Based on pitch histograms, the following features are computed:

- FA0: amplitude of the maximum peak of the FPH. This corresponds to the most dominant pitch class of the song;
- UP0: period of the maximum peak of the UPH. This corresponds to the octave range of the dominant musical pitch of the song;
- FP0: period of the maximum peak of the FPH. This corresponds to the "main" pitch class of the song;

- IPO1: pitch interval between the two most prominent peaks of the folded histogram;
- SUM: overall sum of the histogram (indication of the strength of pitch detection).

### 3.3.4 Pianist Performance Features

Pianists do not play pieces of music mechanically, with constant tempo or loudness exactly as written in a music score. Rather, they speed up and slow down at different places, and stress certain notes and passages by various means. The most important parameter dimensions available to a pianist are timing and continuous tempo changes, dynamics (loudness variations), and articulation (the way successive notes are connected) [WZ04].

From the audio recordings, rough measurements characterizing the performances were obtained. Changes of tempo and general loudness are measured at the level of beats. From the varying time intervals between successive beats, the beat-level tempo changes can be derived. Overall loudness of the performance at these time points are extracted from the audio signal and taken as a very crude representation of the dynamics for that pianist [WZ04]. For each measured point, the following is stored: absolute time in seconds, local tempo in beats per minute (bpm) and loudness level measured in *sone*. These sequences of measurements can be presented as two sets of performance values, one presenting variations in beat-level tempo over time and the other beat-level loudness changes — or in an integrated 2D way, as trajectories over time [WZ04], as shown in Figure 3.7. Each sliced instance of two bars in length is a subsegment of the trajectory.

For each window $w_i$ of the original raw data, the following features are computed both for tempo and loudness, as described in [WZ04]:

- **average** tempo and loudness value within a window $\mu(w_i)$;
- **standard deviation** $\sigma(w_i)$;
- **range** of the value within the window $R(w_i) = max(w_i) - min(w_i)$;
- **normalized** features $\sigma'(w_i) = \sigma(w_i)/\mu(w_i)$;
- **correlations** between time and tempo;
- **directness of movement** measured by the ratio between the length of a direct movement between the end points of a segment and the length of the actual trajectory between the same points in a two-dimensional space.

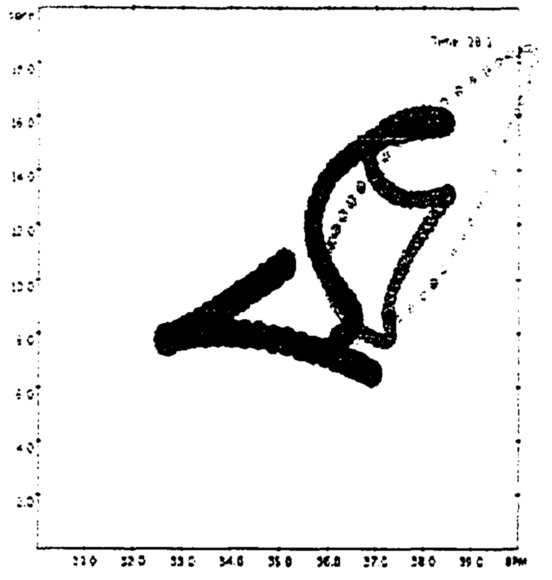


Figure 3.7: Horizontal axis: tempo in *bpm*; vertical axis: loudness in *sone*. Movement to the upper right indicates a speeding up and loudness increase. The darkest point represents the current instant, while instants further in the past appear fainter [WZ04].

- **derivatives** for tempo and loudness, including the maximum and the average of the absolute value of the derivative, and the normalized version.

## 3.4 Feature Normalization

Since our features are extracted using different techniques and focus on various aspect of music, such as human voice, tempo, pitch, etc., the features actually fall into various ranges: different features have values in different orders of magnitude. It is very import to scale features into similar range before applying machine learning algorithms such as Artificial Neural Network (ANN), SVM, etc [Sar97]. Therefore, we need to scale features to make them have approximately the same effect by independently normalizing each feature to the same range. The main advantage is to avoid attributes in greater numeric ranges dominating those in smaller numeric ranges. Another advantage is to avoid numerical difficulties during the calculation [HCL03].

The term "scaling" is used more or less interchangeably with the term "normalizing" and "standardizing", within various fields. Normalizing a feature vector often means dividing it by a norm of the vector; in the neural network literature, it also often refers to scaling by the minimum and range of the vector, to make all the elements lie between 0 and 1 [Sar97]. Standardizing a feature vector most often means subtracting a measure of location and dividing by a measure of scale. For example, if the vector contains random values with a Gaussian distribution, one might subtract the mean and divide by the standard deviation, thereby obtaining a standard normal random variable with mean 0 and standard deviation 1 [Sar97]. In our work, we choose to normalize the features into range of $[0, 1]$ for training data and use the minimum and range of features in training data to perform normalization on the testing data.

## 3.5 Summary

This chapter provided some basic music elements that are involved in the work and their corresponding physical or perceptual features. It also described the datasets we use and the feature extraction strategy applied to the source data to get features that are used to train classifiers.

# Chapter 4

# Machine Learning and Classification

The chapter describes the established and customized machine learning algorithms we use to build the artist classifier. These algorithms include ANN, kNN, SVM and Naive Bayes. All these algorithms are widely used in the machine learning literature. Their corresponding customized 1-class detectors Auto-encoder, 1-class kNN, 1-class SVM and 1-class Naive Bayes are also described in this chapter.

The vast space of artists makes the classification task tough for an artist identification system. On one side, the multi-class classification gets harder as the number of artists in the training dataset gets bigger. This raises the issue of classification scaling up. On the other side, it is impossible for an artist classifier to recognize all the artists around the world. Therefore we design our classifier to reject examples from unknown artists. This raises the issue of the lack of training data from the unknown artist space. To resolve these two issues, we choose to use 1-class learning methods to build detectors for each of the artists in the training dataset, then combine all the 1-class detectors to perform multi-class classification. By combining these 1-class detectors, the multi-class classifier is built in a cumulative manner, and examples from unknown artists can be detected when none of these 1-class detectors recognizes it. On the other hand, the lack of examples for unknown artists is no longer a problem because for training 1-class detectors, negative examples are not needed. However, we did not deeply explore nor experiment much on the idea of rejecting unknown artists. This will be our future work, as stated in Chapter 7. In order to combine the outputs from these 1-class detectors, we put a single layer neural network over the 1-class detectors. By training the neural network and setting thresholds for its outputs, when testing example comes, a class label will be decided at the neural network level.

The terms that will be used throughout this chapter are defined as following:

1. X: a training dataset of $m$ data points;
2. $(x_i, y_i)$: the $i^{th}$ instance in the dataset X; $x_i$ **is an** $n$**-dimensional feature vector,** $y_i$ **is the target value (class label);**
3. $f_j(x_i)$**: the** $j^{th}$ **feature of the** $i^{th}$ **instance;**
4. **C: a finite set where there are s values** $C_1, C_2 ... C_s$;
5. $C_k$**:** $k^{th}$ **target value of finite set C;**
6. $F : R^n \to C$**: the mapping function from feature space to target space;**
7. $\hat{y}_i$**: predicted class label for** $i^{th}$ **instance.**

## 4.1 Artificial Neural Network and Auto-encoder

Artificial neural networks provide a robust approach to approximating real-valued, discrete-valued and vector-valued target functions over continuous and discrete-valued attributes. The Backpropagation [Mit97] algorithm is the most common network learning method and has proven successful in many practical problems such as handwriting recognition, robot control, etc.

The networks are made up of nodes, called neurons, that are grouped into one or more layers (input, hidden and output), and connected via weighted edges, which are typically adjusted during training phase. A 3-layer ANN is shown in Figure 4.1. The weights of an ANN are learned during the training phase. The attributes of a training example are fed to the input layer and the weights are automatically adjusted so that the output layer indicates the correct label of the training instance. The process repeats for all training instances. The trained network with the learned weights represents a classifier.

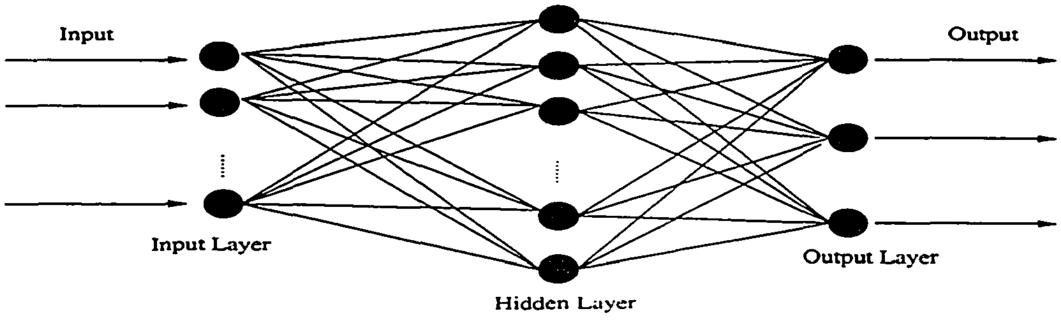


Figure 4.1: A Multi-layer Neural Network

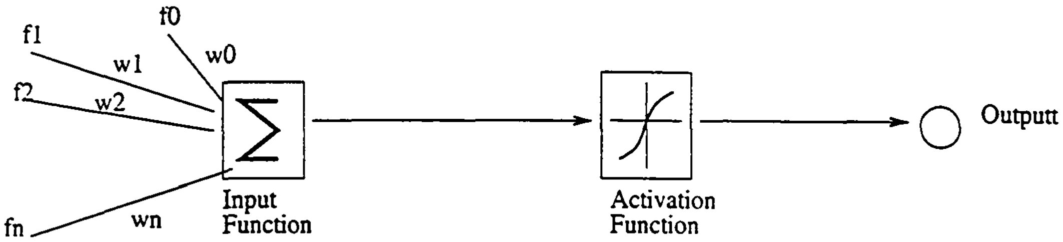


Figure 4.2: A neuron

A simple computation on neural network inputs is illustrated in Figure 4.2. The neuron has $n$ inputs $f_i$ and an extra input $f_0 = 1$, and accordingly $n + 1$ weights. The computation is split into two parts: a linear component, called the *input function*, which computes the weighted sum of the input units, and the second is a nonlinear component called the *activation function*, which transforms the weighted sum into the final output. More specifically, the input function is $in(x_i) = \sum_{j=0}^{n} f_j(x_i)w_i$ and the activation function is actually a sigmoid function: $g(y) = \frac{1}{1+e^{-y}}$. The final output ranges between 0 and 1, increasing monotonically with its input. Notice that the quantity $-w_0$ is a threshold that the weighted combination of inputs $w_1 f_1(x_i) + ... + w_n f_n(x_i)$ must surpass in order for the neuron to output a number over .5.

The procedure of learning weights of a single-layer neural network is as followings [Mit97]:

- **Create a feed-forward network that allow signals to travel one way only, from input to output;**
- **Initialize all weights with random values;**
- **Until the termination condition is met, Do**

  for each training instance $(x_i, y_i)$

  compute the linear combination of all the inputs to the input layer;

  compute the value of the output nodes using the activation function;

  evaluate the error between the expected target value and the output;

  update the weights of the network as following:

  - for each weight $w_j$, do $\Delta w_j = \Delta w_j + \eta(y_i - \hat{y}_i)x_i$ where $\eta$ is the learning rate
  - for each weight $w_j$, do $w_j = w_j + \Delta w_j$

The learning rate $\eta$ is a positive constant which moderates the degree to which weights are changed at each step. It is usually set to some small value, e.g., 0.1 [Mit97]. The termination condition could be a fixed number of iterations through the loop, or once the error on the training examples falls below some threshold, or the weights converge, etc [Mit97].

Learning in multi-layer neural networks proceeds the same way as above. Sample inputs are presented to the network. If the output computed by the network is same as the target, nothing will be done. Otherwise, there is an error between output and target and then the weights are adjusted to reduce the error. It is more complicated than in single-layer network to update all the weights because there are many weights connecting each input to an output. The Backpropagation algorithm [Mit97] is used to adjust the weights for a multi-layer network, given a network with a fixed set of units and interconnections. Once the neural network is built and the thresholds are set for the outputs, it can be used to classify new instances. To do this, one must feed the features of the new instance to the input nodes of the network; the class label will be generated at the output layer according to the thresholds.

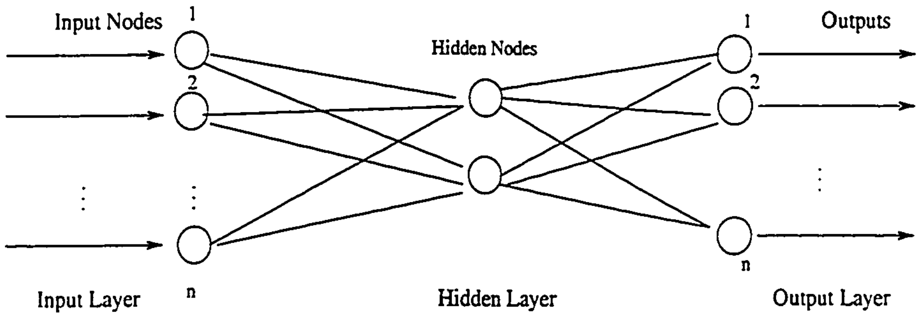


Figure 4.3: A Typical Auto-encoder Neural Network with 2 Hidden nodes. The network attempts to learn weights that map the input patterns to an internal representation and then back again to patterns identical to the inputs.

By slightly changing the structure and learning scheme of neural network, an auto-encoder network [JMG95] can be built using examples from only one class. An auto-encoder network, also called an auto-associator, is a neural network which learns to map from input nodes to output nodes through a narrow hidden layer [JMG95]. An auto-encoder network has the same number of output nodes and input nodes but a smaller number of hidden nodes. Figure 4.3 shows the structure of a typical one-hidden-layer auto-encoder neural network. An auto-encoder tries to encode as accurately as possible the identity mapping from inputs to outputs. Since there is a reduced number of dimensions in the hidden layer, it creates a lower dimensional bottleneck through which input information must be squeezed. Thus, the hidden layer compresses the information from the input layer and then decompresses it at the output layer, as the name auto-encoder implies.

Similar to a general ANN, an auto-encoder is trained using Backpropagation, but only on positive instances of one class. Once trained, the attendee can be fed new instances that it tries to reconstitute at its output layer. The quality of the reconstruction is evaluated by computing the sum of the squared error at each corresponding input and output node. If this error is small enough, then the instance is labelled "positive" since it is likely to be an instance of the class; otherwise, it is labelled "negative" since it is likely to be a counter-example [Jap99].

The intuition behind the learning strategy of an auto-encoder is that when the auto-encoder is tested on new positive examples, it is expected to reconstruct them well since they involve a pattern similar to those of the positive training data examples. Since negative examples present patterns different from those of the positive data, the compression and decompression which allows the network to reconstruct the positive examples will not work on the reconstruction of negative patterns. Therefore, we expect that an auto-encoder succeeds in reconstruction of positive data and fails on negative data and, thus, new instances can be classified based on the failure or success of reconstruction [Jap99].

In order to determine whether a novel test example is a positive or a negative instance, an auto-encoder's reconstruction error must be compared against some threshold, separating the positive from the negative class. By feeding a number of positive and negative examples to the auto-encoder after its training phase, a threshold can be determined from the observation of these examples [Jap99].

