University of Nevada
Reno

Ringermute: An audio data mining toolkit

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science
with a major in Computer Science.

by

Marcel Andrew Levy

Dr. Frederick C. Harris, Jr., Thesis advisor

December 2005

UNIVERSITY OF NEVADA RENO

THE GRADUATE SCHOOL

We recommend that the thesis
prepared under our supervision by

MARCEL ANDREW LEVY

entitled

Ringermute: An audio data mining toolkit

be accepted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

Frederick C Harris

Frederick C. Harris, Jr., Ph.D.,Advisor

[illegible]

Sergiu Dascalu, Ph.D. ,Committee Member

[illegible]

Sushil J. Louis, Ph.D. ,Committee Member

[illegible]

William L. Eubank, Ph.D. ,Graduate School Representative

Marsha H Read

Marsha H. Read, Ph. D., Associate Dean, Graduate School

December, 2005

Dedication

To Rachael, who made it possible,

and to Chaja, Sam and Max, who waited all their lives for it.

Acknowledgments

The completion of this thesis would not have been possible without Dr. Harris's support and advice. Dr. Dascalu renewed my interest in user interaction, and his kind words of encouragement were much appreciated. Dr. Louis provided the original challenge, which has turned out to be harder than I expected, that led to the Ringermute project. I also appreciate the cooperation of Dr. Eubank, who is helping me finish yet another degree.

Abstract

This thesis presents Ringermute, an application designed to support audio feature recognition and machine learning, from the training and testing to the deployment phase. By choosing from a combination of feature extraction routines provided by plug-ins, a researcher can quickly produce files for input to standard data mining tools. The best combination of feature-extraction and classifier plugins can then be used to drive a near-real-time application for further testing or production use.

Contents

List of Figures

Acronyms

ANN	Artificial Neural Network
ARFF	Attribute-Relation File Format
CASR	Computational Auditory Scene Recognition
CLAM	C++ Library for Audio and Music [2, 3]
DSP	Digital Signal Processing
EAC	Enhanced Autocorrelation
GUI	Graphic User Interface
HPS	Harmonic Product Spectrum
INRIA	The French National Institute for Research in Computer Science and Control
MARSYAS	Music AI Research SYstem for Analysis and Synthesis [84]
PIM	Personal Information manager
SOM	Self-Organizing Map
WEKA	Waikato Environment for Knowledge Analysis [88]

Chapter 1

Introduction

Context-aware computing and communication is a relatively recent area of research. The goal is to produce systems and applications that modify their behavior in response to changes in the physical or social environment of the primary user or users [75]. These systems consist of sensors and one or more controllers, which can operate according to behavioral models of variable complexity [42]. Context-aware computing can be seen as a marriage of machine learning with human-computer interaction.

Sensor design is one of several challenges in the field. Tasks that humans find intuitive (face recognition, for example) are still more difficult for machines and can only be done in a narrower range of circumstances, or force larger tolerances for error. Even recognizing the ring of a phone over the sound of conversation or music—a trivial task for most humans—is difficult for a machine [89, 45]. But such real-world sensors are crucial to providing machines with the same context that supports successful human interactions.

Most sensors, and particularly those that operate on audio or video, are given noisy and unpredictable input. While Digital Signal Processing (DSP) is a mature field of research, most of its algorithms are straightforward mathematical transformations and are frequently designed for human decision support, and do not necessarily obviate the need for further analysis. The problem is essentially a machine-learning problem, in that machines are faced with second-order input from DSP algorithms that varies in its complexity. For example, an office environment may be generally quiet and simple to segment into periods of distinct activity types, perhaps even on the basis of volume alone. However, a restaurant, factory floor or even the passenger compartment of an automobile present far more challenging environments. In these cases, spectrum analysis will get us only so far.

Voice recognition has long been a fruitful area of research and development, but audio sensing in support of context-aware systems is relatively less sophisticated. While data mining tools are plentiful, audio feature recognition is still an evolving area of research, and very few audio tools are designed with the needs of machine learning in mind. Even ignoring the vast majority of tools that are designed for music or multimedia production, most audio analysis tools are focused on DSP, and not machine learning tasks.

This thesis presents Ringermute, a tool designed to support audio feature recognition and machine learning, from the training and testing to the deployment phase. By choosing from a combination of feature extraction routines provided by plug-ins, a researcher can quickly produce files for input to standard data mining tools. The best combination of feature-extraction and classifier plug-ins can then be used to drive a near-real-time application for further testing or production use.

The problem of audio event and scene classification, and the current state of research, is summarized in Chapter 2. The intent and scope of the Ringermute system is discussed in more detail in Chapter 3, and the implementation is covered in Chapter 4. Two usage scenarios are outlined in Chapter 5. The results of the project and avenues for future research are found in Chapter 6.

Chapter 2

Background

2.1 Context-aware Computing

As computers have evolved from a job description to a ubiquitous and pervasive part of human life, their peculiarities and limitations have come into sharp focus, and their effects a serious concern. Where once an error in computation might see months to take effect, and pass through a number of human gatekeepers, the state of the art is such that computers are an active and immediate part of all major activities, including transportation, commerce, communication and even creativity itself. Soon after it became popular to speak of computers as "electronic brains," the limitations of this analogy became all too apparent. Far from replicating human intelligence and thought, computers serve, at best, as rapid idiot savants. While it is now equally popular to delineate the limits of computing and regale readers with the problems of software as it is practiced,[18] it is more useful to concentrate on the key phenomelogical differences between humans and computers. One such area is context, and the application of what is known as context-aware computing.

Although the field is relatively recent and fluid, Dey, Abowd and Salber [24] provide an excellent definition for the central property of context:

> Any information that can be used to characterize the situation of entities (i.e., whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves. Context is typically the location, identity,

and state of people, groups, and computational and physical objects.

This covers the topic, but sometimes examples are the best guide. One classic scenario is that of the urgent, time-sensitive notice that pops up when no one is there to see it. Such notices would be generated by such applications as diverse as personal information managers (PIMs) reminding someone of an appointment or an industrial control system indicating a change in coolant level. In either case, a human performing a similar function would immediately perceive that the message would have to be either deferred or delivered in an alternative way, depending on the context. In an industrial setting, there is a clear distinction between an event that would require immediate operator attention, one that would require information to be noted and collected for later analysis by an engineer [14].

In the first example, it may be that the user is already on their way to the appointment's location, in which case the notice might be deferred, or if the appointment is important enough, a text reminder could be sent to their cell phone. Similarly, if the appointment is to take place in their office, perhaps they forgot, and paging their cell phone is also in order if the appointment involves another person. Perhaps the user is in their office, but is simultaneously speaking on the telephone and typing into the computer. A pop-up message would seem intrusive at the moment. In this case, context involves the following facts about the user and their environment:

- Where is the user?
- What are they doing?
- What else is happening around them?
- What are their preferences in situations like these?

These facts are not always easy to come by for a computer. They may involve audiovisual sensors, knowledge about the user and their relationships to others gleaned from structured and unstructured data, and data collected from the past, particularly data the computer can use to correct its actions in the future. In the past, context would have been limited to what was provided such operating system modules as the mouse, keyboard, internal clock, the processes run by the user, and network activity. That the task is difficult does not mean it is out of reach: Context-aware notification systems supporting the typical office worker already exist in research settings [34, 47].

It is useful to note, at this point, that applications rarely, if ever, directly interact with humans. Standard software design practice is such that all such activity is mediated by the operating system itself. So it is only natural, when envisioning a new set of contextual information available to applications, to see this being provided by the operating system as well, or at least by services available to all processes. Dey, Abowd and Salber are clearly pointing toward this with their context-aware framework and toolkit [24]. The central problem of integrating a myriad of sensors, rules and inferences makes robust context-awareness not just a sensor problem or a machine learning problem, but a software design problem as well [32].

2.1.1 Affective Computing

A closely related subject is that of affective computing, which shares the goal of redefining the user interface in ways that better support human activities and human-computer interaction. Various interaction paradigms such as the command line and direct manipulation through a graphic user interface (GUI) have evolved from previously dominant paradigms, but continue to coexist [68]. It becomes apparent that, rather than having discovered the ideal means of interaction, we have found that our increasingly broad relationship with computers leads to an equally broad collection of interaction modes and paradigms. "There is rarely a one-size-fits-all solution for the growing variety of computer users and interaction [64] ." In affective computing, the concern is the emotional content of the interaction between the human and the computer [65]. While our expressions of frustration, short of violence, have little effect on current computers, affective interfaces would both recognize subtle and not-so-subtle human emotional cues, and react accordingly. Affective computing, like context-aware computing, implies both ubiquitous and attentive computing. It requires an environmental awareness in order to perform the higher-level task of deducing emotional states and possible responses. A good example of an affective application that makes use of context, and auditory context in particular, is Rea, a virtual real estate agent [13]. It recognizes both verbal and non-verbal cues and seeks to respect the natural human conversational conventions, including feedback and turn-taking.

2.1.2 Interruptibility And User Attention

Most computing devices and interfaces are in large part unaware of context, or do not fully make use of the available context information. One of the implications of this situation is that computer-initiated communication in the form of notifications and alerts will not always occur at opportune moments. One of the motivations for context-aware research is to create systems that take into account the limits of human cognitive abilities, in the sense of minimizing interruptions and their effect on task performance. Recent work has begun with the premise that human attention is a scarce resource, and that our cognitive abilities are fundamentally limited at any given moment, but limited in ways that allow for successful prediction even under uncertain conditions [35]. User studies have been able to quantify the effect of interruptions on users, including the increase in annoyance, anxiety and the perception of the task's difficulty [6, 7]. It is also apparent that, beyond the interruption itself, the nature of the interruption and the nature of the preexisting task, the timing of the interruption can, intuitively enough, be correlated to the intensity of the negative effect [1]. Through a series of studies Fogarty et al. [28] have demonstrated that effective models of human interruptibility can be constructed using simple, cheap and minimally intrusive sensors. Fundamentally, the problem of interruptibility can be seen as one of a trade-off between the cost of interrupting and the benefit of notification [53].

2.1.3 Some Context-Aware Applications

Speaking of what he terms “perceptual intelligence,” Pentland classifies context-aware applications in one of two categories: smart rooms and smart clothing [62]. The taxonomy should not be taken too literally: “Smart rooms” encompasses wide range of environments, ranging from typical office settings such as meeting rooms to vehicles as well. The key is that the context resides in the environment itself, which can be seen as a stationary robot [42], or at least stationary with respect to the user(s). Smart clothing, on the other hand, concerns the items, ubiquitous or not, that follow the user around. This includes such items as cell phones, personal digital assistants, and other more exotic devices. Perhaps a better, albeit more prosaic, set of terms would be *entity/environmental context* (the context of an environment or non-human entity) versus *personal context* (the context of a given person). Given a pervasive and ubiquitous computing landscape, these distinctions will of course tend to blur, if not disappear.

Environmental Context

The “intelligent” meeting room is the near-canonical example of a context-aware environment. This comes as no surprise given the complexity that accompanies the utility of presentation technology. Facilitating either a smooth presentation or collaboration is never easy, especially given the rise in meetings that involve geographically dispersed participants [19]. Even without the need for flexible state models [42], an organization benefits from a building that can determine which rooms need lighting and environmental control [27].

The Reactive Room project is a context-aware meeting room, with an integrated telepresence system [19]. Through the use of sensors (video, light) the interface to the system becomes invisible. Tasks such as lighting control and camera selection (for the remote attendees) are managed entirely by the system. In the SmartOffice project, researchers from the French National Institute for Research in Computer Science and Control (INRIA) used a combination of cameras and microphones to create an “intelligent” conference room capable of identifying occupants and their activities [44]. Though very similar to the Reactive Room project, SmartOffice goes further by incorporating gesture recognition, which transforms the whiteboard into an interface to the system. The Intelligent Room is also a meeting-room application, and Kulkarni[42] presents a solution to the problems of behavior and decision-making, problems which intelligent environments share with robots in general.

Personal Context

While environmental context concerns the behavior of a room or place given its context, personal context is the set of conditions and facts that influence the behavior of objects with respect to a single person, or that person’s identity and profile. Devices that fall in this category are concerned with assistance and communication in a particular sense. Pentland presents the analogy of a well-trained dog [62], and while he presents it with view toward smart environment, the comparison is apt here. One obvious component of personal context is physical location [74], which plays a large role in the issue of notification. Location can be determined using a number of schemes, ranging from specialized networks [74] to GPS or even wireless local area networks [80]. More recent work has moved beyond the issue of specific location toward the use of multiple sensors in different modes [5]. The goal is that devices can use their general context (a meeting inside the building vs. a walk outside, for example) to help determine their behavioral state. This would require the incorporation of context sensors and lower-level interpreters in a wide variety of devices, ranging from cell phones to coffee cups [32]. Mobile audio context has proven a difficult problem, although some results have been promising [73, 16]. Personal Information Managers and their ilk should also be seen in a personal context; projects have included the Notification Platform mentioned earlier [35] and the Syncophant project [47].

