Rhodri Karim

Timbral classification

Computer Science Tripos, Part II
Churchill College, University of Cambridge

May 10, 2011

Proforma

Name:	**Rhodri Karim**
College:	**Churchill College**
Project Title:	**Timbral classification**
Examination:	**Computer Science Tripos, Part II (2011)**
Word Count:	**11777**[1]
Project Originator:	Rhodri Karim
Supervisor:	Dr Alan Blackwell

Original Aims of the Project

To develop a classifier for sounds that discriminates on the elusive notion of *timbre*, described as the texture or colour of a sound. Taking an uncompressed recording as input, a feature vector of sound qualities would be computed and fed to a classifier sensitive enough to distinguish between five instrument classes.

Work Completed

A system for timbral classification of single-instrument sounds was designed, implemented, and evaluated against both ground truth and human judgement. This was enabled by research into auditory perception, statistical correlates of timbre and machine learning. A novel approach was formulated that exploited the time-varying nature of sound produced by musical instruments. Experimental software was also developed for running the human portion of the evaluation. The resulting classifier approaches human accuracy in sound labelling.

Special Difficulties

None.

[1] Ran `delatex <chapters> | tr -cd '0-9A-Za-z \n' | wc -w` over all chapter files.

Declaration

I, Rhodri Karim of Churchill College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed:

Date: *May 10, 2011*

Contents

Acknowledgements

I would like to thank my supervisor, Dr Alan Blackwell, for his invaluable help over the past year, and also for taming my original project idea from a 5-year PhD to one more suitable for an undergraduate dissertation.

I'd also like to thank Christian Steinruecken for pointing me in a more productive (and rigorous) direction, and everyone who agreed to participate unrewarded in my human evaluation.

Chapter 1

Introduction

Timbre is an elusive phenomenon. Most humans can tell the difference between a piano and an acoustic guitar, but when asked to describe a sound we refer to the mechanics of its source, or more often we resort to synæsthetic metaphors. We hear of *bright* trumpet sections, a *sharp* synthesizer patch, or a *thick* guitar tone. To describe sound we can only reappropriate adjectives from the sensory realms of vision, taste and touch.

Timbre is a perceptually complex and thus poorly defined phenomenon. Indeed, ANSI officially defines timbre as that aspect of a sound which is neither pitch, duration nor loudness[1]. Timbre encompasses every other aspect of the perception of sound. This is not a very helpful definition.

1.1 The Work in Context

In this light, recognising an instrument given only the sound it produces begins to resemble similarly ill-posed (and AI-complete) problems in Computer Vision. Work has been done in the fields of psychoacoustics and music theory to develop an understanding of how timbre varies. It can be regarded as a multi-dimensional space, with studies attempting to discover its primary perceptual dimensions [2, p. 282] arriving at psychedelic axes like *colourful–colourless* and *full–empty* [4].

Even if we cannot agree on the topography of timbre, we can be confident in manipulating it. Timbre is affected by a sound's spectral and temporal envelopes, and variations of the former over time. Indeed, sound data is presented to the brain by the inner ear as a low-resolution power spectrum [1, p. 65]. Thus we

[1] "Timbre is that attribute of auditory sensation in terms of which a listener can judge that two sounds similarly presented and having the same loudness and pitch are dissimilar."

can begin to explore timbre space in terms of compact statistical quantities (or *features*) extracted from raw spectral and temporal data.

We must then identify how these features correspond to aspects of timbre perception. For example, ideal white noise has a flat power spectrum, so the flatness of a sound's spectrum should indicate how "noisy" it sounds. EchoNest[2] (an American company) have performed a proprietary principal component analysis on a large sound library to discover relevant timbral features.

Once viable statistical correlates have been discovered, we can begin to catalogue sounds according to them. *Machine listening* is gaining applications, from recognising animal calls to detecting abnormalities in vital organs. The MPEG-7[3] standard provides features for summarisation of timbre, and Audio Analytic[4] (a Cambridge company) is attempting to apply its patented technology widely, to detect aggression on city streets and monitor machinery for failure modes.

Much more detail on individual studies and applications can be found in [5].

1.2 Scope of the Work

With this project, my aim was to construct a classifier that would take a sound recording as input and return a label corresponding to the instrument used to create the sound, with accuracy approaching that of a human. To this end, my efforts have resulted in success, although not quite to the level I expected.

As input to the classifier, a feature vector is computed describing the prevalence in a sound of certain timbral qualities, such as *noisiness* and *brightness*. I did not aim to construct a general model of timbre perception, but instead to investigate and apply mathematical correlates of specific aspects of timbre.