## 4.2 $k$-Nearest Neighbor and 1-class kNN

The $k$-Nearest Neighbor algorithm assumes all instances correspond to points in the $n$-dimensional space $R^n$. The nearest neighbors of an instance are defined in terms of the standard Euclidean distance. More precisely, let an arbitrary instance $x_i$ be described by the feature vector $< f_1(x_i), f_2(x_i) ... f_n(x_i) >$. Then the distance between two instances $x_1$ and $x_2$ is defined to be $d(x_1, x_2)$, where

$$d(x_1, x_2) = \sqrt{\sum_{j=1}^{n} (f_j(x_1) - f_j(x_2))^2}$$

If we consider each instance as in the form of $< x_i, F(x_i) >$, $F$ is a mapping function $F : R^n \to C$ where $C$ is the finite set of classification categories $c_1, c_2 ... c_s$ and $s$ is the number of classes in the target space. The kNN algorithm for learning and classifying a new testing example is the following [Mit97]:

- **Training: for each training example $< x_i, F(x_i) >$, add the example to the list *training_examples*;**
- **Classification: Given a query instance $x_q$ to be classified,**

  Return: the most commonly occurring class among the $k$ training examples nearest to $x_q$.

For example, if we choose $k = 1$, then the 1NN algorithm labels $x_q$ the class of its nearest training instance.

Figure 4.4 shows the operation of kNN algorithm for the case where the instances are points in two-dimensional space and where there are two target classes, positive and negative. An unknown testing example $x_q$ is shown as well. In this example, 1NN algorithm classifies $x_q$ as positive whereas 5NN classifies it as negative.

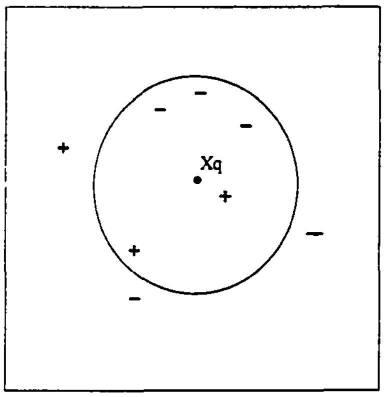


Figure 4.4: k-Nearest Neighbors. Different values of $k$ result in different classification [Mit97].

The kNN algorithm is easily revised to perform 1-class classification. To accomplish this, we divide training data for each class, and store them separately. When a testing example is encountered, we calculate the distances of the new example to all the training data of class $i$. To perform classification, a distance threshold needs to be determined. If $k = 1$, the distance to the nearest training example in class $i$ is compared to the threshold to decide if the testing example belongs to this class. For larger values of $k$, we can have the algorithm calculate the mean value of the $k$ nearest training examples and compare the average distance to the threshold.

## 4.3 Support Vector Machines and 1-class SVM

A simple SVM learns how to classify objects into two classes $+1, -1$ from a set of labelled training instances

$$(x_1, y_1), ...(x_m, y_m)$$

where $x_i \in R^n$, and $y_i \in \{-1, +1\}$ for $i = 1, 2, ...m$, where $m$ is the number of training instances. The main idea is to find the hyperplane that separates the training instances by a maximal margin, as shown in Figure 4.5. The margin is defined as the minimal distance of an instance to the hyperplane. In the Figure, there are 3 different hyperplanes. All of them can separate the two classed linearly, but the optimal hyperplane would be the one denoted as "Optimal Hyperplane" in the Figure, based on the definition of maximal margin. It is obvious that in the Figure the distance of the closest instance to that optimal hyperplane is larger than that of the other two hyperplanes. The instances lying closest to the separating hyperplane, both positive and negative, are called *support vectors.* If the training examples are separable by a hyperplane, we can use a linear function of the form $F(x) = w \cdot (x) + b$ [MMR+01]. If $F(x) \geq 0$, $x$ is classified into the positive class; otherwise, it is negative.

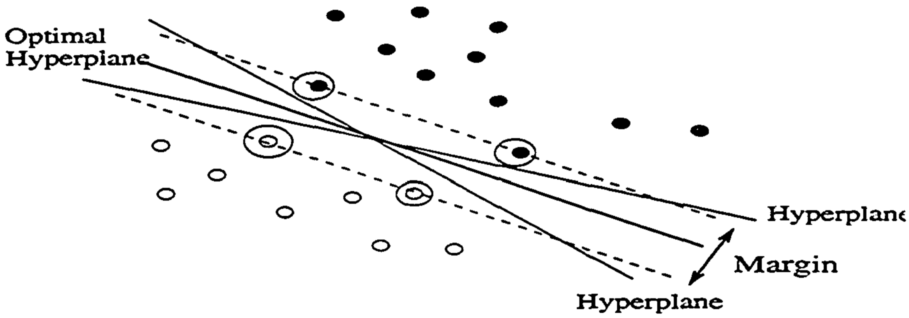


Figure 4.5: Linear separating hyperplanes for the separable case. The support vectors are circled. The optimal hyperplane separates the training examples by a maximal margin.

More generally, an SVM provides a projection of the original training instances into a higher dimensional feature space via a nonlinear mapping [MMR+01]. The data $x_1, x_2, ...x_m \in R^n$ is mapped into a potentially much higher dimensional feature space $R^{n'}$. Thus the learning problem is in the higher dimensional feature space $R^{n'}$ instead of $R^n$. For a given learning problem one can now consider the same algorithm in $R^{n'}$ instead of in $R^n$. There are many different mapping functions. Figure 4.6 is an example of a polynomial mapping function.

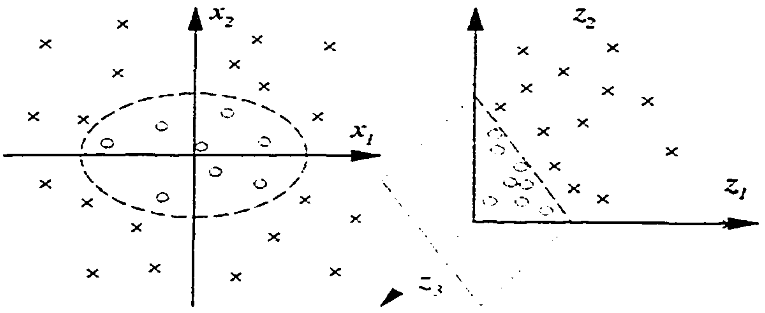


Figure 4.6: A polynomial mapping from two dimensional space to three dimensional space. The examples are linearly separable after the mapping. Left: In input space this construction corresponds to a *nonlinear* ellipsoidal decision boundary. Right: Using $z_1 = (x_1)^2$,$z_2 = sqrt(2x_1x_2)$ and $z_3 = (x_2)^2$ as features, a separation in feature space can be found using a *linear* hyperplane [MMR+01]. $x_1, x_2$ are the two features in the original feature space; $z_1, z_2, z_3$ represent the three new features in the 3-D feature space.

Originally SVMs were designed for binary classification. For a multi-class problem, an SVM cannot be applied directly. Instead, a multi-class learning problem has to be divided into disjoint binary classification tasks. The division can be in the manner of one-against-the-rest or pairwise. To classify a new object, all binary SVM classifiers have to be invoked and their decisions combined to make a final decision.

To tackle the problem of 1-class classification, a 1-class SVM algorithm was proposed in [SPSTS01]. The algorithm is to find some hyperplane $w$ that separates the positive training data from the *origin* at threshold $p$. The *origin* is treated as the only negative example. If such a hyperplane and threshold can be found, then the estimation function

$f(x) = sgn(w \cdot \Phi(x) - p)$ where sgn(a) = 1 if $a > 0$ and sgn(a) = 0 otherwise, and where $\Phi$ is the mapping function from $R^n$ to $R^{n'}$.

will be used to decide whether a new patten $x_q$ belongs to positive class. If $f(x_q) < p$, $x_q$ is classified as positive. A different approach in [TD99] uses a *sphere* to describe the data feature space, which contains as many as possible of positive training data while keeping the sphere small. This algorithm has been shown to be equivalent to the approach in [SPSTS01]. Figure 4.7 illustrates the two similar ideas.

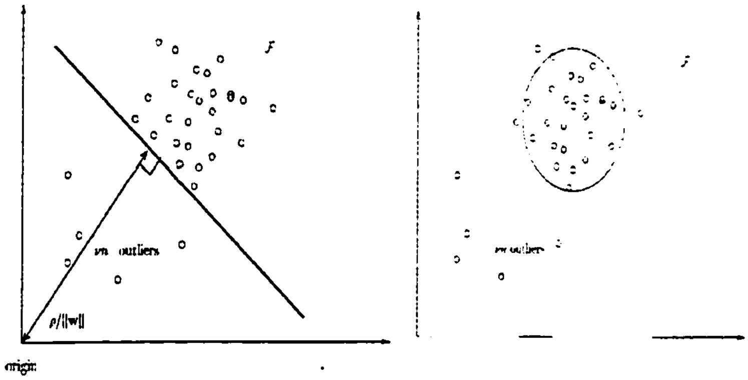


Figure 4.7: (Left): a hyperplane is constructed that maximizes the distance to the origin while allowing for $v$ outliers. (Right): construction of the smallest sphere under some threshold that contains positive data [MMR+01].

For our multi-class task, an individual 1-class SVM is trained for each artist class. Suppose we have $k$ classes, the $i^{th}$ SVM is trained with examples in the $i^{th}$ class. The examples of the other $k-1$ classes are expected to be classified as "outliers" in the testing phase. Therefore, $k$ SVMs need to be trained respectively, one for each of the $k$ classes. The predictions of all of the $k$ 1-class SVMs have to work together to perform the final classification (see section 4.5).

## 4.4 Naive Bayes and 1-class Naive Bayes

A Naive Bayes classifier is a highly efficient and practical Bayesian network classifier [Mit97]. Naive Bayes classifier applies to learning tasks where each instance $x$ is described by a conjunction of attribute values and where the target function $F(x)$ can take on any value from some finite set $C = c_1, c_2..c_s$ where $s$ is the number of classes in the target space. For example, in Table 4.1, $C$ represents *Play Tennis* and it has two values: $c_1 = true, c_2 = false$.

| Play Tennis | Temperature | | Windy | |
|---|---|---|---|---|
| | normal | high | yes | no |
| True | 0.90 | 0.10 | 0.40 | 0.60 |
| False | 0.20 | 0.80 | 0.70 | 0.30 |

Table 4.1: An example of Naive Bayes Classifier

Given a new instance $x_q$, described by the tuple of attribute values $< f_1(x_q), f_2(x_q)...f_n(x_q) >$, e.g., $f_1(x_q)$ might be *Temperature = normal*, $f_2(x_q)$ might be *Windy = yes*, the classifier is asked to predict the target class [Mit97]. To do that, the classifier must learn from training data, the

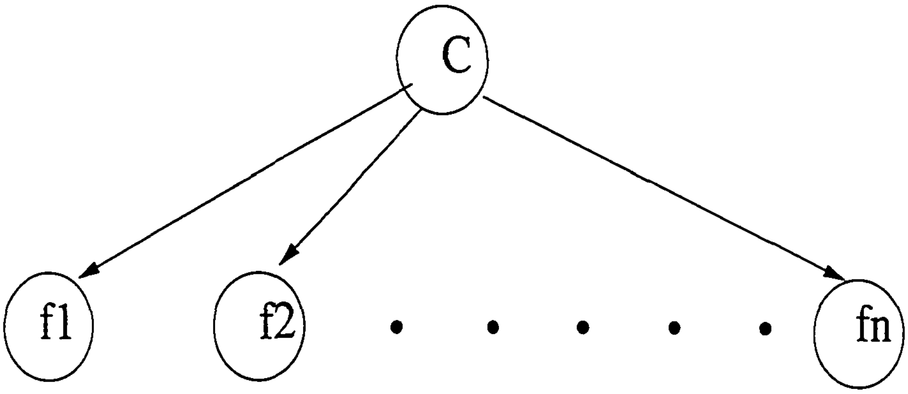


Figure 4.8: A Naive Bayes classifier depicted as a Bayesian network in which the predictive attribute $(f_1, f_2...f_n)$ are conditionally independent given the target class.

conditional probability of each attribute $f_j(x_q)$ given each class label $c_k$ in $C$. For example, the probability of *Temperature = normal* given *Play Tennis = true*,

$$P(Temperature = normal|PlayTennis = true) = 0.90$$

. Table 4.1 is called ***conditional probability table (CPT)***. By using the probabilities from the *CPT*, the probabilistic classifier will typically compute posterior probabilities for each class $P_k = P(c_k|f_1(x_q), f_2(x_q)...f_n(x_q))$ and then return the class label $\hat{y}_i = argmax_k(P_k)$.

The Naive Bayes classifier is based on the simplifying assumption that the attribute values are conditionally independent given the target value. In other words, the assumption is that given the target values of the instance, the probabilities for the individual attributes is: $P(f_1(x_q), f_2(x_q)...f_n(x_q)|c_k) = \prod_j P(f_j(x_q)|c_k)$. Given this assumption, we can derive a formula using Bayes Rule:

$$\begin{aligned}\hat{y}_q &= argmax_{c_k} P(c_k|f_1(x_q), f_2(x_q)...f_n(x_q))\\ &= argmax_{c_k} \frac{P(f_1(x_q), f_2(x_q)...f_n(x_q)|c_k)P(c_k)}{P(f_1(x_q), f_2(x_q)...f_n(x_q))}\\ &= argmax_{c_k} P(f_1(x_q), f_2(x_q)...f_n(x_q)|c_k)P(c_k)\\ &= argmax_{c_k} P(c_k)\textstyle\prod_j P(f_j(x_q)|c_k)\end{aligned}$$

where $\hat{y}$ denotes the class label output by the Naive Bayes classifier. Thus, when depicted graphically, a Naive Bayes classifier has the form shown in Figure 4.8.

Building a Naive Bayes classifier involves estimating all the $P(c_k)$ and $P(f_j(x_q)|c_k)$ terms, based on their frequencies in the training data. To use the classifier for predicting unknown examples, it employs the pre-computed CPTs and applies the Bayes rule formula to the new example. For example, if given a new instance *Temperature = normal, Windy = yes*, it computes

$$P(PlayTennis = true|Temperature = normal, Windy = yes) =$$
$$P(PlayTennis = true) * P(Temperature = normal|PlayTennis = true)*$$
$$P(Windy = yes|PlayTennis = true) = 0.5 * 0.9 * 0.4 = 0.18$$

and $P(PlayTennis = false|Temperature = normal, Windy = yes) = 0.07$. Therefore, the conclusion is the target class is *Play Tennis*. The probabilities are actually $P_{PlayTennis=true} = \frac{.18}{.18+.07} = .72$ and $P_{PlayTennis=false} = \frac{.07}{.18+.07} = .28$.

An important problem for the Bayesian approach is how to handle continuous variables. It can be solved by either discretizing, or assuming that the data are generated by a single Gaussian [JL95]. Based on the assumption that the values of numeric attributes are normally distributed, we can write

$$P(f_j(x_q)|c_k) = g(f_j(x_q), \mu, \sigma), \text{ where}$$
$$g(f_j(x_q), \mu, \sigma) = \frac{1}{\sqrt{(2\pi)}\sigma} e^{-\frac{(f_j(x_q)-\mu)^2}{2\sigma^2}}$$

the probability density function for a normal (Gaussian) distribution. For each class and attribute, one can estimate the probability that the attribute will take on each value in its domain, given the class. For each class and continuous attribute, one must estimate the mean and standard deviation of the attribute given the class. Maximum likelihood estimation of these parameters is straightforward. The estimated probability that a random variable takes a certain value is assumed to be equal to its sample frequency. Under this assumption, the maximum likelihood estimates of the mean and standard deviation of a normal distribution are the sample average and the sample standard deviation [Sch92].