2.2 Auditory Context

Auditory context has been particularly interesting in mobile settings where a variety of environments permit scene segmentation. Clarkson, Sawhney and Pentland first identified the presence of sound objects or events (such as certain speakers or automobile engines), and then trained a Hidden Markov Method framework to classify scenes. They were able to detect nine out of 10 transitions accurately [16].

A somewhat different problem of a conversational topic (linguistic context) was the target of another study, which used speech recognition and machine learning to determine the conversational topic, with a 93 percent accuracy rate over three topics [37]. The team also wished to explore the possibility of classifying the emotional tone of the conversation.

In general, audio context in isolation has not been the focus of as much study as location-based context, particularly GPS-driven sensing. More than likely, auditory and visual context sensors would integrate into a system containing multiple sensors in varying modes [32].

There are significant problems to be solved before using audio sensors to detect either sound events or tackling the even more difficult problem of scene segmentation and recognition: even human subjects can have trouble detecting scenes by sound alone [60]. Even so, Computational Auditory Scene Recognition (CASR) has been a fertile ground for research. Early work attacked the problem of searching multimedia databases, particularly video [25, 92], and this is still a strong motivation [55, 49, 50]. Given the difficulty of the task, much of the research has been concerned with recognizing general classes of sound: speech, music, silence or applause, for example [55, 49, 50, 29]. Some approaches have attempted a two-stage approach of recognizing sound object patterns, which are then fed to a probabilistic network, such as Bayesian or a Hidden Markov Method [16, 48].

Although the research has grown out of the needs of multimedia databases, auditory context in its strict sense is becoming a topic of research as well. Aside from the work by Clarkson, Sawhney and Pentland [16] mentioned earlier, Korpipää et al. [41] have used audio features in the recent MPEG-7 standard [51] to generate a set of binary attributes that are then fed to a Bayesian network to infer context with a relatively high accuracy. Some work has also been done on the detection of alarm and notification sound objects, such as telephone rings and the like, in the presence of background noise or music [45, 26, 46].

2.2.1 Human Audio Perception

Before turning to the relevant details of audio signal processing and feature extraction for pattern recognition and machine learning, it helps to briefly cover what is known about the most successful system for recognizing and discriminating sounds and sound scenes: The human ear. The details of human hearing have been extensively covered elsewhere [78, 33, 79], but the important fact to note for auditory context research is that the basilar membrane in the inner ear acts as a frequency spectral analyzer and associates specific neurons with specific frequencies. The auditory cortex itself outnumbers the neurons directly attached to the basilar membrane by a factor of about a 1,000, making for a very complex system indeed [33]. In addition, the acuity of human hearing is frequency-dependent. While the range of human hearing extends from 20 Hz to 20 kHz, human hearing is more sensitive to the the narrower band of 1 kHz to 4 kHz, and can detect sounds at a much lower amplitude [79].

2.2.2 Audio Digital Signal Processing

Audio signals can be seen as a subset of the larger topic of digital signals in general. While sounds manifest themselves as waves in the physical medium of the atmosphere, they are represented digitally as sample points in the Cartesian coordinate plane, with the Y-axis representing amplitude and the X-axis representing time. An example can be seen in Figure 2.1.

Although some features can be extracted directly from the raw time-domain signal, most auditory pattern recognition and context inference has made use of either

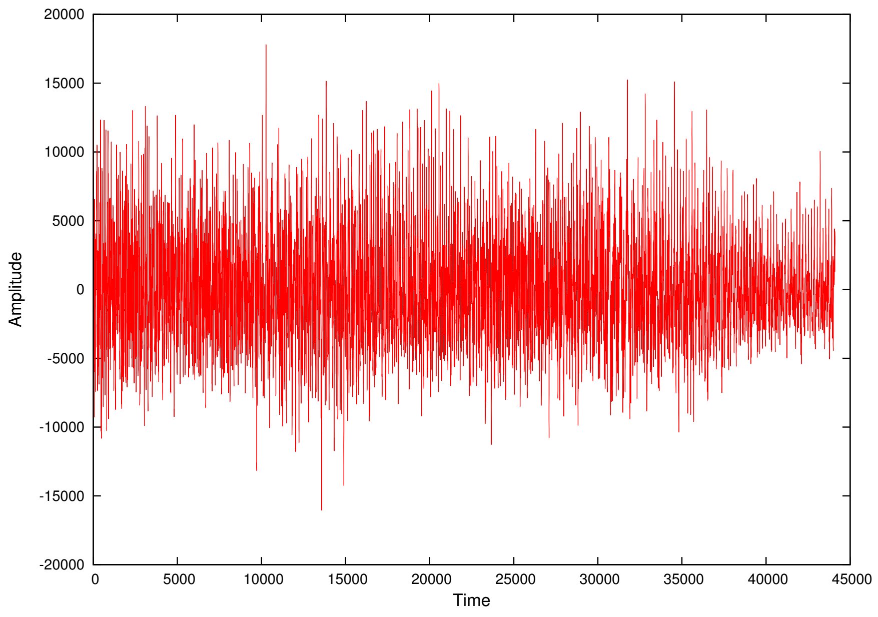


Figure 2.1: Example of the digital representation of a raw audio signal

the spectral data or features extracted from the spectrum. Derivation of spectral data is discussed in detail in Section 2.2.3. Studies have made use of either the standard spectrum [89, 46] or one adjusted to match the sensitivity profile of human hearing [16, 81].

2.2.3 Relevant Audio Features

In this section we will briefly discuss the various audio features that have been used in the past for audio scene recognition, speech/music discrimination, and other approaches to the general problem of audio segmentation. I have made extensive use of [60], [50] and [63] here, and it is interesting to note that there always seem to be at least two different ways to extract the same feature. In addition, this list of features is by no means exhaustive. These features are either already incorporated, or are planned for future inclusion in the Ringermute system. In reviewing the literature, one is struck by the amount of trial and error involved in auditory feature selection. They are chosen either because of their presumed emulation of human audio perception processes, or because of their correlation to phenomena in small samples. Since the goal of classification is to transform complex input into more simplified output, it would be interesting to explore automated means of feature generation and selection, such as the GA-based technique proposed by Vafaie and De Jong [85].

Short-time Energy Function

While the average energy over the short term of a given analysis window is not particularly discriminating for most audio object and scenes, its variation over the long term becomes more useful. It can be directly derived as the root-mean square of raw time-domain signal:

$$E = \sqrt{\frac{\sum_{i=0}^{N-1} x_i^2}{N}} \tag{2.1}$$

Frequency Spectrum

Although a few useful features may be derived from the audio signal without performing spectral analysis, most feature extracting will involve the spectrum as a starting point. As a first step, we transform the original time domain waveform into the frequency domain. An example of such a transformation can be seen in Figure 2.2.

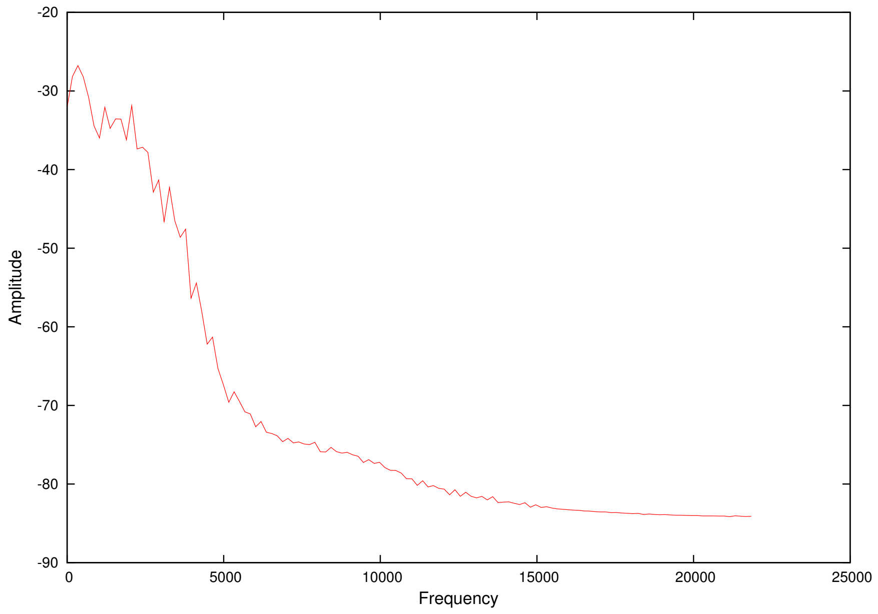


Figure 2.2: Example of the frequency spectrum of the audio signal from Figure 2.1.

We will briefly discuss the process, which is covered in greater detail in [33] and [79]. The Discrete Fourier Transform, or DFT, is our primary tool. This transformation takes the original audio signal vector $x[N]$ and produces the real part $ReX[N/2]$ and imaginary part $ImX[N/2]$. The real part contains the amplitude of the cosine waves, while the imaginary part contains the amplitude of the sine waves. We are primarily concerned with the real part, so the imaginary portion is discarded. We start with the DFT basis functions:

$$c_k[i] = \cos(2\pi ki/N) \tag{2.2}$$

$$s_k[i] = \sin(2\pi ki/N) \tag{2.3}$$

In Equations 2.2 and 2.3, c_k is the vector containing the cosine wave for the amplitude in $ReX[k]$, and s_k contains the sine wave for the amplitude in $ImX[k]$. The key to these functions is the frequency parameter k, which is equal to the number of complete cycles that occur over the N points of the original signal [79]. We pass over a considerable amount of derivation detail here, and proceed directly to the equations for producing the DFT:

$$ReX[k] = \sum_{i=0}^{N-1} x[i]\cos(2\pi ki/N) \tag{2.4}$$

$$ImX[k] = \sum_{i=0}^{N-1} x[i]\sin(2\pi ki/N) \tag{2.5}$$

To produce a given sample in the frequency domain, we multiply the original time domain signal by the particular sine or cosine wave sought, and then add the resulting points. Note that this describes the Real DFT, which is a simpler version of the complex DFT, which will not be discussed except to bring up the Fast Fourier Transform, or FFT, credited to Cooley and Tukey [17]. The FFT is the efficient algorithm generally used in spectral analysis to calculate the frequency domain. It is quite complex and will not be covered here, although it should be noted that the length of time required to calculate the DFT is usually a $O(N^2)$ operation; using the FFT, it becomes $O(N \log N)$.

It can be seen that the size of the initial frequency spectrum of the analysis window is dependent upon the length of the window. To facilitate analysis the initial spectrum may be converted by averaging into sub-bands. This hard limit places a lower bound on the size of the analysis window.

Mel-scaled Cepstral Coefficients

The Mel-scaled Cepstral Coefficients (MFCC) filter bank is an example of a feature extracted by applying more than one Fourier Transform to the original signal in sequence. The term *Cepstrum* (an anagram of spectrum) derives from this repeated use of the Fourier Transform. The MFCC feature seeks to emulate human perception through the following general procedure [33]:

1. Compute the power spectrum using the Fast Fourier Transform discussed in Section 2.2.3.
2. Apply the Mel-scale filter bank, which is based on human pitch perception.
3. Compress the spectral amplitudes to match human intensity and loudness perception.
4. Perform the inverse DFT.
5. Perform spectral smoothing, usually by ignoring the higher filter magnitudes.

This final representation is an approximation of the compressed and equalized signal produced by the mechanism of human hearing. Given the correlation, it makes an attractive feature for context-aware research [16], and has been found to be more discriminatory compared to other features mentioned here [60].

Spectral Centroid

This is the balancing point of the spectral energy distribution, and is related to the brightness of sound. It is the amplitude-weighted mean of the spectrum components. For relatively pure tones the spectral centroid will tend to equal the frequency of the tone, while more complex tones can be placed on a simple “brightness” scale. It can be calculated as follows:

$$SC = \frac{\sum_{k=1}^{N} kX[k]}{\sum_{k=1}^{N} X[k]} \tag{2.6}$$

Spectral or Signal Bandwidth

The spectral or signal bandwidth is the range of frequencies occupied by the signal—since noisy real-world signals generally measure *some* value in most or all the frequencies, a cutoff value equal to some fraction of the maximum value is usually employed. Peltonen [60] uses the following definition:

$$BW = \sqrt{\frac{\sum_{n=0}^{M} (n-C)^2 \cdot F(n)^2}{\sum_{n=0}^{M} F(n)^2}} \tag{2.7}$$

where C is the spectral centroid found in Equation 2.6, F is the DFT of the signal and M is the index of the highest-frequency sample greater than or equal to the cutoff value.