That being said, my original success criterion (found in the Project Proposal) required correlation with human judgement, rather than ground truth. To this end, I developed software to perform an evaluation of human classification skill on sounds taken both from the labelling domain (trumpets, violins, etc.) and from other sources (such as exotic string instruments and farmyard animals).

I also developed a novel approach to capturing the variation of timbre with time that is characteristic of instrumental sounds. The onset and decay of a sound are identified, separately analysed and then integrated with the overall impression of a sound at the classifier.

[2] `http://developer.echonest.com/`

[3] `http://mpeg.chiariglione.org/standards/mpeg-7/mpeg-7.htm`

[4] `http://www.audioanalytic.com/`

Chapter 2

Preparation

In this section, I describe the theory underlying my approach to sound classification. Extensive research was necessary into auditory perception, machine learning and existing work, the results of which I summarise here. I also expand upon the aims set out in the project proposal.

2.1 Acoustics and Psychoacoustics

The perception of a sound can be characterised by four attributes – pitch, loudness, duration and timbre. The first three of these have familiar and tangible meanings, and we intuitively expect each to be related (perhaps in a non-linear fashion) to a single physical characteristic of a sound wave: the fundamental frequency, amplitude and temporal extent respectively.

The truth of the matter is rather more complex, and these distinctions begin to break down upon closer inspection. For instance, the perceived loudness of a tone of constant amplitude is dependent on its fundamental frequency, with our highest sensitivity being to frequencies in the range of the human voice (between 2 and 5.5 kHz), as shown in Figure 2.1.

In this case, the resonant properties of the ear canal cause this filtering effect. There is, however, a more fundamental reason for the confusion of perceptual attributes. The *cochlea* is the transducer of the inner ear, converting mechanical sound energy into an electrical signal. The *basilar membrane* that forms one wall of the cochlea is covered in piezoelectric hair cells, but the variation in thickness of the membrane along its length means that different sections resonate at different frequencies, as detailed in Figure 2.2. The membrane in effect encodes a Fourier transform of the received sound wave directly onto the auditory nerve.

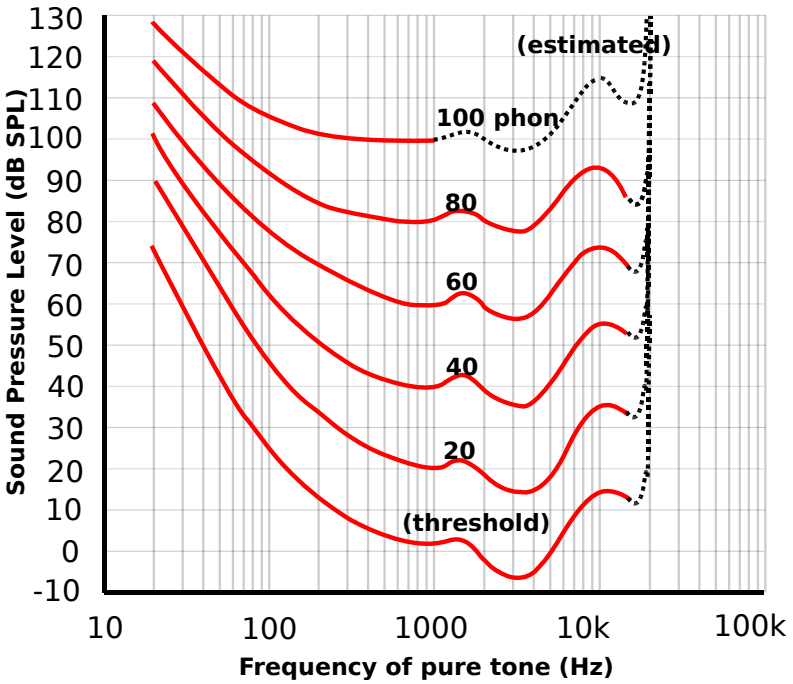

Figure 2.1: A set of empirically-determined ISO 226:2003 equal-loudness curves.