Instead of using one single Gaussian, a method called **Flexible Naive Bayes** using *kernel density estimation* was proposed in [JL95]. It is the same as Naive Bayes in all respects but one: the estimated density is averaged over a large set of data points

$$P(f_j(x_q)|c_k) = \frac{1}{m} \sum_i g(f_j(x_q), \mu_{ij}, \sigma_{c_k})$$

where $i$ ranges over the training points in class $c_k$, $\mu_{ij}$ is the value of feature $j$ of the $i^{th}$ training point and $m$ is the number of training points in class $c_k$. Thus **Flexible Naive Bayes** must store every continuous attribute value it sees during training. The only statistic for the list of $\mu_{ij}$ is the list of values of $j^{th}$ feature itself. When computing the weighted kernel density for a continuous attribute to classify a new instance, the algorithm performs $m$ evaluations, one per observed value of $f_i$ in the class $c_k$ training examples. For the value of $\sigma$, we set $\sigma_{c_k} = \frac{1}{\sqrt{m}}$, $m$ is the number of training instances observed with class $c_k$ [JL95].

We can construct a 1-class detector using the Naive Bayes approach. By multiplying the estimated density over training data with class $c_i$, for each feature value of a testing example, the product shows the likelihood of the testing example being in class $c_i$. If a threshold is set, the output likelihood from the Bayesian estimator for certain class $c_i$ can be used to perform classification. In our work, we construct such a 1-class estimator for each artist and combine them with all the other 1-class detectors to produce final decisions.

## 4.5 Combining Multiple Classifiers

In our work, instead of setting thresholds for the classifiers built using the aforementioned learning algorithms and use them to perform classification directly, we use them as *scoring functions*. For auto-encoder and 1-class kNN, the output is reconstruction error and Euclidean distance respectively. The smaller the output is, the more the example in question is considered to belong to the class the auto-encoder or kNN is trained for. On the contrary, for 1-class SVM and Naive Bayes, a larger the output is a stronger indication that an example is a positive example of the class.

For each class, its four different types of 1-class detectors produce numbers in different ranges. They have to be scaled into a similar range before being combined with detectors for other classes to perform the final classification. In order to take a right decision based on the outputs from these multiple 1-class detectors, we need to find a combining rule which is able to resolve the conflicts when 1-class detectors disagree, i.e. more than one class is considered promising or no class can be considered sufficiently promising. This step is sometimes called Meta-learning [CS95].

To combine our detectors, we use a single layer neural network stacked over the pre-trained multiple 1-class detectors to combine these different base detectors and to improve their performance. The base classifiers can be individual auto-encoders, 1-class kNN, 1-class SVM, 1-class Naive Bayes estimators or any combinations of them. Each neuron $i$ of the neural network scales the outputs from its different 1-class detectors into the range of $(0, 1)$, and the output of the neuron $i$ can be considered as an estimate of an instance being in class $i$. Figure 4.9 shows an example structure of our system.

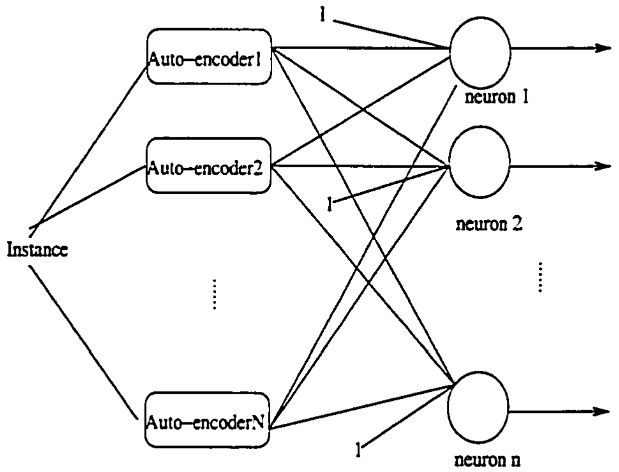


Figure 4.9: A Single Layer Neural Network Stacked on Auto-encoder Detectors

In Figure 4.9, the neural network has same number of input neurons as auto-encoders. Each neuron gets inputs from all $n$ auto-encoders and has another input which is 1. By learning the weights for these inputs, each neuron is supposed to learn to combine the pre-trained auto-encoders, amplifying the strength of good ones and reducing the negative impact of bad ones. Thus, each neuron and its base 1-class detectors comprise a new 1-class detector.

After training of all the base detectors, the single layer neural network can be trained by feeding the base detectors' outputs for all the training data to the neural network. Classification decision is supposed to be produced at the neural network level, while the outputs from the neurons are numbers between 0 and 1, which can not generate a class label directly. Therefore, the other task for the training of the single layer neural network is to find some thresholding rule. For that purpose, we experimented with several different methods described and compared in detail in Chapter 6.

Once the training of the single layer network is done, testing examples can be fed to the stacked system to be classified. A testing example will first be pushed through all the 1-class detectors on the base level and the outputs from base level will be fed to the neural network. The output values from the $n$ output neurons of the neural network will be compared and thresholded according to some pre-defined thresholding rule, and a class label will be picked for the testing example.

## 4.6 Summary

This chapter described four major classification techniques we employed in our research and their modification for the 1-class detection task. Instead of making these 1-class detectors into classifiers for each class and performing classification directly, we use them as components in the base level of our system. This chapter also stated our strategy for scaling and combining different 1-class detectors outputs which have different ranges and meanings, in order for the system to produce classification decisions.

# Chapter 5

# Evaluation of Classifier

This chapter describes the evaluation criteria we use to measure the performance of each individual classifier, as well as the whole system. In order to take a deeper look into the individual classifiers to give a deeper analysis of their performance, we introduce some more sophisticated evaluation methodology, rather than using only classification accuracy [PFK98]. These more detailed evaluation tools includes Confusion Matrices [Mit97], ROC Curves [Mar03] and Cost Curves [DH00].

## 5.1 Confusion Matrices

As described previously, a confusion matrix is a table which records the classification results. As a more complicated example than the one in chapter 2, Table 5.1 shows a confusion matrix for our system, trained and tested on data from 16 artist classes. Each entry in the table represents the number of music segments in the test set whose actual label is the row label and whose predicted label is the column label. For example, the number of testing segments of artist 3 that are incorrectly classified as artist 11 is 1. The sum column on the right end of the table indicates the number of test segments whose actual label is specified by the row label. The sum row at the bottom of the table indicates the number of test segments whose predicted labels is the column label.

As mentioned before, various statistics can be computed from the confusion matrix to evaluate the effectiveness of the classification system [vR79]. We use three standard statistics, ***precision***, *recall* and *accuracy*. Given a $n * n$ confusion matrix $M$ and a set of class labels $c_i$, the definitions of these statistics are as follows. The ***precision*** for each label $c_i$ is:

$P_i = M_{ii} / \sum_{k=1}^{k=n} M_{ki} = M_{ii}/SUM_i^{prd}$ where $SUM_i^{prd}$ is the column sum of column i;

The *recall* for each label $L_i$ is defined as:

$R_i = M_{ii} / \sum_{k=1}^{k=n} M_{ik} = M_{ii}/SUM_i^{obs}$ where $SUM_i^{obs}$ is the row sum of row i;

| Prd⇒<br>Obs⇓ | a1 | a2 | a3 | a4 | a5 | a6 | a7 | a8 | a9 | a10 | a11 | a12 | a13 | a14 | a15 | a16 | $sum^{obs}$ | recall |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a1 | **11** | | | | | 1 | 1 | | 1 | | | | | | | | 14 | .786 |
| a2 | | **6** | | 2 | | | | | | | | | 2 | | | 1 | 11 | .546 |
| a3 | | | **16** | | | | | | | 1 | 1 | | | | | | 18 | .889 |
| a4 | | | 1 | **3** | 1 | | 1 | 2 | | 2 | | | 2 | 2 | 2 | 1 | 17 | .176 |
| a5 | | 1 | | | **6** | | 2 | | 1 | 3 | 3 | | 2 | | | | 18 | .333 |
| a6 | | 1 | | | | **6** | | | | 1 | 3 | | 2 | 1 | | | 14 | .429 |
| a7 | | | | 1 | 2 | | **16** | | | | 4 | | 1 | 3 | | 2 | 29 | .552 |
| a8 | | | | | | | | **8** | | | 3 | | | | | 1 | 12 | .667 |
| a9 | 1 | 3 | | | | | 1 | | **4** | | 1 | 1 | 1 | | 2 | 1 | 15 | .267 |
| a10 | 1 | | | | 2 | | 3 | | | **11** | 2 | | 2 | 1 | | | 22 | .500 |
| a11 | | | | 2 | | | | 1 | | | **22** | | | 2 | 1 | 1 | 29 | .759 |
| a12 | | | | | | | | | | | | **21** | | | | | 21 | 1.00 |
| a13 | 2 | | | 2 | 3 | | 4 | | | 3 | 2 | | **6** | 1 | | | 23 | .261 |
| a14 | | 2 | | | | | 1 | | | | 2 | | 1 | **13** | | | 19 | .684 |
| a15 | | | | | | | | | | | | | | | **16** | 2 | 18 | .889 |
| a16 | 1 | | | | | | | | | 1 | 2 | 1 | | | | **9** | 14 | .643 |
| $sum^{prd}$ | 17 | 15 | 17 | 19 | 13 | 7 | 29 | 9 | 6 | 19 | 44 | 23 | 19 | 22 | 19 | 17 | **295** | |
| precision | .647 | .400 | .941 | .632 | .462 | .857 | .552 | .889 | .667 | .579 | .500 | .913 | .314 | .591 | .842 | .529 | | .624 |

Table 5.1: Confusion Matrix for the Classification System on 16-artists dataset

The *accuracy* of the classification system is defined by:

$A_i = \sum_{i=1}^{i=n} M_{ii} / \sum_{k=1}^{k=n} \sum_{j=1}^{j=n} M_{kj} = \sum_{i=1}^{i=n} M_{ii}/SUM$ where $SUM$ is the sum of all the numbers in the matrix $M$.

For example, the precision of class 3 is $P_3 = 16/17 = .941$, the recall of class 3 is $R_3 = 16/18 = .889$. The accuracy of the classification system is $A = 184/295 = .624$.

## 5.2 ROC Curves and Cost Curves

For classification problems, as previously mentioned, accuracy is often not an appropriate measure of classifier performance especially with imbalanced data, because using classification accuracy as a measurement assumes equal misclassification costs and that the class distribution is known for the target environment [PFK98]. Unfortunately, these assumptions do not hold for real-world data. To amend the problems of evaluating classifier using accuracy, ROC(Receiver Operating Characteristic) curves have been proposed, which measures classifier performance over the full range of possible costs and class frequencies [DH00]. As an alternative to the ROC curve, the idea of a Cost curve [DH00] can be derived from ROC curve and provides another method of classifier evaluation.

### 5.2.1 ROC Curves

Originating from the field of Signal Detection Theory developed during World War II from the analysis of radar images for enemy detection, *Receiver Operating Characteristic (ROC)* curves measure the ability of radar receiver operators to make important distinctions among enemy targets, friendly ships, and noise. They have been adapted and put into use for interpreting classification results since the 1970's. Figure 5.1 shows a simple example of an ROC curve. The ROC curve is a two dimensional measure of classification performance. Basically it deals with binary classifications, and in its simplest form is a plot of the hit rate (true positive) vs. the false alarm (false negative) rate, as a decision threshold is varied [Mar03].

Each point on an ROC curve is constructed from a $2*2$ confusion matrix, since it reflects binary classification performance, and a threshold as the cut point of examples from the two classes, i.e. positive and negative. Figure 5.2 shows the distribution of the outputs for positive and negative examples made by a classifier. The distributions overlap and consequently no classifier distinguishes positive and negative examples with 100% accuracy. By setting a threshold, we consider the examples

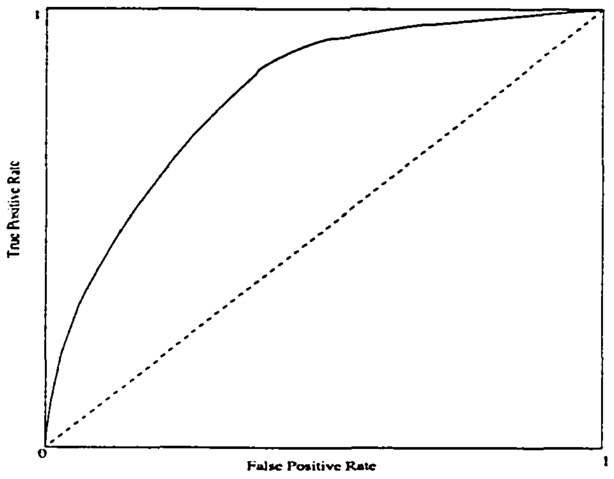


Figure 5.1: A Typical ROC Curve.

above it to be positive and negative below it. The position of the threshold will determine the number of true positive, true negatives, false positives and false negatives. Using different thresholds may minimize one of the erroneous types of prediction results.

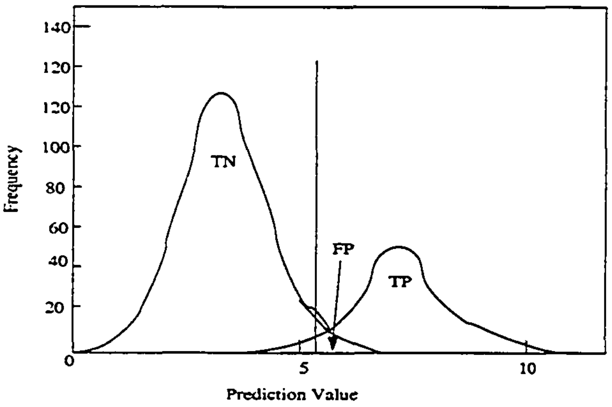


Figure 5.2: Overlapped Distribution of Prediction Values

Suppose we have prediction results in Table 5.2 when the threshold is set at 5. The true positive rate is 18/20 and the false positive rate is 4/96. These two numbers define the coordinates of a point on the ROC curve. Once the threshold changes, the true positive and false positive rate change accordingly and a new point of the ROC curve is made.

| **Prediction⇒**<br>**Actual Class⇓** | **$\geq 5$(Positive)** | **$< 5$(Negative)** | **Totals** |
|---|---|---|---|
| Positive | 18 | 2 | 20 |
| Negative | 4 | 92 | 96 |

Table 5.2: A 2-class confusion matrix

Notice that the true positive rate can be improved by moving the threshold to a smaller value, i.e. to make the criteria for a positive example less strict. On the other hand, the false positive rate can be reduced by moving the threshold to a bigger value to make the criteria for a negative example less strict. Thus, there will be a trade-off between true positive rate and false positive rate. Changing the definition of a positive example may improve one of the two rates, but the other will decline. By setting several thresholds, different points representing pairs of true positive and false positive rates can be calculated and an ROC curve is made by connecting these points.

An ROC curve demonstrates several things:

- It shows the trade-off between true positive and false positive rates.
- The closer the curve follows the left-hand border and then the top border of the ROC space, the more accurate the classifier is.
- The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the classifier.
- The *Area Under the Curve (AUC)* is a measure of test accuracy.

Figure 5.3 shows three ROC curves representing excellent, good, and worthless classifier performance plotted on the same graph.

### 5.2.2 Cost Curves

Traditional ROC analysis has as its primary focus determining which diagnostic system or classifier has the best performance independent of cost or class frequency. But there is also an important secondary role for selecting the set of system parameters for an individual classifier, which gives the best performance for a particular cost or class frequency [DH00]. To directly compare the performance of two classifiers we can transform an ROC curve into a cost curve.

The x-axis in a cost curve is the probability-cost function (PCF) for positive examples.