Spectral RMS

The root-mean square of the spectral data, defined as follows:

$$\sqrt{\frac{\sum_{i} = 0^N x(i)^2}{N}} \tag{2.8}$$

where N is the total number of frequency "bins."

2.2.4 Pitch Detection Features

Zero-Crossing Rate

The zero-crossing rate (ZCR) is the number of times the time-domain signal crosses the y-axis zero point for a given sample window. It can be used as a very rough pitch-detection attribute in that it depends on the frequency, but does not require calculation of the spectrum:

$$ZCR = \frac{1}{2N}\left(\sum_{i=1}^{N-1}|sgn(x(i)) - sgn(x(i-i))|\right) \tag{2.9}$$

In Equation 2.9 x is the original time-domain audio signal, N is the size of the processing frame, and the function *sgn* is defined as follows:

$$sgn(x) = \begin{cases} 1, (x > 0) \\ 0, (x = 0) \\ -1, (x < 0) \end{cases} \tag{2.10}$$

Lu and Zhang found the ZCR variation to be more discriminating than the ZCR value *per se* [50]. They define the high zero-crossing rate ratio (HZCRR) as follows:

$$HZCRR = \frac{1}{2N}\sum_{i=0}^{N} -1[sgn(ZCR(i) - 1.5avZCR) + 1] \tag{2.11}$$

where i is the audio frame index, $ZCR(i)$ is the zero-crossing rate at the ith frame, N is the total number of frames, *avZCR* is the average *ZCR* (they use a one-second window), and *sgn* is the sign function defined in Equation 2.10.

Harmonic Product Spectrum (HPS) Sub-bands

The harmonic product spectrum is one of several algorithms used to detect the fundamental frequency, or pitch, of a given audio signal. A fairly simple but robust method, the mechanics are covered in more detail by de la Cuadra, Master and Sapp [23]. In earlier stages of the Ringermute project the recognizer implemented a simple three-harmonic version of the HPS algorithm (base frequency plus two harmonics).

EAC Subbands

A slightly more complex pitch-detection algorithm than HPS, it was chosen after examining the audio samples in the open-source audio editor Audacity, which provides a pitch visualization using EAC [12]. Ringermute's algorithm is taken directly from the Audacity code, which in turn is based on the enhanced autocorrelation algorithm described by Tolonen and Karjalainen [83]. The algorithm seeks to duplicate the characteristics of human pitch perception, but does so in a computationally efficient way. As with the spectrum, the spectral output from EAC is condensed to a user-configurable number of sub-bands.

2.3 Machine Learning and Context

Coping with noisy floating-point data is one of the critical issues for the use of machine-learning algorithms in audio research. One of the central problems of noisy data is that the most effective approaches for segmenting and classifying may not actually give us a great deal of insight into the problem and its solution—all we can say is that the approach worked for a particular problem set. Since these techniques are all fundamentally machine learning techniques, it is implied that their effectiveness also depends entirely upon the choice of the training set and composition of the feature vectors. That being said, all of the following techniques lend themselves to some form of visualization and generalization, given that they effectively solve the problem.

Before discussing some of the approaches and algorithms, it would help to define the terms we will be using. We will start with Mitchell's formal definition:

> A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E [54].

Relating the definition to context-aware audio, E is a collection of *exemplars* consisting of fixed-length windows of audio data and the features derived from them. Each exemplar can be seen as a vector of *features* or *attributes.* For our purposes, a feature might be a vector of spectrum values, the zero-crossing rate, or average energy across the audio window. The exemplars in the training set R and test set S (where

$R \cup S \subseteq E$) are given expected values in order to evaluate the learning algorithm's ability to discern the *target function,* which is the function that correctly determines the correct values for all exemplars in E.

While we cover the major machine-learning and pattern-recognition approaches in the field, it should be noted that research has gravitated toward hybrid approaches using some form of prior classification or segmentation, perhaps one suited to noisy data, which is then fed to another approach, which may excel in classifying vectors of discrete attributes [47, 41]. Some research has also focused on improving feature selection and makes use of heuristic rules [92, 81, 93].

2.3.1 Training and Validation

Aside from the problem of feature selection and/or generation, machine learning presents the challenge of validation: That is, the problem of determining whether the combination of features and classifier would be effective on any data other than the training set, and the problem of evaluating one classifier/feature combination against another. The problems of system evaluation, and solutions developed in response, is covered in more detail by Theodoridis and Koutroumbas [82] and Witten and Frank [91]. Stratified tenfold cross-validation is one of the more popular techniques, and the one that is used by the WEKA machine learning toolkit by default [88]. With this method, the available dataset is randomly split into 10 equal-sized "folds." Nine of the folds are used to train the classifier and the remaining fold is held back for testing. This process is repeated until each fold has been used for testing. The entire cross-validation process can be repeated to guard against bias introduced by the process of creating the folds [91].

2.3.2 Supervised Training

The primary difference between supervised training and unsupervised training (discussed in Section 2.3.3), is that supervised algorithms seek to exploit *a priori* information, in the form of a training dataset composed of pre-classified instances. While this may seem an obvious advantage over unsupervised-training algorithms, it does mean that improperly-classified data will skew the results and increase the trained classifier's error rate on future data.

Decision Trees

Decision Trees, such as the ID3 algorithm [70] and its descendants C4.5 [71] and J48 [91], work by using the training set to construct a binary tree whose nodes consist of decisions that are applied to new exemplars. Figure 2.3 demonstrates part of a decision tree constructed from audio features. Sub-band 13 and Sub-band 15 are bands at the high end of the frequency spectrum, and the decision tree nodes are evaluating exemplar attribute values in those bands. The tree itself is constructed

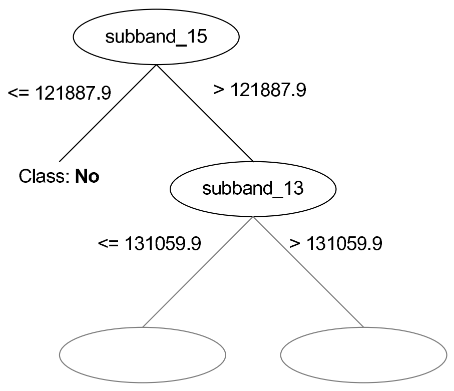


Figure 2.3: Example of a decision tree constructed from audio data.

in a top-down fashion by choosing the most discriminatory attribute at each level on the basis of its information gain—that is, how effective it is in classifying exemplars on the basis of that attribute alone. This is covered by Quinlan [71] in more detail, but a brief discussion taken from Mitchell [54] follows. We start with the idea of information entropy, or uncertainty, as originally advanced by Shannon [76]. The entropy measurement for a boolean classification is fairly simple:

$$Entropy(S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus} \tag{2.12}$$

where $p_\oplus$ is the proportion of positive examples in S and $p_\ominus$ is the proportion of negative examples. It can be seen from Equation 2.12 that the entropy increases as $|p_\oplus - p_\ominus| \to 0$. As the entropy approaches 1, its discriminatory power decreases. The entropy is used to derive the *information gain* of an attribute via the following calculation:

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{S_v}{S} Entropy(S_v) \tag{2.13}$$

In Equation 2.13, S is the collection of examples, S_v is the subset of S where attribute A has value v and $Values(A)$ is the set of all possible values for the attribute A. Decision tree algorithms are biased toward simpler and thus more comprehensible trees, and this information-gain metric tends to prevent excessive branching [91]. Even so, it is usually necessary for the algorithm to "prune" the tree after building it, removing subtrees that have little effect on the overall classification accuracy [69]. Decision tree-pruning and other methods of keeping the number of nodes small also helps in combating the problem of *overfitting,* where the classifier increases its accuracy on the training data at the expense of the test data. Very little auditory-context work has been done with decision trees, although some of my earlier research was promising [45]. The use of decision trees is more established in more general context-aware research, particularly where boolean attributes can be employed [47, 28].

Hidden Markov Models

Since one of the main goals of auditory scene and object recognition is to find an abstract decision function within a *time series* of noisy real-world data, Hidden Markov Models are an obvious choice [16, 94, 15]. Reduced to its essence, a HMM is a finite state automaton with probabilistic transitions. They are covered in more detail by Theodoridis and Koutroumbas [82], but we can explain HMMs by way of an example auditory scene problem. If we were to attempt to determine whether a phone was ringing in an office, we would construct a model with the states N and P, where P represents the state where the phone is ringing. These are, in fact, the "hidden" states, in that they are not directly observable to the system—all it can observe is the various audio features gleaned from the microphone. We can establish rough initial probabilities for these states, of course, and we can also estimate transition probabilities—the likelihood of a change from N to P or P to N at any given time. These initial probabilities will, of course, be altered in the process of training, which usually is done using the iterative Baum-Welch algorithm [8].

Artificial Neural Networks

Artificial Neural Networks (ANNs), or multi-layer perceptrons, hold quite a bit of promise for auditory classifiers due to their robust performance with noisy data [54]. Biologically inspired, ANNs are a necessary simplification: Neurons and synapses are simulated using a highly-connected directed graph. The nodes contain real values used to weight the input to the node, the output of which is then used as an input for the next node(s). The hypothesis space H is the set of all possible weights, and the target function is iteratively approximated, commonly by using the gradient-descent algorithm Backpropagation. For each training exemplar, Backpropagation propagates the input forward through the network, and then propagates the error adjustment backwards through the network. Other methods can also be used to calculate weights for the network, such as one using a genetic algorithm [89]. Figure 2.4 shows a three-layer perceptron—an ANN with two hidden layers. It can be shown that a network with two hidden layers can classify certain input spaces that are impossible for a network with a single hidden layer [82]. One problem faced in training ANNs, which they share with decision trees, is that of *overfitting* to the training set [54, 90]. Under normal circumstances it will be seen that the classification error for both training and validation sets will decrease, but as the number of training iterations approaches ∞ the validation error will increase while the training error will continue to decrease. At this point the ANN is adapting itself more perfectly to the training set, but is losing its ability to classify unseen exemplars. Given their robust handling of noisy continuous data, it is no surprise to see the use of ANNs in every-

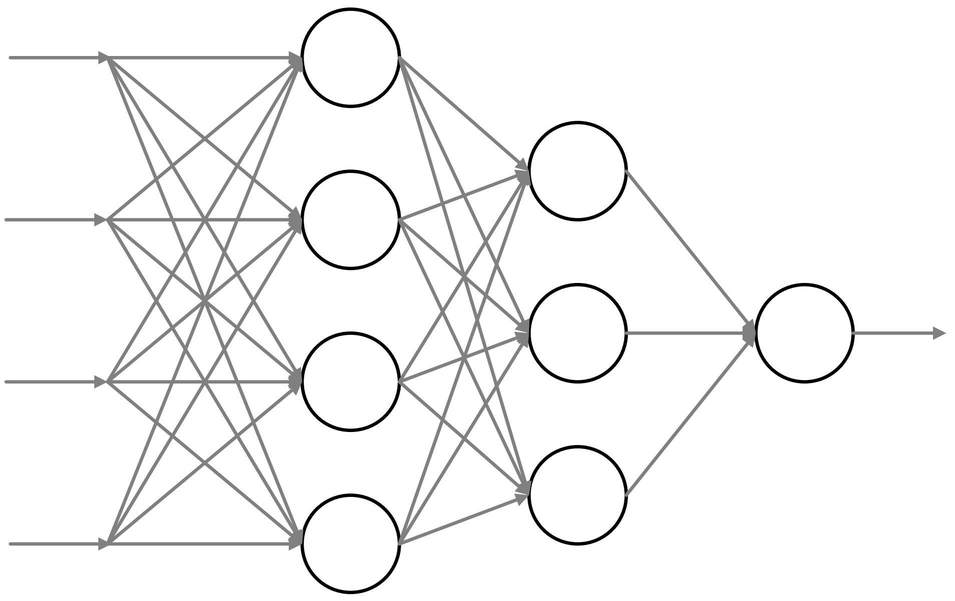


Figure 2.4: Example of an artificial neural network with two hidden layers

thing from the analysis of multiple simple sensors [43], detection of alarm sounds [26] or the automatic segmentation of speech and music [38].