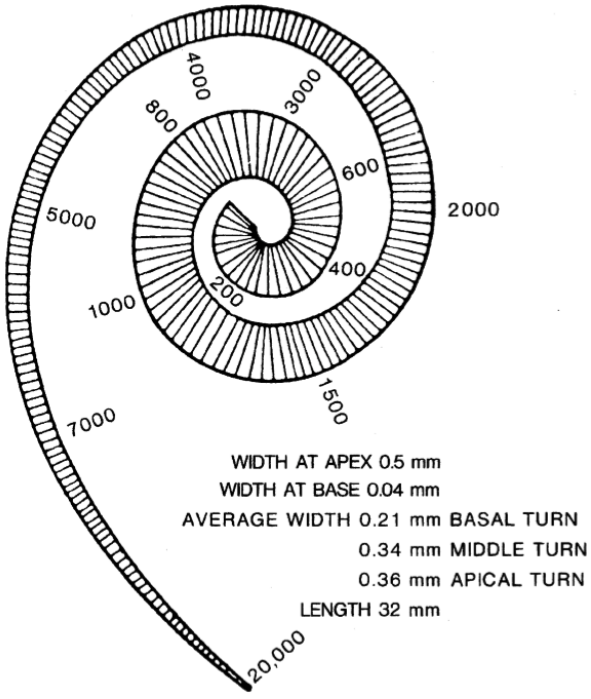

Figure 2.2: The basiliar membrane, with local resonant frequencies indicated [1].

This suggests that pitch, loudness, and a good proportion of timbre are derived by our auditory system from the power spectrum of the incoming sound. On a perceptual level, duration can significantly alter the perception of pitch, with studies reporting that shorter instances of the same tone seem higher in pitch, and atonal at the extreme. As such the temporal envelope of a sound must also play a part in its perception.

We have not considered location as an attribute of perception, nor have we discussed the experience of rhythm or the emotional connotations a sound may carry. These factors (and more) influence a sound's perception, and to confound the process further, the *McGurk effect* [6] shows that hearing and vision interact strongly in the perception of speech. This complexity means that constructing a machine that listens in a human way is an AI-complete problem.

2.2 Timbre and its Correlates

So, instead of attempting to build a model of timbre, it makes sense to seek out statistical features that correlate with it. This would give a sufficient basis on which to differentiate between distinct instrumental timbres. To begin with, I explored the power spectra and temporal envelopes of a number of instrument classes to gain an intuition for their within- and between-class variation.

2.2.1 The Power Spectrum

Tonal instruments gain some of their character from the distribution of energy across their *harmonics*, frequency components at integer multiples of the fundamental frequency, f_0. An abundance of high-frequency harmonics is what gives the trumpet its sharp, bright sound, and these show up as clear peaks in Figure 2.3. A pure sine wave has all of its energy in the first harmonic, giving it a smooth, simple tone.

The rest of the power spectrum is occupied with noise and *partials*, other frequency components produced by the instrument that are not harmonically related. The deviation of higher harmonics from true integer multiples of f_0 contributes to a piano's warm and rounded tone [7], and the spectra of struck objects such as bells and drums are often dominated by these partials.

Thus, we should expect the spectral power distribution to be dominated by harmonics, but in order to include the influence of partials we should more generally summarise how power is distributed with frequency.

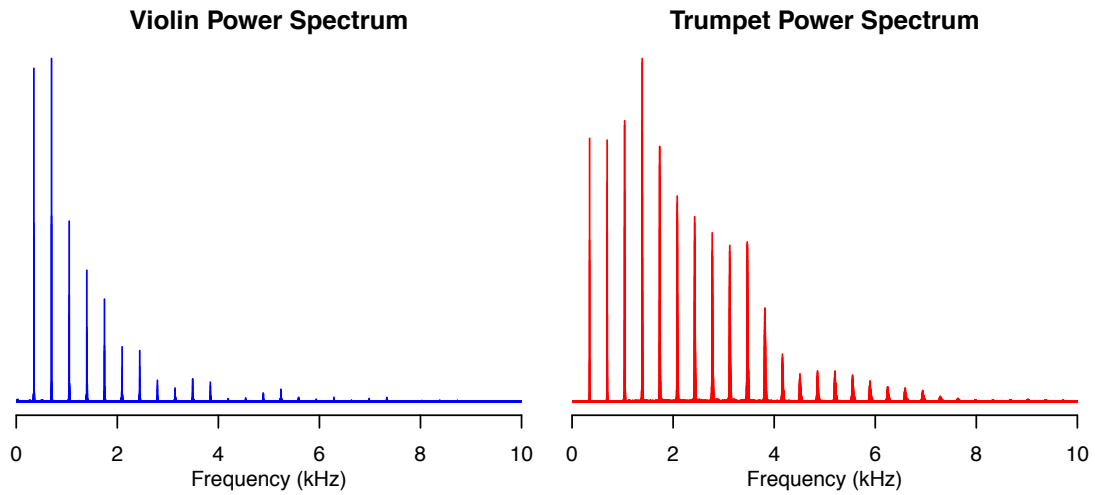

Figure 2.3: Typical power spectra for instruments playing the same note, from two examples in my training set.