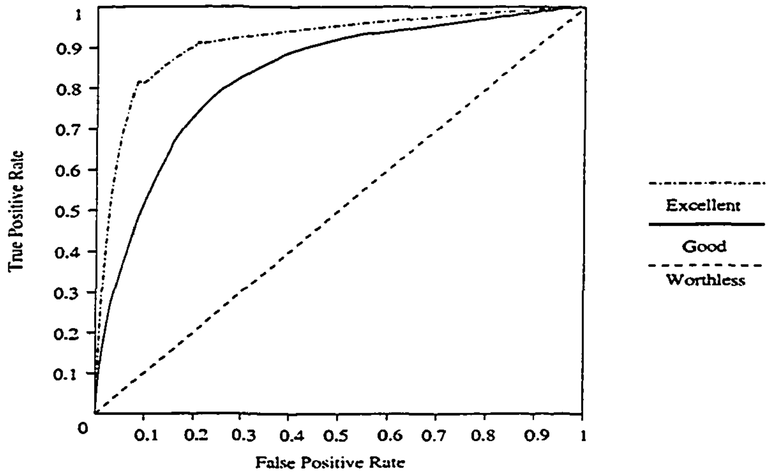


Figure 5.3: Comparison of Three ROC Curves

$$PCF(+) = w_+/(w_+ + w_-) \text{ where}$$
$$w_+ = p(+)C(-|+) \text{ and } w_- = p(-)C(+|-)$$

$p(a)$ is the probability of a given example being in class $a$ and where $C(a|b)$ is the cost incurred if an example in class $b$ is misclassified as being in class $a$.

PCF turns out to be simply $p(+)$, the probability of a positive example, when the costs are equal. The y-axis is expected cost normalized with respect to the cost incurred when every example is incorrectly classified [DH00]. This is simply the error rate of classification when the costs are equal.

To construct a cost curve from an ROC curve, a point (TP, FP) representing a classifier in ROC space is converted into a line in cost curve space using the equation:

$$y = (1 - TP - FP) * PCF(+) + FP$$

Figure 5.4 shows lines representing four extreme classifiers in the cost space. At the top is the worst classifier which is always wrong and has a constant error rate of 1. At the bottom is the best classifier which is always right, and has a constant error rate of 0. The classifier that always chooses negative has zero error rate when PCF(+)=0 and a error rate of 1 when PCF(+)=1 [DH00]. The classifier that always chooses positive has error rate 1 when PCF(+)=0 and a zero error rate when PCF(+)=1. It is apparent that we should never use a classifier outside the shaded region of Figure 5.4, as it actually performs worse than the majority classifier which chooses one or other of the trivial classifiers that label everything with the majority class depending on PCF(+).

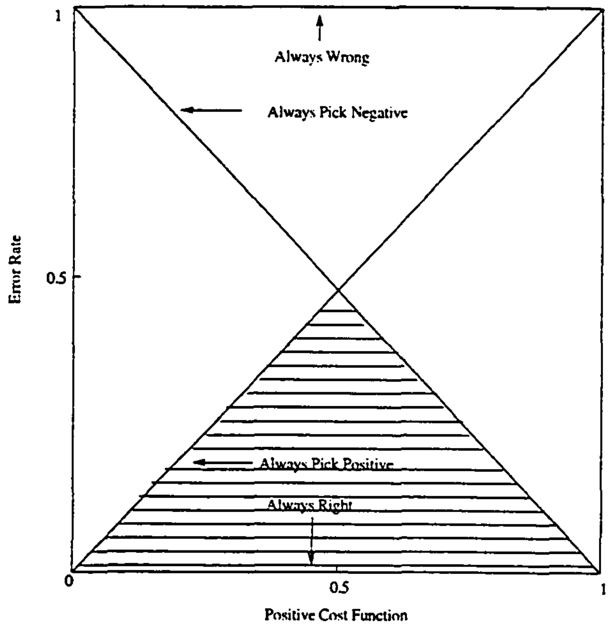


Figure 5.4: Extreme Classifiers

By repeatedly converting points from ROC space, a set of classifiers can be represented in cost space, as shown in Figure 5.5. We can directly measure the vertical height difference at some particular probability-cost value. If one classifier is lower in expected cost (error rate) across the whole rage of the probability-cost function, it dominates the others [DH00]. Each classifier delimits a half-space. The intersection of the half-spaces of the set of classifiers gives the ***lower envelope*** which is indicated by the dashed line in Figure 5.5.

Figure 5.6 shows two ROC curves and their corresponding cost curves are shown in Figure 5.7. The two straight lines are two trivial classifiers. Cost curve 1, which is the higher curve in Figure 5.7, corresponds to ROC curve 1, which is lower in Figure 5.6, and the lower cost curve 2 corresponds to the upper ROC curve.

The distance between cost curves for two classifiers directly indicates the performance difference between them [DH00]. In Figure 5.7, the classifier represented by cost curve 2 outperforms the one by cost curve 1 because cost curve 2 has a lower or equal expected cost for all values of PCF(+). The maximum difference is about 200%(0.1 compared to 0.3), which occurs when PCF(+) is about 0.4 or 0.5. Their performance difference is negligible when PCF(+) is less than 0.1.

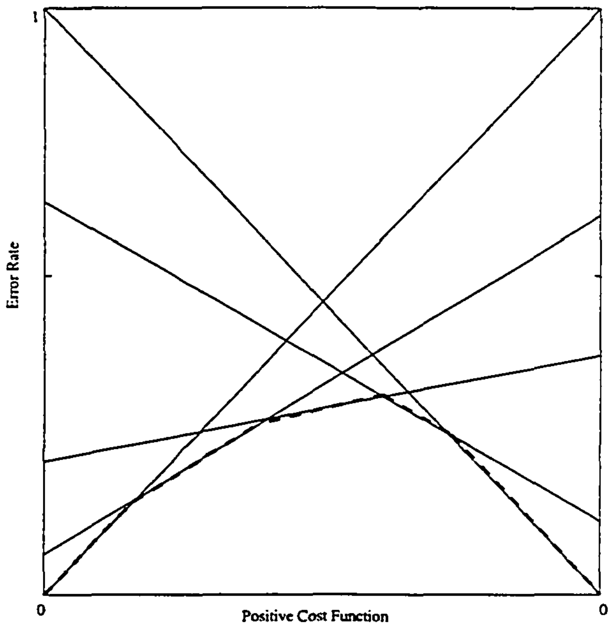


Figure 5.5: A Set of Classifiers

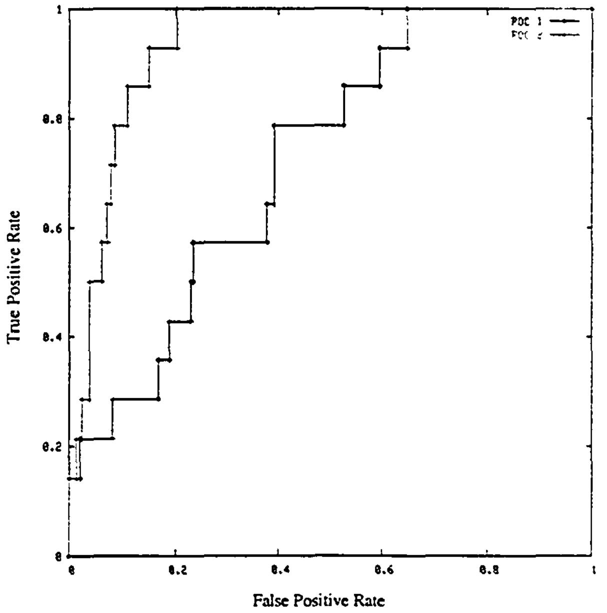


Figure 5.6: Two ROC Curves

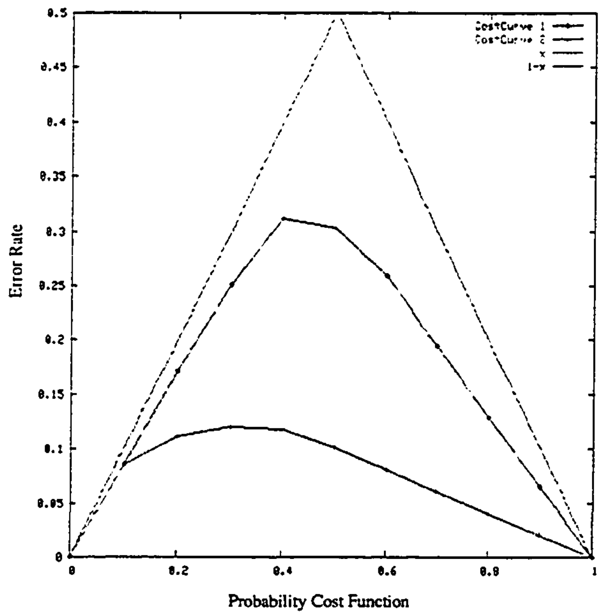


Figure 5.7: Two Cost Curves

## 5.3 Summary

This chapter gave detailed description about the classifier evaluation tools used in our work. Confusion Matrices are used to illustrate the performance of the whole system straight-forwardly, while another standard machine learning analysis tool, ROC curves, gives deeper analysis for each 1-class classifier. An alternative to ROC curves, cost curves, gives more comprehensive insights regarding imbalanced data distribution and misclassification cost. These tools provides better and more precise explanation about classifier performance than accuracy by itself.

# Chapter 6

# Experiment Design and Results

This chapter describes the design of various experiments and analyzes the results. We conducted different experiments on a 16-artists dataset, a 21-artists dataset and a 6-pianists dataset, using different sampling methods, combinations of base 1-class detectors and multi-classifier combination strategies. The experimental results are shown and analyzed using the measurements introduced above.

## 6.1 Experimental Design

In this work, we designed a number of different experiments to find better system settings and to improve the performance of our classifier.

After training four different sets of base 1-class detectors using the 1-class learning methods described previously, we first need to scale the outputs from different detectors into the same range. We then combine the detectors by stacking a single-layer neural network over the base level. We tried two different sets of inputs for building the stacked neural network to illustrate the effect of each set of neuron inputs.

To combine all the 1-class detectors and find best combination of them at the base level, we tried different combinations of these detectors and fed their outputs to train the meta-level neural network. This experiment helped us finding the best setting for base level components and build the best 1-class classifier for each class.

When training the stacked neural network, for the $i^{th}$ neuron, both positive and negative training data is fed into it and there is a data imbalance. For example, for the 21-artists dataset, data from each class is approximately $1/21$ of the whole training dataset, therefore, the number of examples from all the other classes is roughly 20 times as many as from the $i^{th}$ class. We designed several sampling methods to counteract the effect the data imbalance might have on the neural network training.

At the neural network output level, each neuron outputs a number between 0 and 1. To produce a final classification result, these neuron outputs have to be compared or combined according to some rule. To do this, we tried three methods to process the outputs of the neural network, including kNN, max-selection and a rescaling neuron on top.

The results of these experiments are presented in detail in Section 6.3. When experimenting with the 16-artists and 21-artists dataset, we randomly separate the feature vectors calculated from the 30 second segments. We use 80% of the data as training data, and 20% as testing data. All the experiments conducted for training our classifier and finding best setting of the system are at the segment level, instead of the song level. Note that for the 6-pianists dataset, we build only an auto-encoder, kNN and SVM for each class as 1-class detector. The Naive Bayes detector is not experimented on this dataset due to the problems caused by underflow in the process of estimation. Therefore, for this dataset, there are no results presented for experiments where the Naive Bayes detector is involved.

## 6.2 Datasets

One dataset we use is the same as in [BEL02]. The names of the artists in the dataset are listed in Table 6.1. Considering the fact there are 5 featured artists in the dataset whose musical productions are not vocal, we choose the 16 vocal artists to make up a sub-dataset as well as the whole dataset to train and test our classifier.

The other dataset we used is a collection of commercial recordings by six concert pianists of piano sonatas by Mozart. A sizable number of pieces are selected for performance measuring and analysis. The pieces, pianists and recordings are listed in Table 6.2 and 6.3.

In this dataset, 34 features are extracted from the recordings, characterizing changes of tempo and general loudness [WZ04]. This dataset is used to train our classifier to identify the performance of different pianists. However, in the original dataset we are provided with, there are 25 instances for which there are unknown feature values. As our system does not handle missing features, we discard those instances. This should not make a big difference since 25 instances are only a small portion of the whole dataset which contains over 23,000 instances.

| Class Label | Name | 16-artists dataset | 21-artists dataset |
|---|---|---|---|
| a1 | Beck | Y | Y |
| a2 | BelleSebastian | Y | Y |
| a3 | BuiltToSpill | Y | Y |
| a4 | EricMatthews | Y | Y |
| a5 | JasonFalkner | Y | Y |
| a6 | MercuryRev | Y | Y |
| a7 | MichaelPenn | Y | Y |
| a8 | RichardDavies | Y | Y |
| a9 | Sugarplastic | Y | Y |
| a10 | TheFlamingLips | Y | Y |
| a11 | TheMoles | Y | Y |
| a12 | TheRoots | Y | Y |
| a13 | Wilco | Y | Y |
| a14 | XTC | Y | Y |
| a15 | ArtoLindsay | Y | Y |
| a16 | AimeeMann | Y | Y |
| a17 | BoardOfCanada | N | Y |
| a18 | Cornelius | N | Y |
| a19 | DjShadow | N | Y |
| a20 | Oval | N | Y |
| a21 | MouseOnMars | N | Y |

Table 6.1: 16 and 21 Artists Datasets

| ID | Sonata | Movement | Key |
|---|---|---|---|
| kv279-1 | K.279 | 1st mvt. | C Major |
| kv279-2 | K.279 | 2nd mvt. | C Major |
| kv279-3 | K.279 | 3rd mvt. | C Major |
| kv280-1 | K.280 | 1st mvt. | F Major |
| kv280-2 | K.280 | 2nd mvt. | F Major |
| kv280-3 | K.280 | 3rd mvt. | F Major |
| kv281-1 | K.281 | 1st mvt. | Bb Major |
| kv282-1 | K.282 | 1st mvt. | Eb Major |
| kv282-2 | K.282 | 2nd mvt. | Eb Major |
| kv282-3 | K.282 | 3rd mvt. | Eb Major |
| kv330-3 | K.330 | 3rd mvt. | C Major |
| kv332-2 | K.332 | 2nd mvt. | F Major |

Table 6.2: Movements of Mozart piano sonatas selected for analysis

| ID | Name | Recording |
|---|---|---|
| DB | Daniel Barenboim | EMI Classicis CDZ 7 67295 2, 1984 |
| RB | Roland Batik | Gramola 98701-705, 1990 |
| GG | Glenn Gould | Sony Classical SM4K 52627, 1967 |
| MP | Maria Joao Pires | DGG 431 761-2, 1991 |
| AS | Adreas Schiff | ADD(Decca)443 720-2, 1980 |
| MU | Mitsuko Uchida | Philips Classics 464 856-2, 1987 |

Table 6.3: Pianists and recordings

## 6.3 Experiment Results

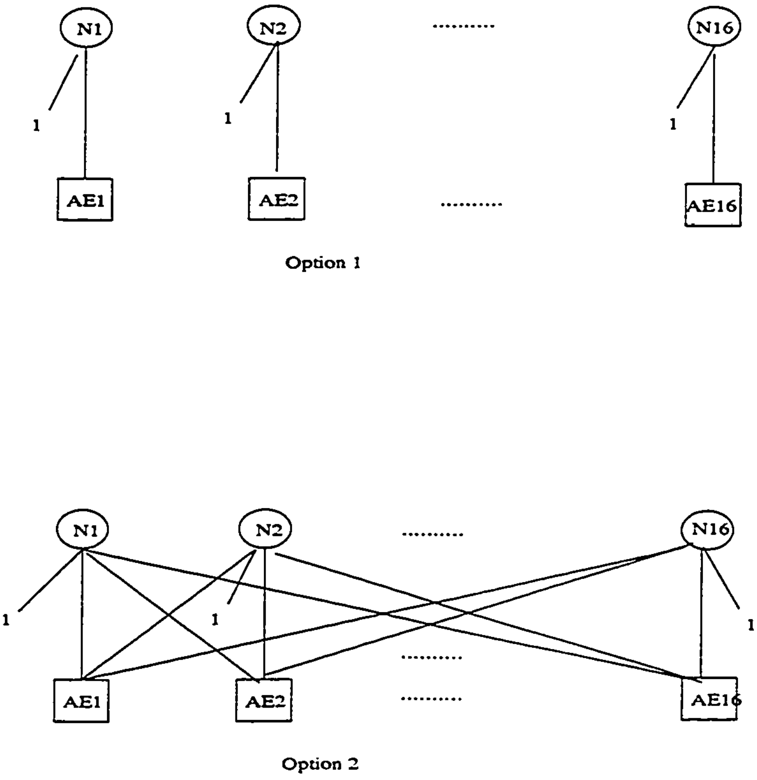
### 6.3.1 Different Inputs Into Meta-level Neurons

**Experiment:** Inputs to meta-level neurons.
**Options:**

1. feed only the output of the 1-class detector for class $i$ to the $i^{th}$ neuron plus a constant input 1;
2. feed outputs of 1-class detectors for all classes to each of the neurons, plus the constant input.