Bayesian Learning

The use of Bayesian learning methods in machine-learning applications has risen since its widespread adoption in junk-email filters [72]. While Bayesian methods perform well in text-based applications [37], they are not limited to that domain. Auditory classification attempts have made use of Bayesian methods such as a naive Bayes classifier [41] or Bayesian belief network [48] after a preliminary classification step, which might simply consist of discretization of the initial continuous values. Bayesian classifiers have also been used in research into human interruptibility, and tend also to be used after initial discretization [34, 47, 28]. Both naive Bayes and Bayesian Belief Network classifiers start with the venerable (and posthumous) Bayes theorem [54, 9]:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)} \tag{2.14}$$

In Equation 2.14, $P(h|D)$ is taken to mean the probability of the hypothesis h given the observed training data D, where $P(x|y)$ is used to denote the probability of x

given y. In practice, the denominator $P(D)$ is discarded because it is a constant independent of h [54]. In Bayesian classifiers, the goal is to find the most probable hypothesis or class from a set of classes, also known as the *maximum a posteriori* (MAP) hypothesis, defined as $h_{MAP} \equiv argmax_{h \in H} P(D|h)P(h)$ [54]. Given a set of exemplars consisting of a vector of features $\langle a_1, a_2 ... a_n \rangle$, we can describe the process of finding the most probable class v_{MAP} like so:

$$\begin{aligned} v_{MAP} &\equiv argmax_{v_j \in V} P(v_j | a_1, a_2 ... a_n | v_j) \\ &\equiv argmax_{v_j \in V} \frac{P(a_1, a_2 ... a_n | v_j) P(v_j)}{P(a_1, a_2 ... a_n)} \\ &\equiv argmax_{v_j \in V} P(a_1, a_2 ... a_n | v_j) P(v_j) \end{aligned} \tag{2.15}$$

In Equation 2.15, we are simply casting Equation 2.14 into the vector-of-features problem domain, where v_j is the particular class in the set of classes C. Note, however, that $P(a_1, a_2 ... a_n | v_j)$ requires that we have an extremely large set of training data (on the order of $n!$) in order to come up with reasonably accurate estimates. To simplify, the naive Bayes classifier assumes that each individual attribute is independent of the others, so that $P(a_1, a_2 ... a_n | v_j)$ is the product of the individual attributes' probabilities:

$$v_{NB} \equiv argmax_{v_j \in V} P(v_j) \prod_i P(a_i | v_j) \tag{2.16}$$

While naive Bayes classifiers assume the independence of all attributes, Bayesian belief networks remove this absolute constraint by allowing that this assumption only applies to certain subsets of the variables. The removal of this constraint allows such classifiers to tackle more complex problems, while preserving some of the efficiency found in naive Bayes classifiers [54].

2.3.3 Unsupervised Training

When no prior classification is available for the training data, it is still possible to discover similar classifications in the input data. This process, *clustering,* is useful in auditory analysis due to the difficulty of properly classifying the training examples. For example, although it is possible to manually locate the beginning of a sound object by viewing the waveform visualization of a file in a sound editor, the very act of labeling might introduce errors by classifying the absolute a priori onset of the sound object, which may not contain the feature values useful for programmatic classification. Using a clustering algorithm, then, allows us to discover what groups occur naturally in the data: "These clusters presumably reflect some mechanism at work in the domain from which instances are drawn [91]."

k-Means Clustering

Also known as isodata clustering, this is a fairly simple and popular technique when the number of clusters is known. Given a set of M classes: $\theta_j, j = 1, ...M$, the N exemplars can be clustered using the algorithm seen in Figure 2.5, taken from Theodoridis and Koutroumbas [82]:

```
repeat
    for $i = 1$ to $N$ do
        Determine the closest representative $\theta_j$ for $x_i$
        Set $c(i) = j$
    end
    for $j = 1$ to $M$ do
        Determine $\theta_j$ as the mean of vectors $x_i \in X$ with $c(i) = j$
    end
until no change in $\theta_j$'s occurs in successive iterations.
```

Figure 2.5: Algorithm for k-Means Clustering

k-Nearest-Neighbor

The k-Nearest-Neighbor (kNN) algorithm is a perfect example of an instance-based learning algorithm, meaning that instead of using the training data to create a general, explicit description, the training data is used, either in part or in its entirety, to classify exemplars. We will briefly discuss it here, but it is covered in more detail by Mitchell [54]. By storing previously-encountered data, it is able to defer computation until the classification step. This does mean that the classifier routine itself is somewhat expensive. However, it is robust for complex target functions. The kNN algorithm boils down every learning problem into one of simple geometry, assuming that n-dimensional space can be considered simple.

We start by defining the problem space as an n-dimensional space $\Re^n$. The target function can be either discrete ($f : \Re^n \to V$, where V is defined as a finite set) or continuous ($f : \Re^n \to \Re^n$). Each instance x is defined as a feature vector: $\langle a_1(x), a_2(x), ...a_n(x)\rangle$. When a new, unclassified instance is encountered, it is compared to the existing data set using a Euclidean distance function:

$$d(x_i, x_j) \equiv \sqrt{\sum_{r=1}^{n}(a_r(x_i) - a_r(x_j))^2} \tag{2.17}$$

Note that the Manhattan or city-block distance, where the differences between x_i and x_j are merely summed, could be used as well, instead of the root-mean-square distance shown in Equation 2.17. It is also common to normalize the attribute values to avoid skewing the distance function with varying scales [91]. The new instance is usually compared to at most k nearest neighbors, and is frequently weighted by distance. It also is possible to make a *global* comparison to all previously-classified instances by using distance-weighting. It can be shown that classification is optimal as $k \to \infty$ [82], but it is common practice to use $k = 5$ [54, 91]. KNN-based approaches have been quite successful depending on the feature selection [73], and are frequently used in combination with secondary classifiers [49, 50].

Self-Organizing Maps

Influenced by research into ANNs, Kohonen self-organizing maps (SOMs) can be seen as a specific type of competitive learning scheme, where the goal is to create clusters by iteratively moving similar exemplars toward each other. Each new exemplar is compared to the set of existing exemplars, and the most similar existing exemplar is then moved within the hypothesis space toward the new exemplar, while the other exemplars stay where they are or are moved very little. Although it is not necessary to have classified the exemplars from the training set, the algorithm does require some knowledge of the number of clusters [82]. Kohonen describes a SOM more formally as a mapping of non-linear and high-dimensional input data (in our example, the audio input and features) onto the elements of a regular and lower-dimensional array [40]. By way of analogy, the SOM is an unsupervised version of the ANN discussed earlier.

Chapter 3

Ringermute Design

While the world certainly does not lack for excellent audio tools [12, 59, 10] or machine-learning applications [91, 88], there does exist a need for an application that:

- Allows researchers to apply machine-learning techniques to live and stored audio data without having to first learn how to use audio and digital signal processing libraries.
- Allows researchers to develop new feature-extraction or classifier plug-ins that are based on existing code, without having to learn much about the preexisting code.
- Allows the rapid creation of an application based on the results of experimental data, without even requiring compilation.
- Is flexible enough to allow further modification and additions.

It may seem that a tool such as MATLAB [52] would suffice to perform research on audio context problems. Although such tools are useful and have a place in the research, their primary limitation is that they were not designed to deliver usable applications in an interactive context, particularly for live audio. And while WEKA's tool set [88] is well-adapted to constructing cross-platform applications, the audio framework and feature selection is up to the researcher to provide. Yet the problems of audio input and sound file formats are not so complex that they necessitate re-inventing the wheel. And the fields of auditory context and CASR provide a rich set of features stable enough to be considered standard as well. The intent is to allow the rapid creation of audio context widgets as conceived by Dey, Abowd and Salber, either at the sensor or Interpreter level [24].

3.1 The Framework

The functionality of Ringermute is contained in three applications: The service (`rimuservice`), the status monitor (`rimutaskbar`) and the feature-extractor (`rimuextract`). `rimutaskbar` serves as the user's GUI interface to the system. It allows the user to view, edit and save settings, start or stop `rimuservice`, and start `rimuextract` within a GUI context. It also displays the current `rimuservice` activity state. `rimuservice` is the engine that is responsible for acquiring the raw audio data (either from hardware input or a sound file), and activating data-processing modules (called Listeners) on the data in turn. The service is also tasked with writing out any data to a file or files. `rimuextract` is a command-line tool that extracts features from a series of audio files and combines them into a single ARFF file. The relationship between the three applications is seen in Figure 3.1.

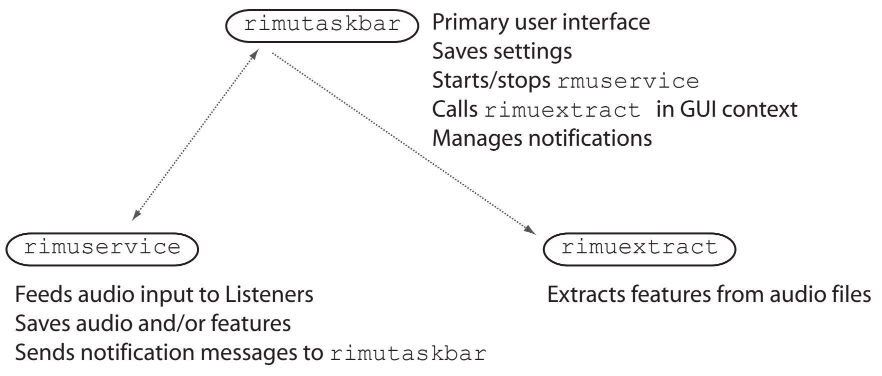

rimutaskbar
Primary user interface
Saves settings
Starts/stops rmuservice
Calls rimuextract in GUI context
Manages notifications
rimuservice
Feeds audio input to Listeners
Saves audio and/or features
Sends notification messages to rimutaskbar
rimuextract
Extracts features from audio files

Figure 3.1: Ringermute components and their interaction.

3.2 The Central Repository

The key data structure is the central repository, also called Ringermute Central. At its core it is a hash table [20] of pointers to objects containing data (usually arrays of various types). Ringermute service places each frame of audio data in the repository for use by the Listeners, which are responsible for managing the data in their own namespaces. If a Listener requires historical data (the last N frames, for example), it is responsible for keeping this data as well. The Ringermute service only promises to provide the original audio data, and to trigger the Listeners whenever the data changes.

3.3 Listeners

All data-processing and feature extraction(except for the original raw audio data) is performed by Listeners. A set of standard Listeners is provided with Ringermute, and provides basic audio processing, including spectral analysis. But the design of Ringermute is such that third-party modules could easily be written to replace the basic functions. As can be gathered by the description, Listeners are a relatively straightforward implementation of the Observer pattern [31, 30]. A graphic representation of the Listener interface can be seen in Figure 3.2. Another key point to make is that Ringermute Listeners are designed to be loaded as dynamic plug-ins at runtime. As such, they are required to provide several informational routines that tell Ringermute what data they provide, and what data they depend on. This helps Ringermute determine in which order to alert the Listeners.

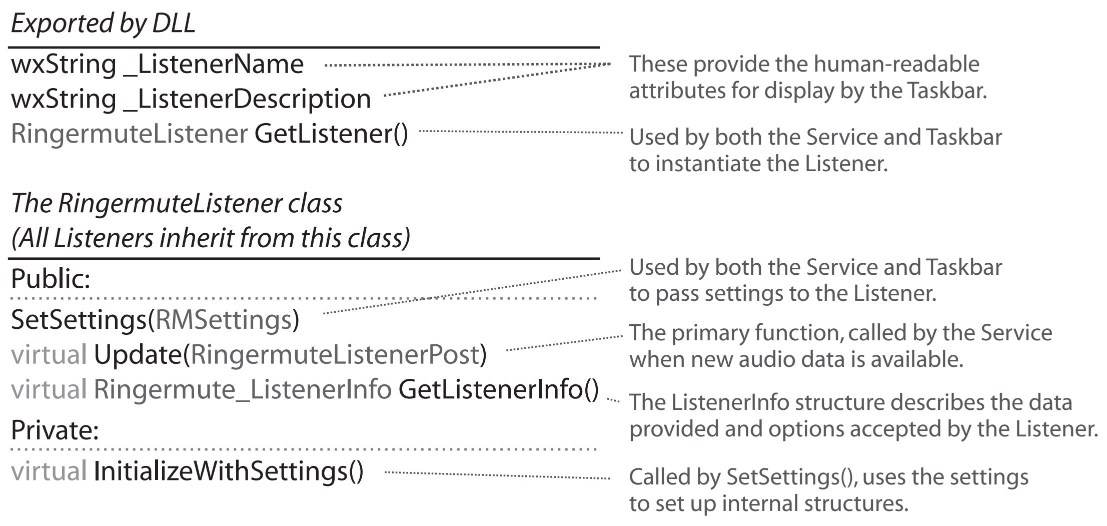

Exported by DLL
wxString _ListenerName
wxString _ListenerDescription
RingermuteListener GetListener()
These provide the human-readable attributes for display by the Taskbar.
Used by both the Service and Taskbar to instantiate the Listener.
The RingermuteListener class (All Listeners inherit from this class)
Public:
SetSettings(RMSettings)
virtual Update(RingermuteListenerPost)
virtual Ringermute_ListenerInfo GetListenerInfo()
Private:
virtual InitializeWithSettings()
Used by both the Service and Taskbar to pass settings to the Listener.
The primary function, called by the Service when new audio data is available.
The ListenerInfo structure describes the data provided and options accepted by the Listener.
Called by SetSettings(), uses the settings to set up internal structures.