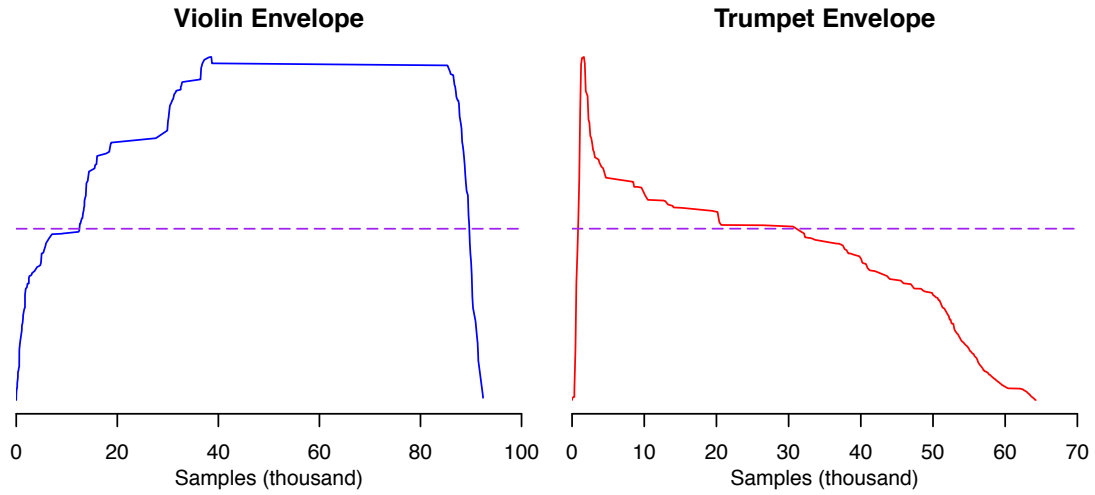

Figure 2.4: Typical temporal envelopes with a line at half maximum intensity, from two examples in my training set.

2.2.2 The Temporal Envelope

The temporal envelope describes a sound's variation in amplitude over time. Thus, that of a plucked violin is characteristically different to that of a bowed violin – not only does a plucked instrument have a sharper onset, but a bowed instrument can be sustained at maximum intensity for as long as the player's stamina allows. It is thus helpful to consider sounds as having attack, sustain and decay portions.

Figure 2.4 shows the typical temporal envelopes of a (bowed) violin and a trumpet. A line at half the maximum intensity of each sound has been marked, and we see that the trumpet reaches this halfway point much sooner than the violin. The trumpet's attack is short-lived, and indeed the main sustained portion of the sound is at around half-power. For a plucked string, we also have that the onset has a markedly different (and more noisy) spectral makeup to the decay portion, as utilised in sound synthesis by the Karplus-Strong method [8].

Thus, there is important harmonic information to be gathered from the sound at the half-power rise and fall points of the envelope, as well as from the position of the points themselves.

2.2.3 High-level Features

Although the raw spectrum and envelope provide plenty of information, it's by inspection and summarisation that we gain a better understanding of how they relate to timbre. We will also wish to maximise the size of our (discrete) Fourier transform to exploit longer sustain periods, causing different sounds to yield different amounts of data. As such, we require a fixed number of high-level *features*, a uniform set of summary statistics derived from the raw input data.

What we've seen so far suggests a number of features, given here for a discrete power spectrum $X(n)$ with N bins at frequencies f_n. All sums and products are over the entire spectrum (from $n = 0$ to $N - 1$):

- Measuring the *centroid* of the power spectrum gives us an indication of where power is concentrated with respect to frequency:

$$\text{Centroid} = \frac{\overline{X \cdot f}}{\overline{X}} = \frac{\sum(X(n) \cdot f_n)}{\sum X(n)}$$

- We can determine the *slope* of the power spectrum by linear regression, which gives us an idea of the relative contribution of higher harmonics:

$$\text{Slope} = \frac{\overline{X \cdot f} - \overline{X} \cdot \overline{f}}{\overline{f^2} - \overline{f}^2} = \frac{N \cdot \sum(X(n) \cdot f_n) - (\sum X(n)) \cdot (\sum f_n)}{N \cdot (\sum {f_n}^2) - (\sum f_n)^2}$$

- The *flatness* of a power spectrum describes how uniform its distribution is, and thus how close it is to the power spectrum of white noise. This gives us a measure of the contribution of partials, and how breathy a sound is:

$$\text{Flatness} = \frac{\text{geometric mean}}{\text{arithmetic mean}} = \frac{\sqrt[N]{\prod X(n)}}{(\sum X(n))/N}$$

- Finally, *onset time* provides a compact description of the sharpness of a sound's attack.