**Conclusion:** option 2 combines 1-class detectors better than option 1.

As mentioned before, the single layer neural network stacked on the multiple 1-class detectors can be trained in two different ways by feeding different inputs into it, as described above. By experimenting with the two options with each of the pre-trained 1-class detectors and evaluating classifier performance by Cost Curves, we conclude that option 2 is a better solution than option 1 as it better combines the outputs from the base level.



Option 1

Option 2

Figure 6.1: Two sets of inputs to neural network

Figure 6.1 illustrates the difference between the two options. In this example, only an auto-encoder is trained for each class. Their outputs are fed to the neural network in two different ways. One is to feed only the output of 1-class detectors for class $i$ to the $i^{th}$ neuron plus a constant input 1; the other is to feed outputs of all 1-class detectors to each of the input neurons, plus the constant input. We construct a cost curve for each of the base 1-class detectors. Then a cost curve is constructed for each of the neurons after training and testing the stacked neural network. Figure 6.2 shows the 3 cost curves for auto-encoder 6 (the higher curve), neuron 6 with option 1 (also the higher curve) and option 2 (the lower curve), trained using 16 classes vocal dataset. When there is only one type of 1-class detector used at the bottom level as shown in Figure 6.1, the cost curve produced by option 1 for neuron $i$ is identical the one constructed for the base detector $i$. This is because with option 1, neuron $i$ is trained only based on the output from its corresponding detector $i$, while with option 2, each neuron gets inputs from the detectors of all classes, and thus it learns to find interactions between detectors. The cost curve constructed for the testing results of a neuron in option 2 is usually better than the curve obtained in option 1. It proves that training each neuron using outputs from multiple 1-class detectors for all the classes improves the performance of every single 1-class detector.

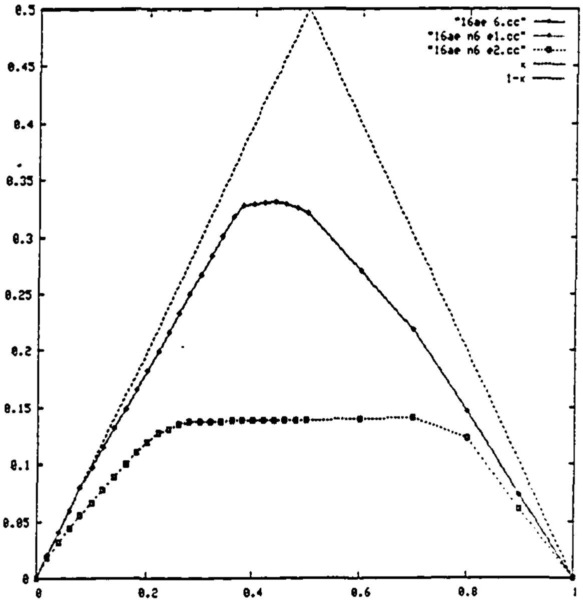


Figure 6.2: Cost curves when the base detector is an auto-encoder. The lower curve is for the neuron trained with option 2; the upper curve is for the neuron with option 1 and the auto-encoder by itself.

Figure 6.3, 6.4 and 6.5 show the analogous set of cost curves using 1-class kNN, SVM and Naive

Bayes at the base level respectively. All these curves show that option 2 outperforms option 1 when only one 1-class detector component is used. These are typical cost curves for most of the classes, produced by the two input options, using different 1-class detectors.

To summarize, Table 6.4 lists the comparison between the two options. Since there are 3 types of 1-class detectors and 3 different datasets, the experiment is conducted on all the combinations of detector and dataset. Each number in the "option 1" row in the table shows the number of classes for which the Cost Curve produced with option 1 is better than option 2; each number in the "option 2" row indicates the number of classes on which option 2 generates better Cost Curves.

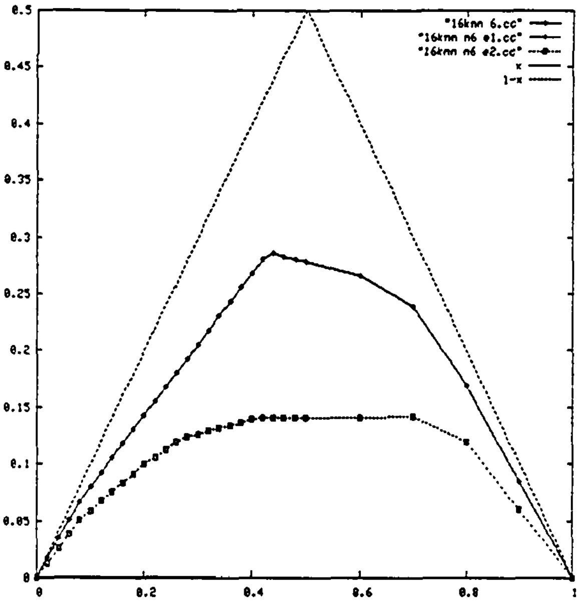


Figure 6.3: Cost curves when the base detector is a 1-class kNN. The lower curve is for the neuron trained with option 2; the upper curve is for the neuron with option 1 and the 1-class kNN by itself.

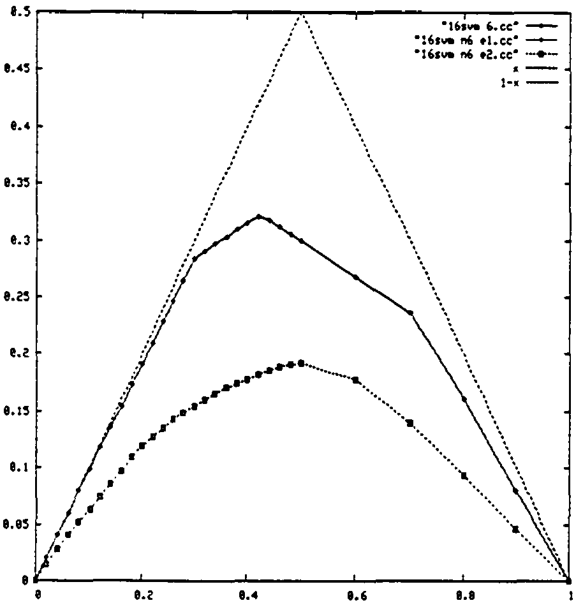


Figure 6.4: Cost curves when the base detector is a 1-class SVM. The lower curve is for the neuron trained with option 2; the upper curve is for the neuron with option 1 and the 1-class SVM by itself.

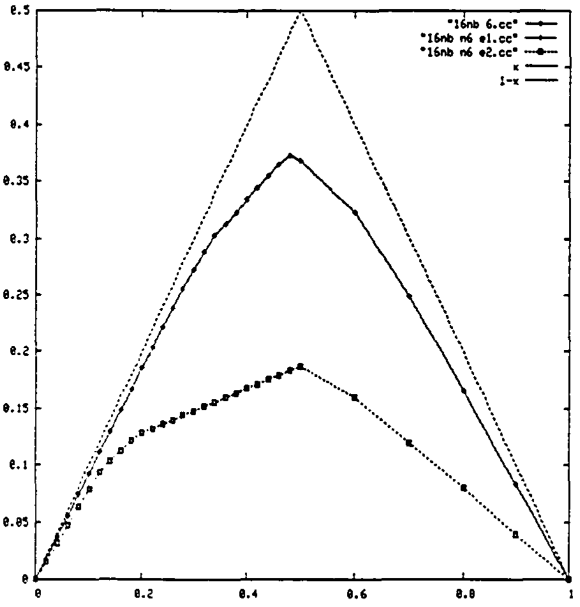


Figure 6.5: Cost curves when the base detector is a Naive Bayes estimator. The lower curve is for the neuron trained with option 2; the upper curve is for the neuron with option 1 and the Naive Bayes estimator by itself.

| Detector | Auto-encoder | | | kNN | | | SVM | | | Naive Bayes | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Dataset** | 16 | 21 | 6 | 16 | 21 | 6 | 16 | 21 | 6 | 16 | 21 | 6 |
| Option 1 | 0 | 2 | 2 | 2 | 2 | 1 | 1 | 0 | 0 | 4 | 6 | - |
| Option 2 | 16 | 19 | 4 | 14 | 19 | 5 | 15 | 21 | 6 | 12 | 15 | - |

Table 6.4: Summary of Results for neuron input options

### 6.3.2 Different Methods for Handling Imbalance

**Experiment:** 4 approaches to data imbalance for neural network training.
**Options:**

1. dynamic learning rate;
2. over-sampling;
3. under-sampling;
4. hybrid of over-sampling and under-sampling.

**Conclusion:** none of these approaches improve the classifier performance significantly.

Consider there is an imbalance in the data used to train the neurons. For example, using the 16-artists dataset to train 16 individual auto-encoders, when the meta level is trained, the meta training data are outputs from the auto-encoders after 16 classes of original training data are pushed through the auto-encoders. Therefore, neuron $i$ gets approximately $1/16$ positive training examples and $15/16$ negative examples which are from the other 15 classes. This happens to all the other neurons also. To reduce the impact the data imbalance may have on the meta level training, several different strategies are considered. They are dynamic learning-rate, over-sampling, under-sampling and the hybrid of over-sampling and under-sampling.

One way to solve the data imbalance is to use different learning rates for positive and negative examples. In the training phase, the learning rate for positive or negative examples changes depending on the ratio of number of negative examples to positive examples in training data set. If the number of negative examples is $n$ times the number of positive examples for neuron $i$, when positive examples are encountered during training, multiply the original learning rate $r$ for neuron $i$ by $n$, when negative examples are encountered, keep the original learning rate $r$ unchanged.

Over-sampling means to copy the examples of the positive class $n$ times, where $n$ is the ratio of number of negative examples to positive examples. For example, if the training data is comprised of $1/(n+1)$ positive examples and $n/(n+1)$ negative examples, we make $n$ copies of all the positive data and therefore achieve a balance between positive and negative examples. The strategy is of low efficiency and requires more space.

Under-sampling, on the other hand, is to down-sample the data of the majority class. Again, if the training data is comprised of $1/(n+1)$positive examples and $n/(n+1)$ negative examples, we use $1/n$ of the negative examples so that positive and negative data are balanced. The method makes the training faster and requires much less space.

The hybrid is to combine over-sampling and under-sampling, i.e. to make copies of positive examples and also down-samples negative examples to balance the data from both sides. For example, in the 16-artist dataset, for each class, the number of negative examples is roughly 15 times of the number of positive examples. We choose to over-sample the positive data by 3 times and under-sample the negative data to 1/5 of its original size.

To analyze the impact of these sampling methods on the performance of each classifiers, we compared them with different datasets and base detectors. The performance is evaluated by cost curves. Note that in both training and testing set, for each individual class, positive examples are approximately of 1/16 of all the examples for 16-artist dataset. Therefore, when analyzing the cost curves, what matters most is the left bottom corner of each cost curve graph.

Figure 6.6 shows the cost curves for the neuron for class 16, when auto-encoders are the base detectors. In the upper figure, the square-dotted line represents the cost curve for neuron 16, trained with outputs from auto-encoders; all the other lines represent the performance of the neuron using dynamic learning rate and different sampling approaches. This is a typical example for this experiment. Among those curves representing dynamic learning rate and different sampling methods, the best one is produced using dynamic learning rate at first sight. However, looking at the area of our interest where $PCF(+) = 1/16$, we can find that at that point none of the sampling methods improves the performance of the classifier for class 16. The figure on right side enlarges the left bottom corner of the figure on the left, to give a closer look at the area of our interest.

In the experiment using either 16 vocal class dataset or 21 class mixed dataset, most auto-encoders and their corresponding neurons result in a similar picture to Figure 6.6.

Instead of using auto-encoders, we also use the set of 1-class SVM's as the base detectors for the stacked system. Similar to the case of auto-encoders, most of the classifiers do not benefit a lot from using dynamic learning rate and sampling rate. Figure 6.7 is an example of the set of cost curves constructed for neuron 16, with different sampling methods, using 16-artist dataset. Although for some of the classifiers, the curves generally get lower than the original one (see Figure

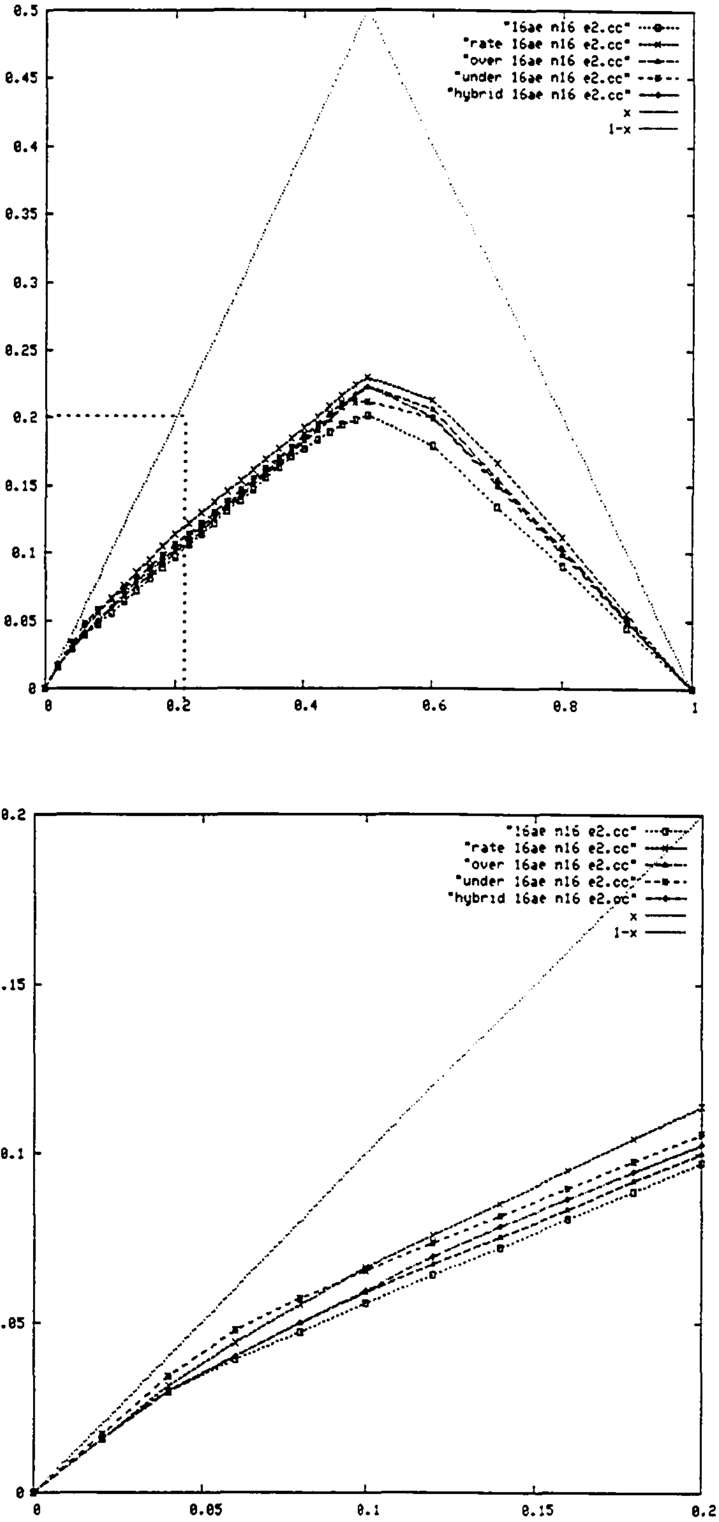


Figure 6.6: Cost curves for neuron and auto-encoder 16 using different sampling methods. The lower plot is the enlargement of the bottom left corner of the upper plot. The lowest cost curve is produced by the original training method.