Figure 3.2: The Ringermute Listener specification.

Chapter 4

Ringermute Implementation

Ringermute's primary components have been implemented in C/C++, using several open-source, cross-platform libraries. One basic requirement for Ringermute was that it run on the three major desktop applications found in research environments: Microsoft Windows, Apple Mac OS X, and Unix/Linux. Several alternatives were considered, including Java and Python. Mindful of Knuth's caution against premature optimization [39], it was felt that the need for responsive and near-real-time audio analysis, especially when functioning as part of a larger context-aware framework, required better performance from the start. While Java has made great strides over the past decade in speed comparisons with C/C++, it is unfortunately still true that it performs at a disadvantage [86, 87]. Python, although it is a more rapid deployment tool than C/C++ and performs nearly as well as Java [67], was eliminated from consideration by its relative obscurity: more people have had experience with C/C++ or Java than with Python. Another advantage of C/C++ is that byte-code based runtime engines can be embedded within the Ringermute engine. This would allow Listeners to make use of the Java-based WEKA [88] machine learning library, for example.

4.1 Audio Input

Given that each major platform implements its own audio API (and in the case of Linux, several different APIs over the years), it was essential to find robust cross-platform libraries in order to read audio data from both a live source and recorded files. As it happens, only two open-source libraries are robust enough, under active development and available on the required platforms: Portaudio [66] and libsndfile [22].

4.1.1 Portaudio

Portaudio [66] is an open-source library designed to provide access to the audio hardware on a wide range of platforms, including Microsoft Windows, Apple Mac OS X, several Unix variants and BeOS. It is used by a number of applications, most notably the Audacity sound editor. Portaudio is usually run under a multi-threaded callback scheme, but can be run as single-threaded process with blocking I/O. This is how it has been implemented in the Ringermute service, since it only needs to worry about audio input during its execution.

4.1.2 Libsndfile

Since Portaudio does not provide for audio data input from stored files, libsndfile has been used for sound file input and output. Like Portaudio, it is a free, open-source library that runs on a wide range of hardware and operating system combinations. It is capable of reading and writing standard audio formats such as WAVEform audio format (WAV), Audio Interchange File Format (AIFF) and the Sun Unix Audio (AU) file format. Additionally, it can read and write non-audio formats such as the MAT file format used with the open-source MATLAB-compatible numerical application Octave [56], and the file format used by the Hidden Markov Model toolkit HTK [36].

4.2 GUI Elements

Although the primary application is designed to run as as a background service, a user interface has been developed to allow the user to start and stop the service, control which modules are run, and access settings for each module. The Ringermute status monitor runs as a “system tray” application, and displays an icon in the Microsoft Windows Taskbar, the Macintosh OS X Dock, or in the area specified by freedesktop.org's System Tray protocol [61], which is supported by both GNOME and KDE. This design allows the status monitor to indicate whether the Ringermute service is active, access the settings menu, and also display notifications in a relatively unobtrusive manner. Examples of similar applications include Google Desktop Search and Microsoft AntiSpyware. Figure 4.1 shows the service monitor in the Microsoft Windows XP Taskbar.

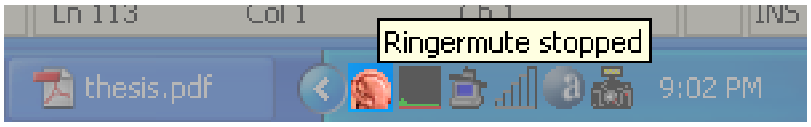
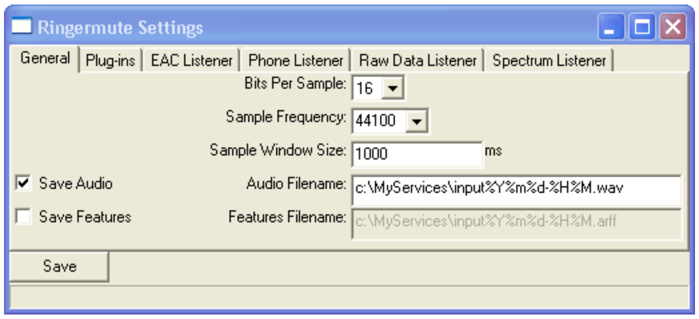


Figure 4.1: The Ringermute service monitor in the Microsoft Windows XP Taskbar.

Both the primary Ringermute settings (Figure 4.2) and the settings for each individual plug-in module (Figure 4.3) are accessed by using "tabbed" windows, which use the metaphor of physical folder or workbook tabs to separate the settings values. The individual plug-in modules do not access the GUI interface directly, but instead indicate to Ringermute what properties are user-controlled. This allows plug-in authors to contribute features without having to learn GUI toolkit routines.

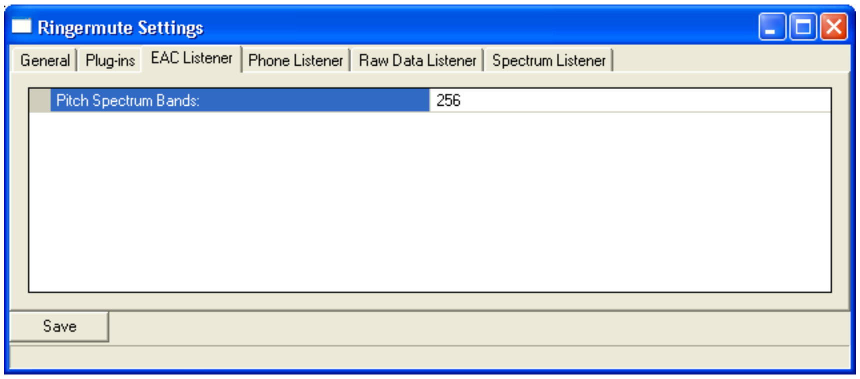


Bits Per Sample: 16
Sample Frequency: 44100
Sample Window Size: 1000 ms
[✓] Save Audio   Audio Filename: c:\MyServices\input%Y%m%d-%H%M.wav
[ ] Save Features   Features Filename: c:\MyServices\input%Y%m%d-%H%M.arff

Figure 4.2: The Ringermute interface.

4.2.1 wxWidgets

Much of Ringermute's functionality has been implemented using the wxWidgets library, a free, open-source, cross-platform toolkit in use since 1992 [77]. Aside from



Pitch Spectrum Bands: 256

Figure 4.3: The Ringermute plug-in property interface for the EAC pitch-detection plug-in.

providing a rich API for GUI applications, wxWidgets also includes several features that made it useful for the Ringermute project. Both the service and status monitor make use of its cross-platform dynamic library features to load the plug-in modules. Settings information for the Ringermute application and the plug-in modules are handled by a cross-platform configuration framework: Under Microsoft Windows, the information is saved to the Registry, and within *.ini files under Mac OS X and Unix/Linux platforms.

4.3 Command-line and Background Applications

Both `rimuextract` and `rimuservice` are fundamentally similar in that they provide a context for Listeners to operate. Their primary difference is the context in which they in turn operate: `rimuextract` is designed to run from the command-line and takes existing audio files as input, while `rimuservice` is designed to run in the background and take live audio as its input. Both applications derive their primary functionality from the Ringermute Central structure mentioned in Section 3.2.

4.3.1 Ringermute Extractor

At the moment, `rimuextract` is a fairly simple application. It accepts a list of audio files as individual arguments, and transforms these into a single file containing the features extracted from the audio file, where each line contains the features from a single audio window. The width of the audio window is set using the `rimutaskbar` application. `rimuextract` makes use of the libsndfile [22] library to read the existing audio files, and also makes use of several classes from the wxWidgets library. Although `rimuextract` is designed to run as a command-line application, `rimutaskbar` allows the user to invoke it within a GUI shell as a convenience feature.

4.3.2 Ringermute Service

Since `rimuservice` runs as a background process, it runs in a different environment than `rimuextract`. Within Mac OS X and Unix systems `rimuservuice` is to run as a daemonized process. This is usually done by forking the process and killing off the parent, so that the newly "orphaned" process is "adopted" by the `init` process. On the Windows platform, the application runs as a Windows Service instead. At present, only the Windows version has been implemented. A service control API has been written that abstracts the primary interface (start, stop, restart), and concrete subclasses are used to implement platform-specific functionality—this is a fairly canonical example of the Bridge [31] design pattern.

Chapter 5

Ringermute Usage

The marriage of context-awareness with audio scene and object recognition makes for a somewhat confusing combination of problems, many of which are still outstanding. The overriding issue is the lack of context standards on any platform, let alone platform-independent standards for gathering, reporting and acting on context. This, in part, is why Ringermute is designed the way it is. Although it is by no means a "context server," it does expose some basic notification functions to its context and recognition plugins. Similarly, while it does not include the breadth and depth of audio analysis functions that other systems, such as Music AI Research SYstem for Analysis and Synthesis (MARSYAS) [84] do, it compensates by integrating its recognition features in the GUI. The goal was not to create the perfect application for researchers or users, but to fit the general needs of both groups. In this way it allows promising recognizers to be quickly used in a real-world setting, while also automating some of the tedium of preparing audio for data mining and machine learning. Since Ringermute was designed for two main categories of usage, we will explore them both in this chapter.

5.1 Feature Extraction and Training

The first step is to gather the audio. Ringermute accepts audio input from the underlying sound API, and can simultaneously save the audio and extract features to an ARFF file. In most cases, the researcher will have recorded audio separately and perhaps prepared it with a tool such as Audacity [12]. In this case, we start with one or more audio files. These will usually be WAV files, but Ringermute is capable of reading all the formats and encodings supported by the open-source library libsndfile. `rimuextract` is a command-line tool that takes the names of audio files as its arguments. Using the central Ringermute settings file, it extracts audio features using the Listener plug-ins mentioned earlier and saves the features to a combined ARFF file. The ARFF format is shown in Figure 5.1. It consists of a header section that describes the number and data types for each exemplar. This is followed by the data for each exemplar, one per line, in comma-delimited format. The @RELATION line names the dataset, the @ATTRIBUTE lines specify the number and types of the data fields, and the @DATA keyword indicates the end of the header. The exemplars follow, one per line, with comma-separated attributes.

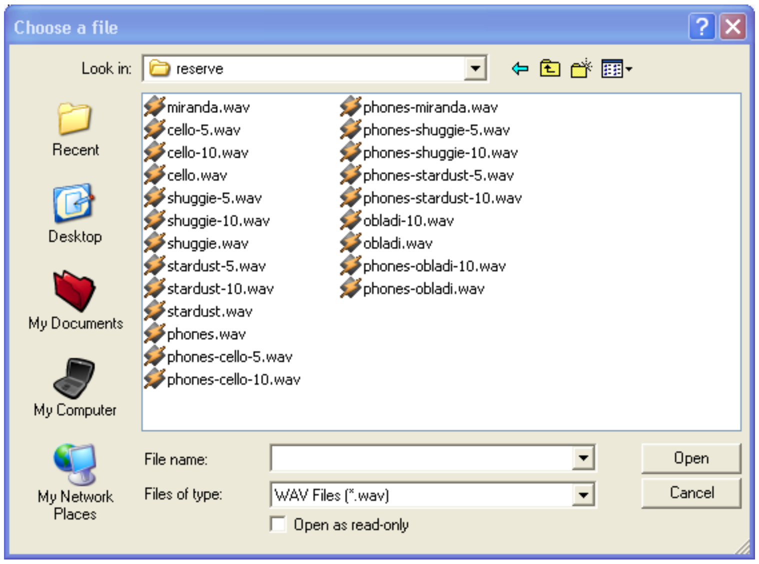
```
@RELATION ringermute
@ATTRIBUTE "Pitch Spectral Bands 0" NUMERIC
@ATTRIBUTE "Pitch Spectral Bands 1" NUMERIC
@ATTRIBUTE "Pitch Spectral Bands 2" NUMERIC
@ATTRIBUTE "Pitch Spectral Bands 3" NUMERIC
@ATTRIBUTE "Spectral Bands 0" NUMERIC
@ATTRIBUTE "Spectral Bands 1" NUMERIC
@ATTRIBUTE "Spectral Bands 2" NUMERIC
@ATTRIBUTE "Spectral Bands 3" NUMERIC
@DATA
0.000000,0.000000,0.000000,0.019188,0.057771,0.004998,0.003088,0.002283
0.030482,0.006997,0.000000,0.024618,0.053476,0.004356,0.003369,0.002325
```

Figure 5.1: An example of an ARFF file.