6.7), when $PCF(+)$ is small, they are still close or even higher than the original curve, which means no essential improvement is gained. Dynamic learning rate results in a better cost curve than all the other sampling techniques, but is still worse than the original training method.

Figure 6.8 and 6.9 are cost curves constructed for neuron 16, trained using kNN's (k=1) and Naive Bayes detectors respectively, with different sampling methods, using 16-artist dataset.

Table 6.5 lists the results on the experiments we have conducted. Each number in the table indicates the number of classes for which a method produces the best cost curves for small $PCF(+)$ values. We can see that changing the learning rate or sampling method does not improve the performance significantly. Our analysis leads to the decision not to make use of dynamic learning rate or any special sampling methods.

| **Detector** | **Auto-encoder** | | | **kNN** | | | **SVM** | | | **Naive Bayes** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Dataset** | 16 | 21 | 6 | 16 | 21 | 6 | 16 | 21 | 6 | 16 | 21 | 6 |
| default | 8 | 13 | - | 10 | 15 | - | 9 | 13 | - | 5 | 5 | - |
| dynamic L-rate | 5 | 3 | - | 1 | 5 | - | 3 | 5 | - | 0 | 3 | - |
| over-sampling | 0 | 3 | - | 4 | 1 | - | 0 | 1 | - | 4 | 6 | - |
| down-sampling | 1 | 1 | - | 0 | 0 | - | 2 | 0 | - | 1 | 2 | - |
| hybrid | 2 | 1 | - | 1 | 0 | - | 2 | 2 | - | 6 | 5 | - |

Table 6.5: Summary of results for alternative training methods

### 6.3.3 Combinations of Different 1-Class Detectors

**Experiment:** Combinations of 1-class detectors.
**Options:**

1. only use one type of 1-class detector;
2. use two types of detectors;
3. use three types of detectors;
4. use all the four types of detectors.

**Conclusion:** using all four types of detectors improves the performance of each neuron the most.

In the previous section, we have used just one type of base detector at a time. In this section, we consider using combinations of different types of detectors. All possible combinations are considered.

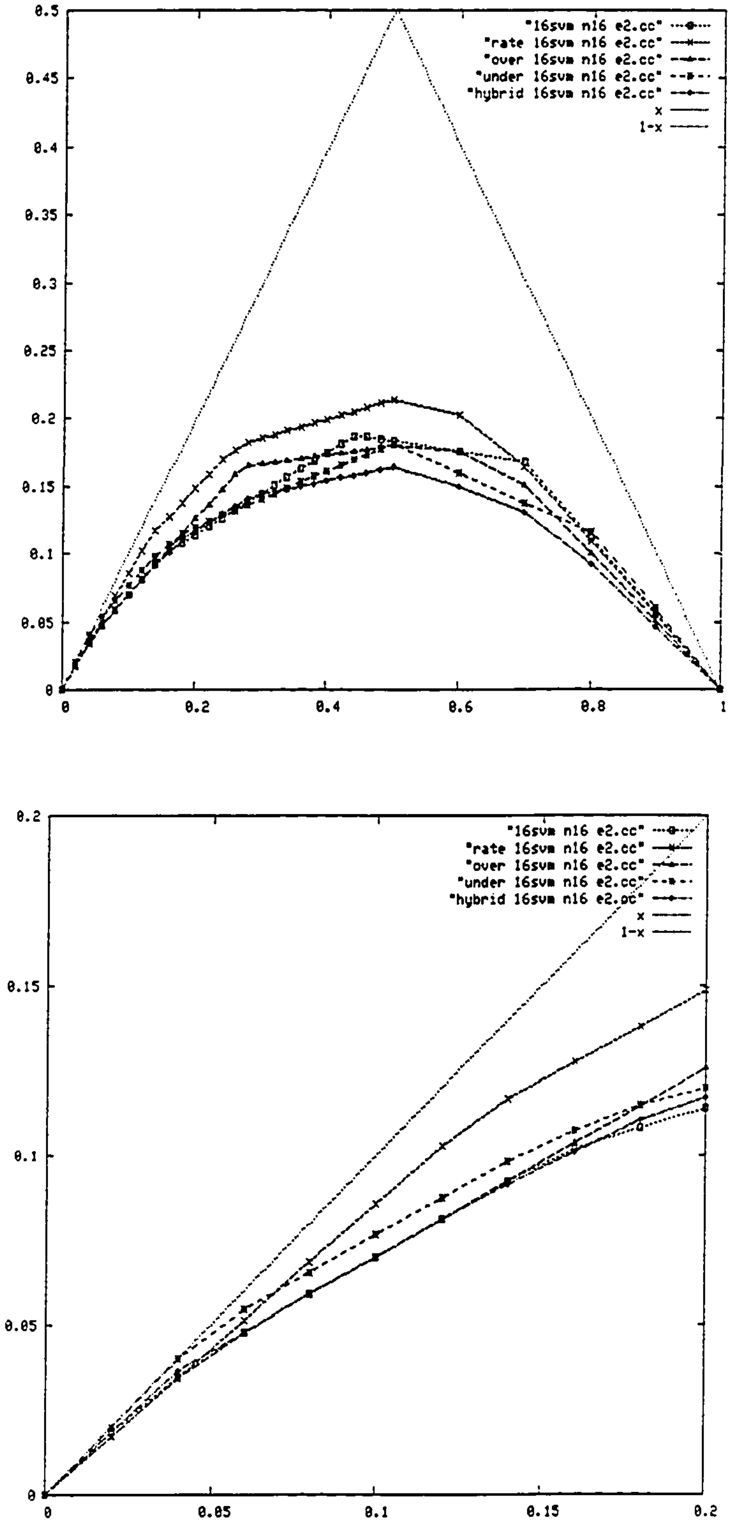


Figure 6.7: Cost curves for neuron 16, using SVM's, with different sampling methods. At the bottom left corner, the lowest curve is produced by the original training method.

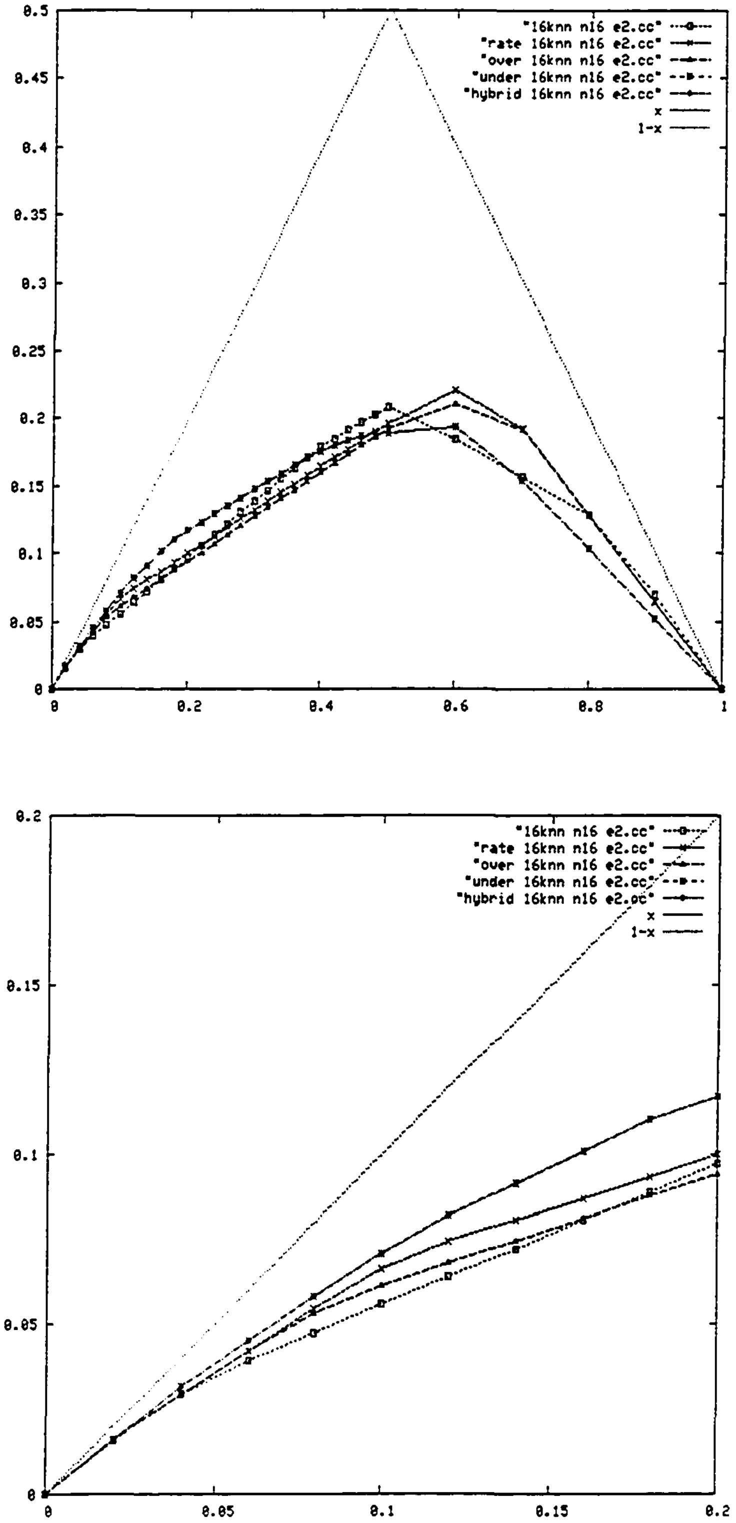


Figure 6.8: Cost curves for neuron 16, using kNN's with different sampling methods. At the bottom left corner, the lowest curve is produced by the orignal training method.

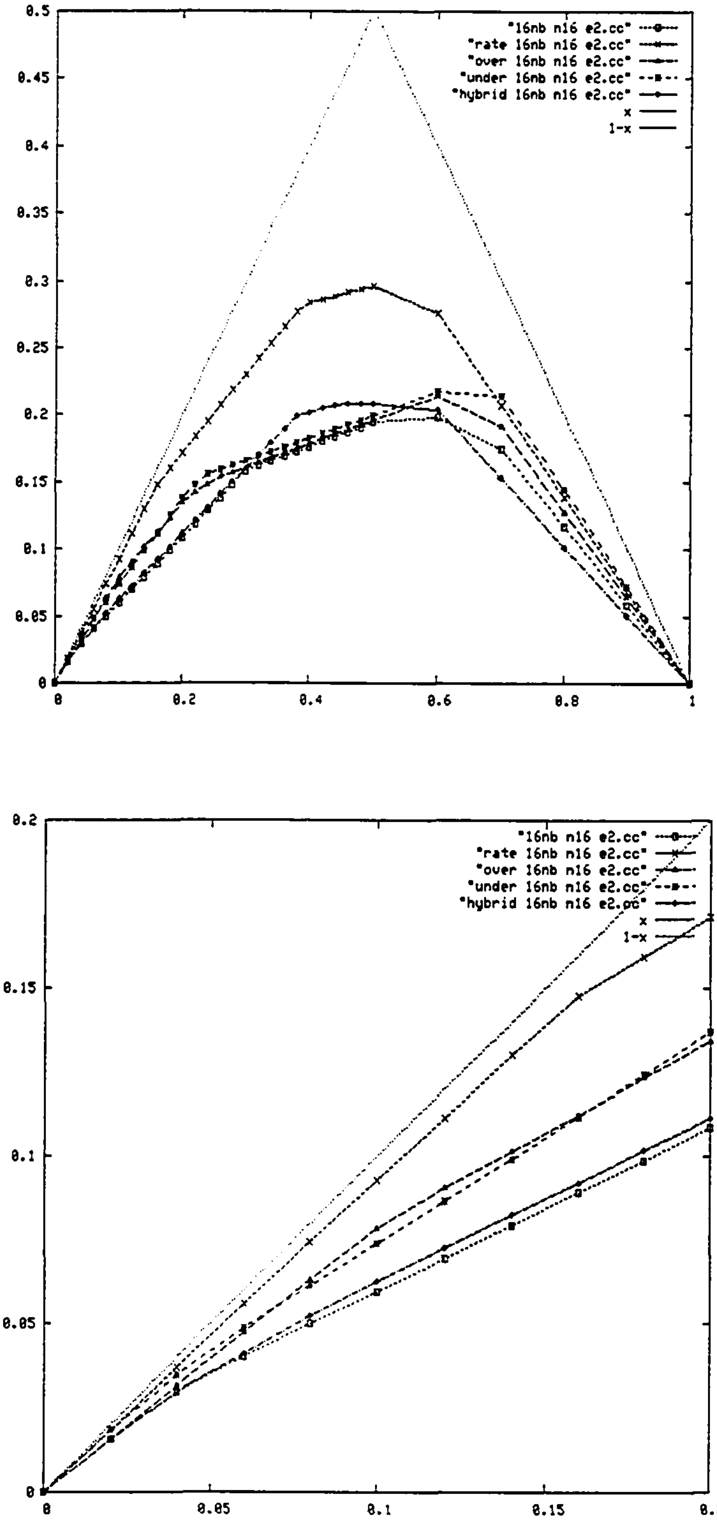


Figure 6.9: Cost curves for neuron 16, using Naive Bayes, with different sampling methods. At the borrom left corner, the lowest curve is produced by the original training method.

By comparing the cost curves constructed from these experiments, we conclude that using the whole ensemble at the bottom level produces the best 1-class classifiers as the performance of each neuron is improved the most.

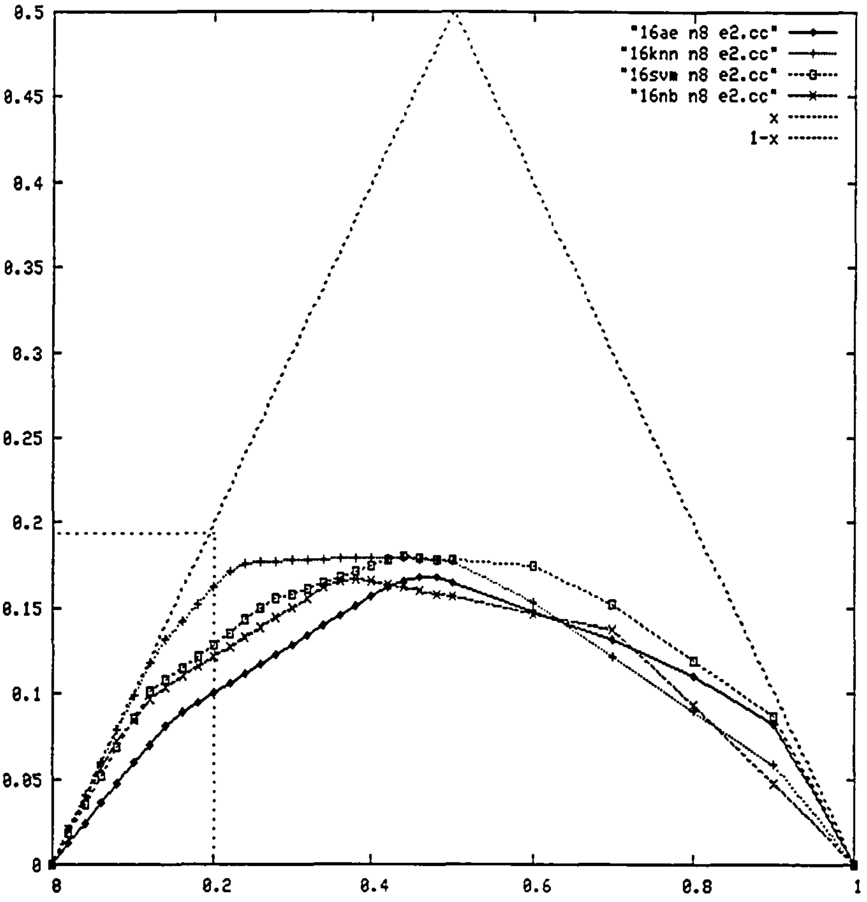


Figure 6.10: Performance of all the individual detectors

Figure 6.10 compares the four cost curves of neuron 8, by using the four different base level 1-class detectors. Table 6.6 lists the number of winning cost curves using each individual 1-class detector for each dataset. From the numbers in the table, we consider the auto-encoder as the best individual 1-class detector as it produces more winning cost curves than SVM and Naive Bayes and slightly more than kNN.

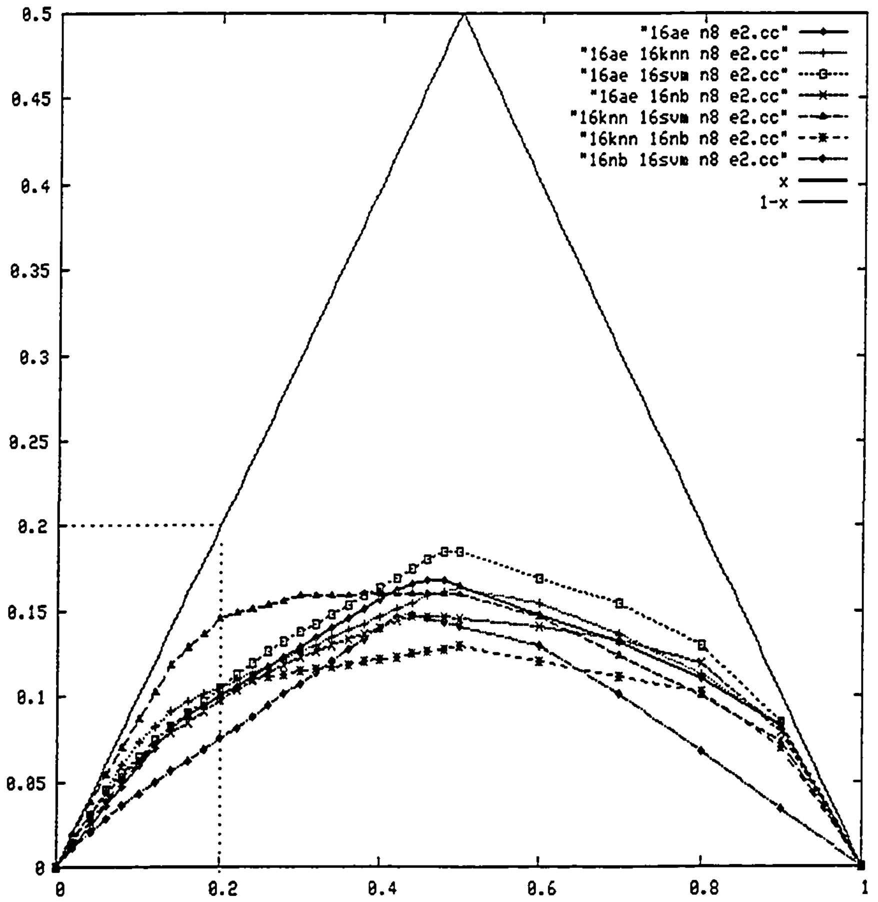


Figure 6.11: Best Individual vs. Pairs

| | Auto-encoder | kNN | SVM | Naive Bayes |
|---|---|---|---|---|
| 16 artists | 8 | 6 | 1 | 1 |
| 21 artists | 8 | 8 | 4 | 1 |
| 6 pianists | 3 | 3 | 0 | - |

Table 6.6: Using Individual Detectors

Figure 6.11 shows the cost curves of neuron 8 using auto-encoder only against using pairs of 1-class detectors. The auto-encoder is worse than some pairwise combinations of 1-class detectors. Table 6.7 provides a complete summary of the results of the comparison. Based on the table, we can see that in most cases, one or more pairs of 1-class detectors outperform the auto-encoder. Also, we consider the pair of auto-encoder and kNN as the best pairwise combination.

| | 16 artists | 21 artists | 6 pianists |
|---|---|---|---|
| AE | 3 | 2 | |
| AE&kNN | 6 | 8 | - |
| AE&SVM | 2 | 4 | - |
| AE&NB | 1 | 0 | - |
| kNN&SVM | 2 | 3 | - |
| kNN&NB | 2 | 4 | - |
| NB&SVM | 0 | 0 | - |

Table 6.7: Best Individual vs. Pairs

Now we compare the performance of the pair with all the triple and quadruple. In Figure 6.12, the curves from neuron 1, using the best pair, all the triplets and the quadruple combination are shown together. In this example, the cost curve for the best pair is in the middle, which means its performance is better than some triplets and worse than others. However, look at the bottom left corner, which is the area of our interest, the lowest curve is the one for the quadruplet. Therefore, in this figure, the winning combination is the quadruplet. The complete experiment results are recorded in Table 6.8. We conclude that by using more types of 1-class detectors, the performance of each 1-class classifier is no worse than using combinations of fewer types, although it is not always significantly better. In other words, the more information the neurons get, the better they learn to integrate the 1-class detectors.

| | 16 artists | 21 artists | 6 pianists |
|---|---|---|---|
| AE+kNN | 1 | 0 | |
| AE+kNN+NB | 3 | 2 | - |
| AE+kNN+SVM | 1 | 1 | - |
| AE+NB+SVM | 3 | 6 | - |
| kNN+NB+SVM | 2 | 1 | - |
| all 4 types | 6 | 11 | - |

Table 6.8: Best Pair vs. Triplets vs. Quadruplet

### 6.3.4 Picking up a Class

**Experiment:** 2 approaches to deciding the class label from the neural network outputs.

**Options:**

1. max-selection;
2. kNN;

**Conclusion:** kNN works slightly better then max-selection.

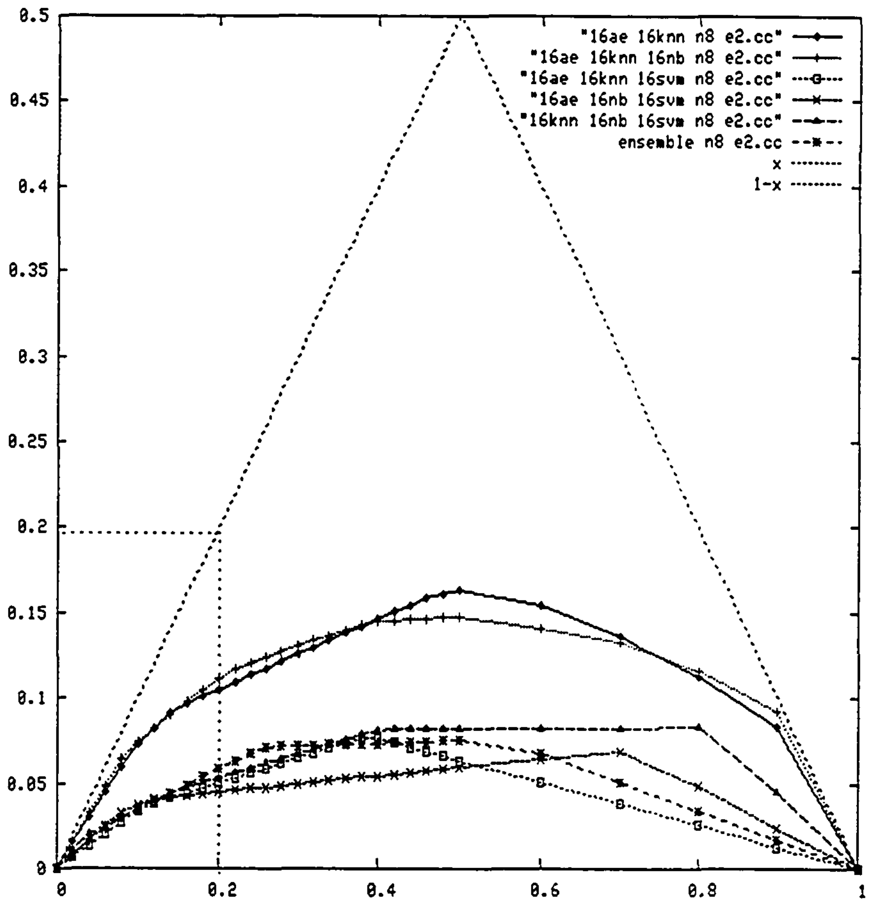


Figure 6.12: Best Pair vs. Triplets vs. Quadruplet

As the neural network at the top of our system outputs a set of values between 0 and 1, in order decide a class label for a testing example, the outputs from the neurons on top need to be thresholded or compared to each other. To do this, we tried two approaches.

First, we simply select the maximum among all the neuron outputs and classify the testing example with the class that neuron represents.

The second approach uses the kNN algorithm. For every example in the training dataset, the neural network outputs a vector of values $< v_1, v_2..v_n >$ where $n$ is the number of artist classes. During the training of the neural network, such vectors are recorded for all training example and they comprise a new training set $T$. When an unknown testing example comes, the kNN algorithm compares its output vector to each vector in $T$ and a class label is then assigned to the testing example.

It turns out that the max-selection and kNN approaches both work well. Table 6.9 and 6.10 show the confusion matrices of the classification results using the two approaches with the 16-artists dataset. Both of the matrices have relatively big number along the diagonal line, and small number elsewhere. In term of classification accuracy, kNN is slightly better max selection. kNN also produces relatively better precision and recall than kNN.

Based on these experiments, we decide to use max-selection approach as the strategy to pick up a class label.

### 6.3.5 Comparison

To compare our results with others, we conducted the classification on the 3 datasets. For the 16-artists and 21-artists datasets, we split the dataset into two parts, one for training, the other for testing, as described in [WFL01, KW02, BEL02]. The split is at the song level, i.e. the songs whose segments are in the training set entirely disjoint the songs whose segments are in the testing set. In order to get a class label for a complete song instead of a segment, the output for each segment of a song has to be combined by a summarizing rule. In our first approach, we tried adding the scores produced by each neuron for all the segments and pick the neuron that produces the maximal sum of the scores and the class of that neuron is assigned to the song. Our second approach is to pick the artist that has been selected most often when the individual segments are classified. The two approaches produce similar classification results. Table 6.11 lists our best segment level and song level classification accuracy and the results from others. Note that the length of frame (segment) in our work and in the other 3 systems are all different. Since the song classification results from the

| Prd⇒<br>Obs⇓ | a1 | a2 | a3 | a4 | a5 | a6 | a7 | a8 | a9 | a10 | a11 | a12 | a13 | a14 | a15 | a16 | $sum^{obs}$ | recall |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a1 | **7** | | | 1 | | | 1 | 2 | | | | | 2 | | 1 | | 14 | .500 |
| a2 | | **5** | | 2 | | 2 | | | | | 1 | | 1 | | | | 11 | .455 |
| a3 | | | **16** | | | | 1 | | | | 1 | | | | | | 18 | .889 |
| a4 | 1 | | | **10** | | | 1 | 2 | | | | | 1 | 3 | | | 18 | .556 |
| a5 | | 1 | | 1 | **7** | | 1 | | 1 | 2 | 2 | | 1 | 2 | | | 18 | .389 |
| a6 | | | | | | **5** | | | | | 2 | | 5 | 2 | | | 14 | .357 |
| a7 | 1 | | | 2 | 2 | 1 | **14** | | | 1 | 3 | | 1 | 1 | | 3 | 29 | .483 |
| a8 | | | | | | | | **10** | | | 1 | | | | | 1 | 12 | .833 |
| a9 | 2 | 1 | | | | 1 | | 1 | **2** | | 2 | 1 | 2 | 1 | 2 | | 15 | .133 |
| a10 | 1 | 1 | | | 2 | 1 | 1 | 1 | 1 | **8** | 1 | | 1 | 3 | | 1 | 22 | .364 |
| a11 | 1 | | 1 | | | | 1 | 3 | 2 | | **17** | | 1 | 1 | 2 | | 29 | .586 |
| a12 | | | | | | | | | 1 | | | **20** | | | | | 21 | .952 |
| a13 | 1 | | | 2 | 1 | 2 | 1 | | | 2 | 1 | | **10** | 1 | | 2 | 23 | .435 |
| a14 | | 1 | | | 1 | | 2 | | | | | | 2 | **10** | 1 | 2 | 19 | .526 |
| a15 | 1 | | | | | | | | | | 1 | | 1 | | **13** | 2 | 18 | .722 |
| a16 | | 1 | | 1 | | | | | | 1 | 1 | 1 | 1 | | | **8** | 14 | .571 |
| $sum^{prd}$ | 15 | 10 | 17 | 19 | 13 | 12 | 23 | 19 | 7 | 14 | 33 | 22 | 29 | 24 | 19 | 19 | **162** | |
| precision | .467 | .500 | .941 | .526 | .538 | .417 | .609 | .526 | .286 | .571 | .515 | .909 | .345 | .417 | .684 | .421 | | .549 |

Table 6.9: Confusion matrix for 16-class classification. Classifier built using auto-encoder. Class label decided through max-selection approach.

| Prd⇒<br>Obs⇓ | a1 | a2 | a3 | a4 | a5 | a6 | a7 | a8 | a9 | a10 | a11 | a12 | a13 | a14 | a15 | a16 | $sum^{obs}$ | recall |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a1 | **8** |  |  | 1 |  |  | 1 | 1 |  |  | 1 |  | 2 |  |  |  | 14 | .571 |
| a2 |  | **6** |  | 2 |  | 3 |  |  |  |  |  |  |  |  |  |  | 11 | .434 |
| a3 |  |  | **15** |  |  |  | 2 |  |  |  | 1 |  |  |  |  |  | 18 | .833 |
| a4 | 1 | 1 |  | **10** |  |  | 2 | 2 |  |  |  |  |  | 2 |  |  | 18 | .556 |
| a5 |  | 1 |  | 1 | **10** | 2 | 1 |  | 1 |  | 1 |  | 1 |  |  |  | 18 | .556 |
| a6 |  |  |  |  |  | **8** |  |  | 1 | 2 | 1 |  | 1 | 1 |  |  | 14 | .571 |
| a7 |  | 2 |  | 2 | 1 |  | **19** |  | 1 |  | 2 |  | 1 |  |  |  | 29 | .655 |
| a8 | 1 |  |  |  |  |  |  | **9** |  |  | 2 |  |  |  |  |  | 12 | .750 |
| a9 | 2 | 1 |  | 1 |  | 1 | 1 |  | **2** | 1 | 4 |  |  |  | 2 |  | 15 | .133 |
| a10 | 1 | 1 |  | 1 | 1 | 2 | 3 |  | 1 | **10** |  |  | 1 | 1 |  |  | 22 | .455 |
| a11 | 1 |  | 2 |  |  |  | 2 | 2 | 2 |  | **18** |  | 1 | 1 |  |  | 29 | .621 |
| a12 |  |  |  |  |  |  | 1 |  | 1 |  |  | **19** |  |  |  |  | 21 | .905 |
| a13 | 1 |  |  | 2 | 2 | 2 | 3 |  |  | 5 | 1 |  | **4** | 1 |  | 2 | 23 | .174 |
| a14 | 1 | 1 |  |  | 1 |  | 2 |  |  |  |  |  | 2 | **10** | 1 | 1 | 19 | .526 |
| a15 | 1 |  |  |  |  |  | 1 |  | 1 |  |  |  | 1 |  | **13** | 1 | 18 | .722 |
| a16 |  |  |  | 1 |  |  | 4 |  |  | 1 | 1 | 1 |  |  |  | **6** | 14 | .429 |
| $sum^{prd}$ | 17 | 13 | 17 | 21 | 15 | 18 | 42 | 14 | 10 | 19 | 32 | 20 | 14 | 16 | 16 | 11 | **167** |  |
| precision | .471 | .462 | .882 | .476 | .667 | .444 | .452 | .643 | .200 | .526 | .562 | .950 | .286 | .625 | .812 | .545 |  | .566 |

Table 6.10: Confusion matrix for 16-class classification. Classifier built using auto-encoder. Class label decided through kNN approach.

frames, we list the classification accuracy at the frame (segment) level as well.

| | Whitman et al. | | Kim et al. | | Berenzweig et al. | | Our results | |
|---|---|---|---|---|---|---|---|---|
| | frame | song | frame | song | frame | song | frame | song |
| **16-artists** | - | - | .307 | .415 | .356 | .683 | .337 | .489 |
| **21-artists** | - | .50 | - | - | .318 | .649 | .337 | .447 |

Table 6.11: Frame & Song Level Classification Accuracy Comparison on 16 and 21 artists datasets

The results show that our classifier outperforms the system in [KW02] by Kim et al., is close to the one in [WFL01] by Whitman et. al, but a lot worse than the classifier in [BEL02] by Berenzweig et al. By comparing our approach to [BEL02], the reasons could be that in [BEL02] the experiment was done using only voice segments for the 16 vocal artists. That can decrease the noise in the training and testing data. Because only voice segments were used for vocal artists, [BEL02] focused on voice features such as MFCC's and ignore other aspects of music. The short length of frames sliced for each song results in more training and testing frames than the segments we used in the experiments. The result for each short frame is more fine-grained than result for our 30 second segment.