In order to make this file useful for machine learning applications such as WEKA, we must include not only features, but a class, such as “phone ringing.” By convention, this is the last field or “attribute” in the list of features. For each audio file, `rimuextract` will search for an accompanying text file that contains the start and stop times for the class in question. For example, if a given audio file is named “example.wav”, `rimuextract` will search for a text file named “example.txt” or “example.wav.txt.” This text file contains lines in the following format:

```
floating-point-number␣(start|begin|stop|end)[␣<string>]
```

This is the same format used by Audacity to save its track-labeling feature, so Audacity can be used to visually mark start and stop points for the desired object or scene. The Ringermute Taskbar settings interface (Figure 4.2) can be used to determine which features are extracted from the file, and can also be used to extract the features without using the command line (Figure 5.2).



Look in: reserve

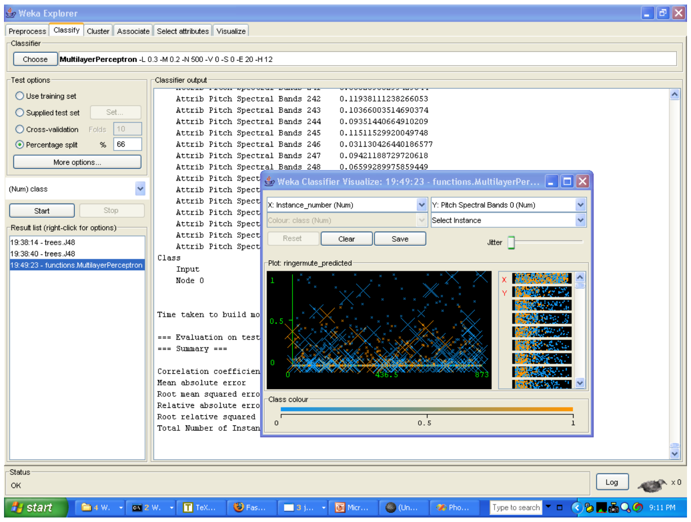
miranda.wav
cello-5.wav
cello-10.wav
cello.wav
shuggie-5.wav
shuggie-10.wav
shuggie.wav
stardust-5.wav
stardust-10.wav
stardust.wav
phones.wav
phones-cello-5.wav
phones-cello-10.wav
phones-miranda.wav
phones-shuggie-5.wav
phones-shuggie-10.wav
phones-stardust-5.wav
phones-stardust-10.wav
obladi-10.wav
obladi.wav
phones-obladi-10.wav
phones-obladi.wav

File name:
Files of type: WAV Files (*.wav)

Figure 5.2: File-chooser window launched from Ringermute taskbar.

Once the ARFF file has been generated, we have a set of exemplars to use for training or evaluation. For example, WEKA provides a comprehensive interface for data-mining, analysis and experimentation. Figure 5.3 demonstrates some of WEKA's visualization capabilites, applied to a Ringermute-generated ARFF file. In the figure, we are seeing a visualization of the resulting error rate after training an artificial neural net on a data set. Given that the Ringermute project grew out of a phone-recognition problem, I have written a small neural-net trainer in C++ that accepts ARFF files with a variable number of numeric input attributes and a single numeric output attribute indicating whether a phone is ringing. The ARFF file was generated using the Ringermute system, and the resulting neural net is used by the

Ringermute PhoneListener plug-in. This illustrates the power of the Ringermute toolset, in that the output from the PhoneListener can then be used as an extracted feature, either with live audio input or pre-recorded audio files. This allows for stepwise refinement as the actual output from the plugin is compared to the expected output, and any improvements to the neural net can be implemented by simply replacing a configuration file.



MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H 12

Cross-validation Folds 10

(✓) Percentage split % 66

(Num) class

Result list (right-click for options)
19:38:14 - trees.J48
19:38:40 - trees.J48
19:49:23 - functions.MultilayerPerceptron

Classifier output

```
    Attrib Pitch Spectral Bands 242    0.11938111238266053
    Attrib Pitch Spectral Bands 243    0.10366003514690374
    Attrib Pitch Spectral Bands 244    0.09351440664910209
    Attrib Pitch Spectral Bands 245    0.11511529920049748
    Attrib Pitch Spectral Bands 246    0.031130426440186577
    Attrib Pitch Spectral Bands 247    0.09421188729720618
    Attrib Pitch Spectral Bands 248    0.06599289975859449
    Attrib Pitch Spect
    Attrib Pitch Spect
    Attrib Pitch Spect
    Attrib Pitch Spect
    Attrib Pitch Spect
    Attrib Pitch Spect
    Attrib Pitch Spect
Class
    Input
    Node 0


Time taken to build mo

=== Evaluation on test
=== Summary ===

Correlation coefficien
Mean absolute error
Root mean squared erro
Relative absolute erro
Root relative squared
Total Number of Instan
```

X: Instance_number (Num)
Y: Pitch Spectral Bands 0 (Num)
Colour: class (Num)
Select Instance
Plot: ringermute_predicted

Status
OK

Figure 5.3: Error visualization in WEKA of results of training of an artificial neural net on ARFF file generated by `rimuextract`.

5.2 Interactive Usage

As previously mentioned, Ringermute is not a complete context server solution, and in interactive mode is intended to be used as a more sophisticated sensor. The Ringermute Listener plug-ins are responsible for taking data from a central repository, processing it in some way (by calculating a spectrum or applying a neural net, for example), and then leaving output for subsequent plug-ins. External actions and notifications are intended to be performed by Listeners themselves, including action-specific Listeners that only perform actions based on the work of previous Listeners. For example, a useful plugin would be one which paused a given media player if a given sound event was detected.

However, some notification routines are built into the system. The main interactive input loop is run as a Windows service or Unix daemon process, and can communicate with the taskbar interface via Dynamic Data Exchange (on Windows) or sockets (on Unix) if a GUI pop-up window is needed. Basic SMTP and HTTP notification is also built into the system as a convenience for plug-in authors.

In any event, the first step for basic interactive usage is to launch `rimutaskbar`, the Ringermute taskbar application, right-click on the icon and and select the "Settings" option from the resulting pop-up window, seen in Figure 5.4.

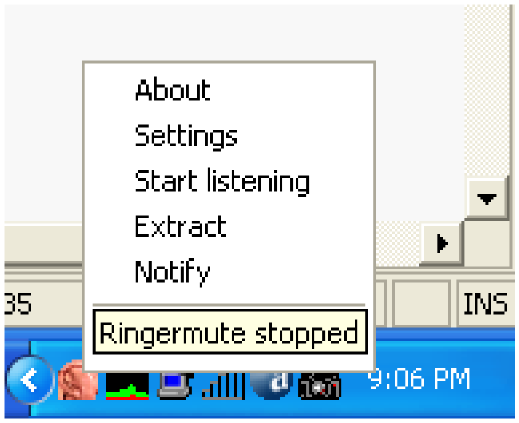
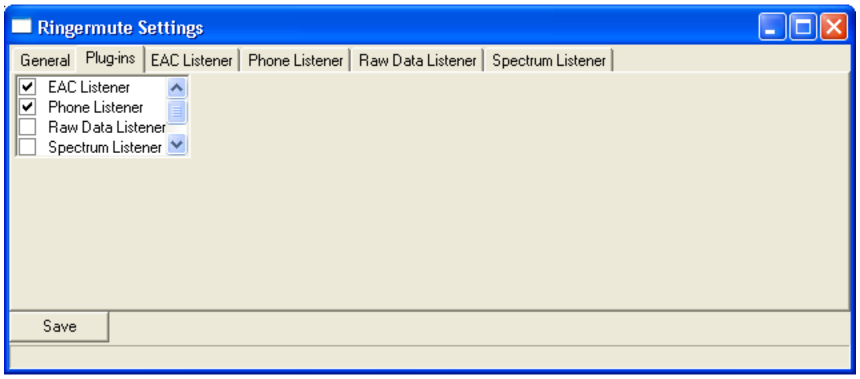


Figure 5.4: Pop-up menu on Ringermute taskbar.

The resulting view can be seen in Figure 4.2. For the sake of demonstration, let's say we are interested in live phone detection. Since the phone detection plug-in depends on the EAC pitch-detection plug-in, we want to make sure it is active as well. Figure 5.5 demonstrates the main plug-in activation list. This shows which plug-ins have been detected, and which ones have been activated.

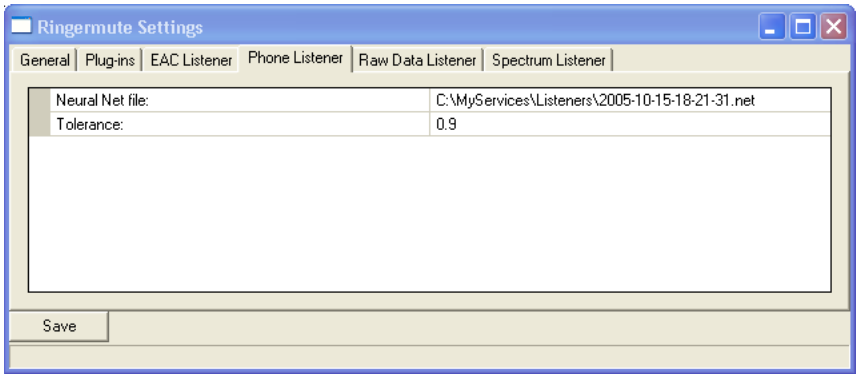
We see that the EAC pitch-detection plug-in is active. The properties window for this plug-in was previously shown in Figure 4.3. We now turn to the properties window for the phone-detection plug-in, shown in Figure 5.6. We can see that it allows the user to determine both which neural net file to use, and the tolerance level to use when detecting phone rings. In this case, the neural net file is in the format



- [✓] EAC Listener
- [✓] Phone Listener
- [ ] Raw Data Listener
- [ ] Spectrum Listener

Figure 5.5: Ringermute plug-in list.

used by the Fast Artificial Neural Network Library (fann). Tolerance refers to the output of the neural net, on a floating-point scale of 0.0 to 1.0, and a tolerance of 0.8 means that any output exceeding 0.8 means a phone ring is present. This allows the user to roughly tune the detection algorithm while the application is running.



Neural Net file:	C:\MyServices\Listeners\2005-10-15-18-21-31.net
Tolerance:	0.9

Figure 5.6: Properties window for the phone-detection plug-in.

Once the settings have been saved, the user starts the Ringermute listening service with the "Start listening" option on the taskbar, and the service begins taking the audio input and calling the plug-ins. If the phone-detection plug-in's output exceeds the tolerance level, it indicates to other plug-ins that a phone was detected. At present it also instructs the taskbar to pop up a notification window—not an ideal action, but suitable for demonstration purposes. This notification can be disabled in the properties window.

Chapter 6

Conclusions and Further Work

The primary niche for Ringermute is sensor design. Before systems can make higher-level decisions about physical or social context, they must first sense more basic phenomena and objects. While calendar data stored inside the computer is trivial to read, the desk calendar next to the computer is not, and yet this may contain useful context information as well.

We have seen where Ringermute fits into the spectrum of context-aware research and applications – it is a toolkit that addresses some of the issues involved in audio object and scene recognition, particularly in concert with machine learning and data mining applications. It has been designed to integrate with users in their natural environment: The graphical user interface. The combination of a plug-in interface with feature extraction allows for the refinement of recognition plug-ins by testing them on live and pre-recorded audio, but also allows their output data to be used as input by subsequent plug-ins, either to perform external actions or additional analysis.

It may seem that a tool such as MATLAB would suffice to perform research on audio context problems. Although such tools are useful and have a place in the research, their primary limitation is that they were not designed to deliver usable applications in an interactive context, particularly for live audio. And while WEKA's tool set is well-adapted to constructing cross-platform applications, the audio framework and feature selection is up to the researcher to provide. Yet the problems of audio input and sound file formats are not so complex that they require re-inventing the wheel every time we want to try a new approach. And the fields of auditory context and scene recognition provide a rich set of features stable enough to be considered standard as well.

6.1 Similar Projects and Existing Tools

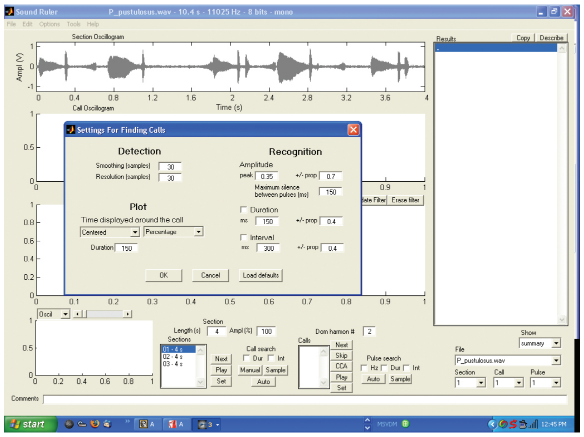
6.1.1 MARSYAS

MARSYAS is an existing framework designed to support audio analysis research [84]. As such, it is not a single application, but includes several command-line applications useful for audio analysis. It shares some goals and features with Ringermute. The predominant metaphor is one of a pipeline, similar to the UNIX concept of pipes, and MARSYAS. Since it is a framework, it is designed to be used by a number of different applications, but no single included application contains all the features in the Ringermute design. Since it has a larger collection of audio feature extraction routines, MARSYAS is an excellent tool to use in the creation of Ringermute plug-ins. The primary difference, then, between Ringermute and MARSYAS is that Ringermute is designed to be a user-friendly, graphical "control panel" for auditory context-awareness research. MARSYAS is a general-purpose library and framework for auditory analysis in general.

6.1.2 Sound Ruler

Sound Ruler is an application that has been developed to meet the needs of bioacoustics researchers [59, 10]. It operates on recorded sound files, and allows the user to recognize and label audio sequences (such as animal calls), either manually or automatically. It offers a large feature set, including the display and graphing of audio data in various forms, including both the time and frequency domains. A screen shot of the interface can be seen in Figure 6.1.

One of SoundRuler's most interesting features is the ability to automatically recognize audio sequences. It does so with a correlation technique: Given an example of the sequence, it seeks to find sequences that match the exemplar's envelope.



P_pustulosus.wav - 10.4 s - 11025 Hz - 8 bits - mono

Smoothing (samples) 30
Resolution (samples) 30
Time displayed around the call: Centered, Percentage
Duration 150
peak 0.35 +/- prop 0.7
Maximum silence between pulses (ms) 150
☐ Duration ms 150 +/- prop 0.4
☐ Interval ms 300 +/- prop 0.4
Oscil
Length (s) 4 Ampl (%) 100
Dom harmon # 2
01 - 4 s
02 - 4 s
03 - 4 s
☐ Dur ☐ Int
☐ Hz ☐ Dur ☐ Int
summary
P_pustulosus.wav
Section 1 Call 1 Pulse 1

Figure 6.1: The Sound Ruler interface.

While this is certainly useful in the context of animal calls, it does not allow for recognition over a general *class* of sequences: the general class of bullfrog calls, for example, versus the specific class of European tree frog *Hyla arborea.* In addition, the recognition algorithm does not appear to be robust enough to recognize all calls individually. Finally, the algorithm itself and the included audio features are not designed to be extensible by others: The application itself is a monolithic one. However, it demonstrates the utility of sound analysis applications in general.

6.1.3 CLAM

On the other end of the spectrum from Sound Ruler, C++ Library for Audio and Music (CLAM) [2, 3] is a framework for audio signal processing. Along with classes and routines for input, processing and analysis, CLAM provides such ancillary functions as data serialization and visualization. Data types range from low-level signal components to higher-level units of analysis, such as phrasing and segmentation [4]. While it provides a wide range of components and features for digital signal applications, CLAM is not an application as such. Neither is it simply a library, in that it provides a conceptual model along with its library functions [2]. A key difference between CLAM and Ringermute, aside from the features CLAM offers, is that CLAM requires more effort to install and deploy in an application. It has been designed to meet the widest range of audio applications, not just context-aware computing. As such it is certainly possible to make use of CLAM's features within the context of a Ringermute Listener.

6.1.4 Audacity

Strictly speaking, Audacity [12] is a sound editing application, and so on first glance may not compare very well to Ringermute at all. However, aside from its obvious utility in preparing sound files for analysis, it includes several analysis visualization features in the program itself, and supports extension through a plug-in interface. Aside from waveform visualization, Audacity includes a spectrum view and a pitch-detection visualization based on work by Tolonen and Karjalainen [83]. Given its feature set, Audacity is a natural candidate to provide supporting features to the Ringermute project—one problem that Ringermute does not solve directly is the issue of labeling sound segments for training purposes. Audacity's interface allows manual labeling, and Aubio, a separate audio labeling project [11, 57, 58], can create label tracks automatically based on audio signal events, such as the beginning of musical notes. Figure 6.2. shows three of Audacity's built-in visualizations: the raw audio waveform, the spectrum, and the EAC pitch-detection algorithm. The label track can be seen below the visualizations.

6.2 Future Improvements

Even though Ringermute is a working and usable tool, there are many opportunities for future refinement. For example, although Ringermute has been built with cross-platform libraries, it has only been developed and tested on the Windows platform. The immediate goal is to produce working executables for the Linux and Mac OS X platforms as well. Documentation, particular API documentation for those seeking to build Ringermute Listeners, is lacking as well.

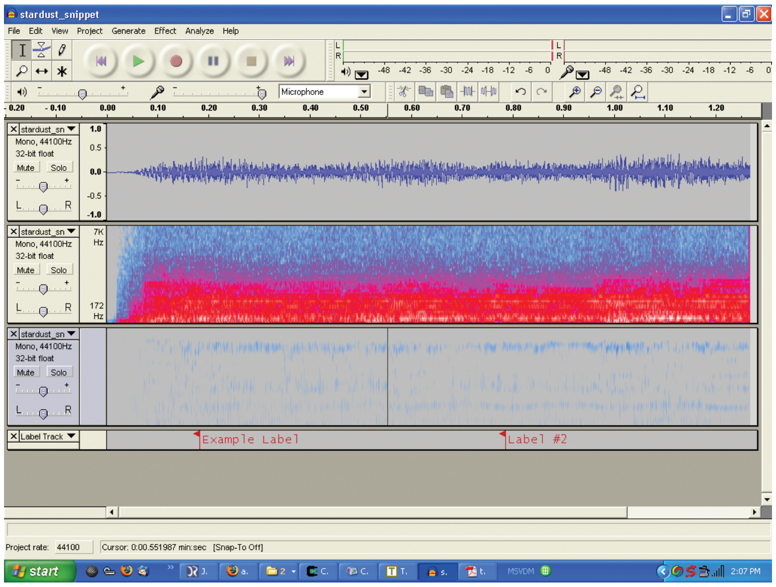

stardust_snippet
File Edit View Project Generate Effect Analyze Help
Microphone
Example Label
Label #2
Project rate: 44100
Cursor: 0:00.551987 min:sec [Snap-To Off]
start
2:07 PM

Figure 6.2: The Audacity interface, displaying the sound file used in Figure 2.1 and Figure 2.2.

6.2.1 Plug-in Dependencies

More sophisticated use of plug-ins would require the system to be aware of plug-in dependencies – at the moment plug-ins are loaded and activated in the order they are discovered by the operating system (alphabetical, in the case of Windows). Plug-ins already provide "provision" information to the system by enumerating the configuration. Adding dependency information and handling would be a fairly simple task.

6.2.2 Multithreading

In its current form, Ringermute performs blocking reads on the audio input, and must wait for all the plug-in modules to execute before reading another input window. A multi-threaded version would simultaneously read audio input to a buffer and execute a plug-in loop. Modules that were not dependent on the execution of earlier modules could run simultaneously. This sub-project would require modification of the data-writing portion as well.

6.2.3 Scripting Interface

Although the plug-in model allows cooperation by researchers without requiring knowledge of GUI or audio libraries, it still requires a separate compilation step for each platform. An ideal situation would be to allow researchers to write plug-ins that can be run on multiple platforms. Audacity, for example, allows the use of plug-ins written in the audio synthesis language Nyquist [21], which contains some features useful in analysis applications. Many languages are designed to be easily incorporated into C/C++ applications: For example, existing gaming engines use languages such as Python and Lua.

6.2.4 Mobile Devices

Aside from the office or workgroup scenarios, the mobile environment offers the largest set of interesting applications for context-aware computing [73, 16]. It also offers a new set of challenges to researchers, although these barriers are rapidly disappearing with the advent of cheaper and more powerful mobile devices. Even so, another project would be to adapt Ringermute to the technical limitations and requirements of mobile devices, such as a the PocketPC or Palm OS.

Bibliography

[1] Piotr D. Adamczyk and Brian P. Bailey. If not now, when?: the effects of interruption at different moments within task execution. In CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems, pages 271–278, New York, NY, USA, 2004. ACM Press.

[2] Xavier Amatriain. An Object-Oriented Metamodel for Digital Signal Processing. PhD thesis, Pompeau Fabra University, 2004.

[3] Xavier Amatriain, Pau Arum, Maarten de Boer, David Garca, Miquel Ramrez, Xavier Rubio, and Enrique Robledo. Clam: C++ library for audio and music. http://www.iua.upf.es/mtg/clam/.

[4] Xavier Amatriain, Pau Arum, and Miguel Ramrez. Clam, yet another library for audio and music processing? In OOPSLA '02: Companion of the 17th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, pages 46–47, New York, NY, USA, 2002. ACM Press.

[5] Stavros Antifakos and Bernt Schiele. Beyond position awareness. Personal and Ubiquitous Computing, 6:313–317, 2002.

[6] Brian P. Bailey, Joseph A. Konstan, and John V. Carlis. Measuring the effects of interruptions on task performance in the user interface. In IEEE Conference on Systems, Man, and Cybernetics 2000 (SMC 2000), pages 757–762, 2000.

[7] Brian P. Bailey, Joseph A. Konstan, and John V. Carlis. The effects of interruptions on task performance, annoyance, and anxiety in the user interface. In M. Hirose, editor, Human-Computer Interaction INTERACT 2001 Conference Proceedings, pages 593–601, Amsterdam, 2001. IOS Press.

[8] L.E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. Annals of Mathematical Statistics, 41(1):164–171, 1970.

[9] Thomas Bayes. An essay towards solving a problem in the doctine of chances. Philosphical Transactions of the Royal Society of London, 53:370–418, 1763.

[10] M. A. Bee. Sound ruler acoustical analysis: a free, open code, multi-platform sound analysis and graphing package. Bioacoustics, 14:171–178, 2004.

[11] Paul Brossier. Aubio: A library for audio labelling. http://aubio.piem.org/.

[12] Matt Brubeck, Joshua Haberman, and Dominici Mazzoni. Audacity: Free audio editor and recorder. http://audacity.sourceforge.net.

[13] Justine Cassell. Embodied conversational interface agents. Communications of the ACM, 43(4):70–78, 2000.

[14] S. Cauvin, M.-O. Cordier, C. Dousson, P. Laborie, F. Lvy, J. Montmain, M. Porcheron, I. Servet, and L. Trav-Massuys. Monitoring and alarm interpretation in industrial environments. AI Communications, 11(3-4):139–173, 1998.

[15] Brian Clarkson, Kenji Mase, and Alex Pentland. Recognizing user context via wearable sensors. In Fourth International Symposium on Wearable Computers (ISWC'00), 2000.

[16] Brian Clarkson, Nitin Sawhney, and Alex Pentland. Auditory context awareness via wearable computing. In Proceedings of the 1998 Workshop on Perceptual User Interfaces (PUI'98), San Francisco, CA, November 1998.

[17] J.W. Cooley and J.W. Tukey. An algorithm for the machine calculation of complex fourier series. Mathematics Computation, 19:297–301, 1965.

[18] Alan Cooper. The Inmates Are Running the Asylum. Sams, 1999.