For the 6-pianists dataset, we train and test the classifier on short segments from the pieces the same as [WZ04]. Our 1-class detectors can discriminate one class from all the others, while in [WZ04], classification is done for each pair of pianists. Specifically, the n-class problem was converted into $n(n-1)$ pairwise problems, one for each possible pair of pianists. For each pianist pair, the classifier was tested on 24 recordings (12 pieces, played by each of the two pianists). Thus, 24 predictions were made for each pair.

In order to do the comparison, we need to convert the overall accuracy into pairwise accuracies in [WZ04]. Our experiments are done in one-against-others manner for each 1-class detector. To get pairwise classification results, we analyze the outputs from each of the neuron pairs, tested on 24 recordings for each pair of pianists. Table 6.12 lists the classification accuracies of [WZ04] and our results. In [WZ04] 6 differert classifiers were trained and tested. In our experiment, we built our classifier using auto-encoder, 1-class kNN and 1-class SVM respectively; the best result presented in the table was produced using 1-class kNN as base detector. Our classifier performs better than 3 classifiers in [WZ04], but loses to the other 3 classifiers.

The different designs of the experiment make our learning and classification task tougher than the one in [WZ04] in that their pairwise classifier training only involves 2 classes each time, while the training of each 1-class detector in our system involves all the 6 classes. Therefore, there is much

| Pairs | c01 | c02 | c03 | c04 | c05 | c06 | ours |
|---|---|---|---|---|---|---|---|
| AS-DB | .667 | .583 | .625 | .667 | .667 | .625 | .583 |
| AS-GG | .583 | .667 | .708 | .708 | .583 | .542 | .625 |
| AS-MP | .625 | .708 | .750 | .667 | .708 | .667 | .625 |
| AS-MU | .583 | .583 | .625 | .667 | .458 | .542 | .500 |
| AS-RB | .625 | .708 | .708 | .708 | .792 | .792 | .708 |
| DB-GG | .667 | .625 | .708 | .750 | .583 | .542 | .708 |
| DB-MP | .667 | .667 | .667 | .625 | .625 | .625 | .667 |
| DB-MU | .625 | .667 | .625 | .625 | .583 | .542 | .625 |
| DB-RB | .708 | .792 | .792 | .792 | .708 | .708 | .708 |
| GG-MP | .625 | .708 | .667 | .708 | .583 | .625 | .625 |
| GG-MU | .667 | .708 | .667 | .750 | .542 | .583 | .750 |
| GG-RB | .500 | .625 | .708 | .708 | .583 | .667 | .542 |
| MP-MU | .500 | .625 | .542 | .542 | .542 | .458 | .583 |
| MP-RB | .583 | .667 | .833 | .833 | .542 | .625 | .583 |
| MU-RB | .583 | .708 | .667 | .708 | .583 | .500 | .583 |
| ave. | .614 | .669 | .686 | .697 | .606 | .603 | .628 |

Table 6.12: Song Level Classification Accuracy Comparison on 6 pianists dataset

more noise in our training data, which adds to the difficulty for classifier training. Despite this fact, our classifier still produced decent song level accuracies.

## 6.4 Summary

This chapter described the design of the experiments to find the best setting for our systems. It presented the results for each individual experiment, conducted on different datasets. It also compared our approach with previous work by other research on the same datasets.

# Chapter 7

# Conclusion & Future Work

Section 7.1 summarizes the contributions of this research. Section 7.2 discusses the limitations in our work, and section 7.3 provides some directions for future work, corresponding to these limitations.

## 7.1 Contributions

We have succeeded in building an artist identification system, designed as a multi-class classifier for identification of different music artists. We have described the architecture of the system, the strategies we used for feature extraction, our customized classifier building approach and classifier evaluation, and presented and evaluated the results of the experiments we designed. Our approach is novel in that we make use of 1-class learning algorithms to build classifier for each class and then linearly combine them by stacking a neural network on top of them to form the multi-class classifier.

We compared our classification results with others. For the 16 and 21 artists datasets, our classifier produced close to better results than two of the previous work [WFL01, KW02], but worse than [BEL02]. For the 6-pianists dataset, although we cannot compare our segment accuracy with the piece accuracy in [WZ04], from the segment level results, we can hope for close or even better classification results at the piece level.

In addition, since we designed our system in that way, the lack of examples from unknown artist is no longer a problem and it is feasible to enable reject option in the system as a potential add-on.

## 7.2 Limitations

As mentioned previously, features are a key component of any machine learning task. As described in Chapter 3, the features extracted from the 21-artists dataset for learning algorithm are produced by the Marsyas software. Although it provides us with a variety of different features, the quantity and quality of these features are dependent on the software. We may lose some other robust and stable features not used we use in our system. On the other hand, among the features we currently have, we can't tell which features benefit the learning and which do not. Neither do we know which features work the best together for the learning.

For the datasets themselves, there are some limitations. For example, for the 16 and 21 artists datasets, there are approximately 10 pieces for each artist. The number of examples for each artists is fairly small, and there are not many examples for us to explore how the difference between different albums by an artist might affect the learning and classification.

Another limitation of our work is the absence of *reject option* [Cho70]. As we designed, our artist identification system is supposed to not only classify artists it recognizes, but also reject those artists it never encountered in training. By training 1-class classifier for each artist class and combining them to perform final classification, our system can be developed to reject unknown artist when a testing instance is not accepted by any of those 1-class classifiers. To do this, the outputs of the multi-class classifier need to be thresholded so that rejections can be made based on the thresholds. We have not done much exploration and experimented on these ideas. This function unit is not included in the current system.

## 7.3 Future Work

For the feature selection issue, we will experiment with some existing feature selection algorithm, e.g., *Wrapper Model* [KG97]. By finding the most distinguishing feature subset from our data, we expect some improvement on the classification performance of the system.

Regarding reject option, some ideas about finding thresholds for linearly combined multiple classifier have been introduced in [FR01, RFV02]. Reject option will be some future work for our research.

Another interesting piece of future work would be experiments with the 6-pianists dataset using the the features describes in [SHSTW04]. In this paper, a set of advanced features are tested on the same dataset for the same learning task and the performance is better than the one reported in [SHSTW04]. We would like to train our classifier using the new features and compare our classification results to their system.

# Glossary

**ANN**

Artificial Neural Network

**FFT**

Fast Fourier Transform

**GMM**

Gaussian Mixture Model

**kNN**

*k*-Nearest Neighbors

**LPC**

Linear Prediction Coefficient

**MFCC**

Mel Frequency Cepstral Coefficient

**MIR**

Music Information Retrieval

**STFT**

Short Time Fourier Transform

**SVM**

Support Vector Machine

# Bibliography

[BEL02] Adam Berenzweig, Dan Ellis, and Steve Lawrence. Using voice segments to improve artist classification of music. In *Proceeding of 22nd International Conference on Virtual, Synthetic and Entertainment Audio*, pages 79–86, Espoo, Finland, June 2002.

[BPR$^{+}$] Phil Burk, Larry Polansky, Douglas Repetto, Mary Roberts, and Dan Rockmore. Music and computers. An interactive web-text. http://eamusic.dartmouth.edu/ book/.

[Cho70] C.K. Chow. On optimum error and reject trade-off. *IEEE Transactions on Information Theory*, 6:41–46, 1970.

[Coo87] James W. Cooley. How the FFT gained acceptance. In *Proceedings of the ACM conference on the history of scientific and numeric computation*, pages 133–140, Princeton, NJ, May 1987.

[CS95] Philip K. Chan and Salvatore J. Stolfo. A comparative evaluation of voting and meta-learning on partitioned data. In *Proceedings of 12th International Conference on Machine Learning*, pages 90–98, Tahoe City, California, USA, July 1995.

[DH00] Chris Drummond and Robert C. Holte. Explicitly representing expected cost: an alternative to ROC representation. In *Knowledge Discovery and Data Mining*, pages 198–207, 2000.

[DHS00] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification.* New York: John Wiley & Sons, 2000.

[FR01] Giorgio Fumera and Fabio Roli. Error rejection in linearly combined multiple classifiers. In *Proceedings of 2nd International Workshop on Multiple Classifiers Systems*, volume 2096, pages 329–338, Cambridge, UK, July 2-4, 2001.

[Har01] A. Harma. A comparison of warped and conventional linear predictive coding. *IEEE Transactions on Speech and Audio Processing*, 9:579–588, 2001.

[HCL03] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. A practical guide to support vector classification. Technical report, Department of Computer Science and Information Engineering, National Taiwan University, 2003.

[HGR04] Chao He, Mark Girolami, and Gary Ross. Employing optimized combinations of one-class classifiers for automated currency validation. *Pattern Recognition*, 37(6):1085–1096, 2004.

[Jap99] Nathalie Japkowicz. *Concept Learning in the Absence of Counter-Examples: An Autoassociation-Based Approach to Classification.* PhD thesis, Rutgers University, 1999.

[JL95] George H. John and Pat Langley. Estimating continuous distribution in bayesian classifiers. In *Proceedings of 11th Conference on Uncertainty in Artificial Intelligence*, pages 338–345, Montreal, Canada, August 1995. Morgan Kaufmann Publishers.

[JMG95] Nathalie Japkowicz, Catherine Myers, and Mark A. Gluck. A novelty detection approach to classification. In *Proceedings of 14th International Joint Conference on Artificial Intelligence*, pages 518–523, 1995.

[KG97] R. Kohavi and G.H.John. Wrappers for feature subset selection. *Artificial Intelligence*, 97:273–324, 1997.

[KW02] Youngmoo E. Kim and Brian Whitman. Singer identification in popular music recordings using voice coding features. In *Proceeding of 3rd International Conference on Music Information Retrieval*, pages 164–169, Paris, France, 2002.

[LM98] Huan Liu and Hiroshi Motoda. *Feature Selection for Knowledge Discovery and Data Mining.* Kluwer Academic, 1998.

[Log00] Beth Logan. Mel frequency cepstral coefficients for music modeling. In *Proceedings of 1st International Symposium on Music Information Retrieval*, Plymouth, MA, October 2000. http://ciir.cs.umass.edu/music2000.

[Mar03] Caren Marzban. A comment on the ROC curve and the area under it as performance measures. Technical report, Center for Analysis and Prediction of Storms, University of Oklahoma and Department of Statistics, University of Washington, 2003.

[Mit97] Tom M. Mitchell. *Machine Learning.* McGraw Hill, 1997.

[MMR$^+$01] Klaus-Robert Mullers, Sebastian Mika, Gunnar Ratsch, Koji Tsuda, and Bernhard Scholkopf. An introduction to kernel-based learning algorithms. *IEEE Neural Networks*, 12(2):181–201, May 2001.

[NJ02] A. Ng and M. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in Neural Information Processing Systems 14*, pages 841–848, Cambridge, MA, 2002. MIT Press.

[PFK98] Foster J. Provost, Tom Fawcett, and Ron Kohavi. The case against accuracy estimation for comparing induction algorithms. In *Proceedings of 15th International Conference on Machine Learning*, pages 445–553, Madison, WI, 1998.

[RFV02] Fabio Roli, Giorgio Fumera, and Gianni Vernazza. Analysis of error-reject rrade-off in linearly combined classifiers. In *Proceeding of 16th International Conference on Pattern Recognition*, volume 2, pages 120–123, 2002.

[Roy] Chris Roy. Short time fourier transform. from wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/STFT.

[Sar97] Warren S. Sarle. Neural network FAQ, 1997. ftp://ftp.sas.com/pub/neural/FAQ.html.

[Sch92] Robert J. Schalkoff. *Pattern Recognition: Statistical, Structural, and Neural Approaches.* Wiley, 1992.

[Sch00] E. Scheirer. *Music-Listening Systems.* PhD thesis, MIT Media Lab, 2000.

[SHSTW04] Craig Saunders, David R. Hardoon, John Shawe-Taylor, and Gerhard Widmer. Using string kernels to identify famous performers from their playing style. In *Proceedings of 15th European Conference on Machine Learning and 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 384–395, Pisa, Italy, September 2004.

[SPSTS01] Bernhard Scholkopf, John C. Platt, John Shawe-Taylor, and Alex J Smola. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13:1443 – 1471, July 2001.

[TD99] D. Tax and R. Duin. Data domain description by support vectors. In *Proceedings of 7th European Symposium on Artificial Neural Networks*, pages 251–256, Bruges, Belgium, April 1999.

[TEC01] George Tzanetakis, Georg Essl, and Perry Cook. Automatic musical genre classification of audio signals. In *Proceedings of 2nd International Symposium on Music Information Retrieval(ISMIR)*, pages 205–210, Bloomington, Indiana, USA, October 2001.

[Tza00] George Tzanetakis. Marsyas: A framework for audio analysis. *Organized Sound*, 4(3):169 – 175, 2000.

[Tza02] George Tzanetakis. Pitch histograms in audio and symbolic music information retrieval. In *Proceedings of 3rd International Symposium on Music Information Retrieval*, pages 31–38, Paris, France, October 2002.

[vR79] C. J. van Rijsbergen. *Information Retrieval.* Butterworths, London, UK, 1979.

[Wei] Eric W. Weisstein. Fast Fourier Transform. From Mathworld–A Wolfram Web Resource. http://mathworld.wolfram.com/FastFourierTransform.html.

[WF00] Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools with Java implementations.* Morgan Kaufmann, San Francisco, 2000.

[WFL01] Brian Whitman, Gary Flake, and Steve Lawrence. Artist detection in music with Minnowmatch. In *Proceedings of IEEE Workshop on Neural Networks for Signal Processing*, pages 559–568, Falmouth, Massachusetts, September 2001.

[WZ04] Gerhard Widmer and Patrick Zanon. Automatic recognition of famous artists by machine. In *Proceedings of 16th European Conference on Artificial Intelligence*, pages 1109–1110, Valencia, Spain, 2004. IOS Press.