[19] Jeremy R. Cooperstock, Sidney S. Fels, William Buxton, and Kenneth C. Smith. Reactive environments. Communications of the ACM, 40(9):65–73, 1997.

[20] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, Second Edition. The MIT Press, 2001.

[21] Roger B. Dannenberg. The implementation of nyquist, a sound synthesis language. Computer Music Journal, 21:71–82, 1997.

[22] Erik de Castro Lopo. libsndfile. http://www.mega-nerd.com/libsndfile/.

[23] P. de la Cuadra, A. Master, and C. Sapp. Efficient pitch detection techniques for interactive music. In Proceedings of ICMC 2001, International Computer Music Conference, http://www-ccrma.stanford.edu/ craig/papers/01/icmc01-pitch.pdf, September 2001.

[24] Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Human-Computer Interaction, 16:97–166, 2001.

[25] Nevenka Dimitrova. The myth of semantic video retrieval. ACM Computing Surveys, 27(4):584–586, 1995.

[26] D.P.W. Ellis. Detecting alarm sounds. In The Consistent & Reliable Acoustic Cues for Sound Analysis Workshop (CRAC), pages 59–62, Aalborg, Denmark, September 2001.

[27] Scott Elrod, Gene Hall, Rick Costanza, Michael Dixon, and Jim Des Rivires. Responsive office environments. Communications of the ACM, 36(7):84–85, 1993.

[28] James Fogarty, Scott E. Hudson, Christopher G. Atkeson, Daniel Avrahami, Jodi Forlizzi, Sara Kiesler, Johnny C. Lee, and Jie Yang. Predicting human interruptibility with sensors. ACM Transactions on Computer-Human Interaction, 12(1):119–146, 2005.

[29] Jonathan Foote. An overview of audio information retrieval. Multimedia Syst., 7(1):2–10, 1999.

[30] Eric Freeman, Elisabeth Freeman, Kathy Sierra, and Bert Bates. Head First Design Patterns. O'Reilly Media, Inc., 2004.

[31] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns. Addison-Wesley Professional, 1995.

[32] Hans W. Gellersen, Albrecht Schmidt, and Michael Beigl. Multi-sensor context-awareness in mobile devices and smart artifacts. Mobile Networks and Applications, 7(5):341–351, October 2002.

[33] Ben Gold and Nelson Morgan. Speech and Audio Signal Processing: Processing and Perception of Speech and Music. John Wiley & Sons, 2000.

[34] Eric Horvitz and Johnson Apacible. Learning and reasoning about interruption. In ICMI '03: Proceedings of the 5th international conference on Multimodal interfaces, pages 20–27, New York, NY, USA, 2003. ACM Press.

[35] Eric Horvitz, Carl Kadie, Tim Paek, and David Hovel. Models of attention in computing and communication: from principles to applications. Communications of the ACM, 46(3):52–59, 2003.

[36] Htk speech recognition toolkit. http://htk.eng.cam.ac.uk/.

[37] Tony Jebara, Yuri Ivanov, Ali Rahimi, and Alex Pentland. Tracking conversational context for machine meditation of human discourse. In AAAI Fall 2000 Symposium–Socially Intelligent Agents–The Human in the Loop, 2000.

[38] M. Kashif Saeed Khan, Wasfi G. Al-Khatib, and Muhammad Moinuddin. Automatic classification of speech and music using neural networks. In MMDB '04: Proceedings of the 2nd ACM international workshop on Multimedia databases, pages 94–99, New York, NY, USA, 2004. ACM Press.

[39] Donald E. Knuth. Computer programming as an art. Communications of the ACM, 17(12):667–673, 1974.

[40] Teuvo Kohonen. Self-Organizing Maps. Springer-Verlag, 2001.

[41] Panu Korpip, Miika Koskinen, Johannes Peltola, Satu-Marja Mkel, and Tapio Seppnen. Bayesian approach to sensor-based context awareness. Personal and Ubiquitous Computing, 7(2):113–124, 2003.

[42] Ajay Kulkarni. A reactive behavioral system for the intelligent room. M. eng. thesis, MIT, Cambridge, MA, 2002.

[43] Kristof Van Laerhoven, Kofi A. Aidoo, and Steven Lowette. Real-time analysis of data from many sensors with neural networks. In Fifth International Symposium on Wearable Computers (ISWC'01), 2001.

[44] Christophe Le Gal, Jrme Martin, Augustin Lux, and James L. Crowley. Smartoffice: Design of an intelligent environment. IEEE Intelligent Systems, 16(4):60–66, July/August 2001.

[45] Marcel Levy. Ringermute: Automated phone detection and response. Unpublished poster produced as part of coursework for CS790q at the University of Nevada, Reno, 2004.

[46] Marcel Levy. Ringermute: Automated phone detection using an artificial neural net. Unpublished paper produced as part of coursework for CS793k at the University of Nevada, Reno, 2004.

[47] Sushil J. Louis and Anil Shankar. Simple context learning can improve user interaction. In IEEE International Conference on Information Reuse and Integration (IEEE IRI -2004), 2004.

[48] Lie Lu, Rui Cai, and Alan Hanjalic. Towards a unified framework for content-based audio analysis. In Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '05), volume 2, pages 1069–1072, March 2005.

[49] Lie Lu, Hao Jiang, and Hong-Jiang Zhang. A robust audio classification and segmentation method. In Proceedings of ACM Multimedia '01, pages 203–211, 2001.

[50] Lie Lu and Hong-Jiang Zhang. Content analysis for audio classification and segmentation. IEEE Transactions on Speech and Audio Processing, 10:504–516, October 2002.

[51] Jos M. Martnez. Standards - mpeg-7 overview of mpeg-7 description tools, part 2. IEEE MultiMedia, 9(3):83–93, July-Sept. 2002.

[52] Matlab. http://www.mathworks.com/.

[53] D. Scott McCrickard and C. M. Chewar. Attuning notification design to user goals and attention costs. Communications of the ACM, 46(3):67–72, 2003.

[54] Tom M. Mitchell. Machine Learning. McGraw-Hill, 1997.

[55] Y. Nakajima, Yang Lu, M. Sugano, A. Yoneyama, H. Yamagihara, and A. Kurematsu. A fast audio classification from mpeg coded data. In Proceedings of the 1999 IEEE International Conference on Acoustics, Speech, and Signal Processing, volume 6, pages 3005–3008, March 1999.

[56] Gnu octave. http://www.octave.org/.

[57] J. P. Bello P. Brossier and M. D. Plumbley. Fast labelling of notes in music signals. In Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR 2004), Barcelona, Spain, October 2004.

[58] J. P. Bello P. Brossier and M. D. Plumbley. Real-time temporal segmentation of note objects in music signals. In Proceedings of the International Computer Music Conference (ICMC 2004), Miami, Florida, USA, November 2004.

[59] Marcos Gridi Papp. Sound ruler acoustic analysis. http://soundruler.sourceforge.net/, August 2004.

[60] Vesa Peltonen. Computational auditory scene recognition. Master's thesis, Tampere University of Technology, 2001.

[61] Havoc Pennington and Mark McLoughlin. System tray protocol specification. http://freedesktop.org/Standards/systemtray-spec, November 2004.

[62] Alex Pentland. Perceptual intelligence. Communications of the ACM, 43(3):35–44, 2000.

[63] Silvia Pfeiffer and Thomas Vincent. Formalisation of mpeg-1 compressed domain audio features. Technical report, CSIRO Mathematical and Information Sciences, 2001.

[64] Rosalind W. Picard. Affective perception. Communications of the ACM, 43(3):50–51, March 2000.

[65] Rosalind W. Picard. Toward computers that recognize and respond to user emotion. IBM Systems Journal, 39:705–719, 2000.

[66] Portaudio - portable cross-platform audio api. http://www.portaudio.com/.

[67] Lutz Prechelt. Technical report 2000-5: An empirical comparison of c, c++, java, perl, python, rexx, and tcl for a search/string-processing program. Technical report, University of Karlsruhe, 2000.

[68] Jennifer Preece, Yvonne Rogers, and Helen Sharp. Interaction Design: Beyond Human-Computer Interaction. John Wiley & Sons, Inc., 2002.

[69] J. R. Quinlan. Improved use of continuous attributes in c4.5. Journal of Artificial Intelligence Research, 4:77–90, 1996.

[70] J.R. Quinlan. Induction of decision trees. Machine Learning, 1:81–106, 1986.

[71] J.R. Quinlan. C4.5: Programs for Machine Learning. Morgan Kaufmann, 1993.

[72] Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A bayesian approach to filtering junk e-mail. In Learning for Text Categorization: Papers from the 1998 Workshop, Madison, WI, 1998. AAAI Technical Report WS-98-05.

[73] Nitin Sawhney. Situational awareness from environmental sounds. Technical report, Speech Interface Group, MIT Media Lab, June 1997.

[74] Bill Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In Proceedings of the Workshop on Mobile Computing Systems and Applications, page 8590, Santa Cruz, CA, December 1994.

[75] B.N. Schilit, D.M. Hilbert, and J. Trevor. Context-aware communication. IEEE Wireless Communications, Volume 9, Issue 5:46–54, October 2002.

[76] Claude E. Shannon. A mathematical theory of communication. The Bell System Technical Journal, 27:379–423, 623–656, July, October 1948.

[77] Julian Smart, Robert Roebling, Vadim Zeitlin, Vaclav Slavik, Stefan Csomor, and Robin Dunn. The wxwidgets library. http://www.wxwidgets.org/.

[78] Steven W. Smith. The Scientist and Engineer's Guide to Digital Signal Processing. California Technical Publishing, 1997.

[79] Steven W. Smith. Digital Signal Processing: A Practical Guide for Engineers and Scientists. Elsevier Science, 2003.

[80] Libo Song, David Kotz, Ravi Jain, and Xiaoning He. Evaluating location predictors with extensive wi-fi mobility data. SIGMOBILE Mob. Computing and Communications Review, 7(4):64–65, 2003.

[81] Savitha Srinivasan, Dragutin Petkovic, and Dulce Ponceleon. Towards robust features for classifying audio in the cuevideo system. In MULTIMEDIA '99: Proceedings of the seventh ACM international conference on Multimedia (Part 1), pages 393–400, New York, NY, USA, 1999. ACM Press.

[82] Sergios Theodoridis and Konstantinos Koutroumbas. Pattern Recognition. Academic Press, 1999.

[83] T. Tolonen and M. Karjalainen. A computationally efficient multi-pitch analysis model. IEEE Transactions on Speech and Audio Processing, Vol. 8(No. 6):708–716, November 2000.

[84] George Tzanetakis and Perry Cook. Marsyas: a framework for audio analysis. Organised Sound, 4(3):169–175, 1999.

[85] Haleh Vafaie and Kenneth De Jong. Feature space transformation using genetic algorithms. IEEE Intelligent Systems, 13(2):57–65, March/April 1998.

[86] Rodrigo Vivanco and Nicolino Pizzi. Computational performance of java and c++ in processing fmri datasets. In OOPSLA '02: Companion of the 17th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, pages 100–101, New York, NY, USA, 2002. ACM Press.

[87] Rodrigo A. Vivanco and Nicolino J. Pizzi. Scientific computing with java and c++: a case study using functional magnetic resonance neuroimages. Software: Practice and Experience, 35(3):237–254, 2004.

[88] Weka machine learning project. http://www.cs.waikato.ac.nz/ ml/index.html.

[89] Brian Westphal and Jim King. A genetic algorithms based automatic phone-ring detection system. Unpublished paper produced as part of coursework for CS790k at the University of Nevada, Reno, 2003.

[90] Patrick Henry Winston. Artificial Intelligence. Addison-Wesley, 1993.

[91] Ian H. Witten and Eibe Frank. Data Mining. Morgan Kaufmann, 2000.

[92] Erling Wold, Thom Blum, Douglas Keislar, and James Wheaton. Content-based classification, search, and retrieval of audio. IEEE MultiMedia, 3(3):27–36, 1996.

[93] Tong Zhang and C.-C. Jay Kuo. Heuristic approach for generic audio data segmentation and annotation. In MULTIMEDIA '99: Proceedings of the seventh ACM international conference on Multimedia (Part 1), pages 67–76, New York, NY, USA, 1999. ACM Press.

[94] Tong Zhang and C.-C. Jay Kuo. Hierarchical classification of audio data for archiving and retrieving. In IEEE International Conference on Acoustics, Speech, and Signal Processing, volume 6, pages 3001–3004, Phoenix, AZ, USA, March 1999.