Other statistics can be derived, but as we can see in Figure 2.5 (and later) these can provide a good separation of timbre space by instrument.

2.2.4 Exploiting Temporal Structure

We have arrived at an initial set of features to compactly describe sounds, but the attack-sustain-decay model of instrumental sounds that we saw in section 2.2.2 suggests an improvement. A more expressive feature set can be obtained by applying our spectral features at key points in the temporal envelope, namely: the half-power rise and fall points, and across the time in between (roughly the sustain period).

I believe that this is a novel approach to describing how the timbre of a sound *changes*. Most previous attempts have ignored time-varying behaviour, drastically summarised it using a feature's variance, or centred on short-time Fourier transforms of a sound [5].

As a counter-argument to those methods, the ability to identify an instrument from a recording depends strongly on timbral variation. The repetition of a few periods of a violin's waveform sounds nothing like a violin [1, pp. 92–93], and as we have seen, a plucked string's noisy attack helps to characterise its timbre.

In addition, the third method is far too computationally expensive for use in any real-time scenario such as a live performance, relying on computing a huge number of (small) FFTs over short intervals. I believe that my method strikes a balance between description of timbral variation and computational practicality.

2.3 Machine Learning

The mapping between our sound's feature vector and its correct label (if such a label exists) is not a simple one. In Figure 2.5 we see that although flatness and centroid provide a good separation for violin and trumpet samples, the edges of these clusters are not as well-defined as we would like.

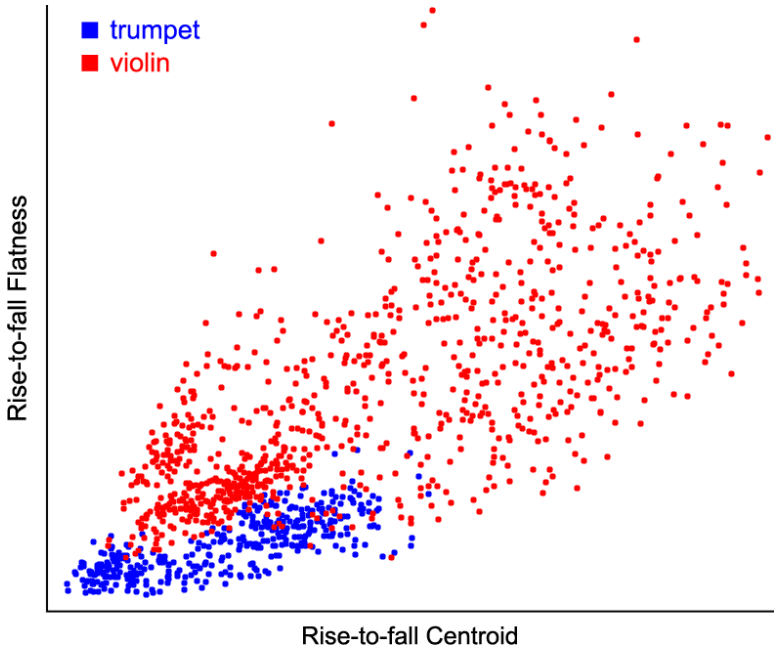

Figure 2.5: Violin and trumpet samples in (a subset of) timbre space.

Having more features should give us better separation, but machine learning techniques will allow us to develop a resilient method of classification. I decided to concentrate on techniques of *supervised* learning, where a data set of fully labelled examples is provided for the purpose of training.

From this training set, our learning algorithm produces a hypothesis which ideally will allow us to label new, unseen examples correctly. This ability is called *generalisation*, and we seek a good fit with the underlying phenomenon, rather than a perfect fit to the labelling of the training set itself (known as *over-fitting*).

Of course, the performance of any classifier is highly dependent on the quality of the training set and the features fed to it. Still, I would be making full use of existing classifier implementations, and an understanding of their attributes would direct my attention towards classifiers suited to the task.

2.3.1 Artificial Neural Networks

The *neural network* can best be imagined as a series of interconnected nodes through which information is transformed as it is propagated. Each node j applies a non-linear activation function σ to some weighted combination of its inputs $a_j = w_0 z_0 + \ldots + w_n z_n$, giving the output $z_j = \sigma(a_j)$. These are usually arranged

in an input layer (where nodes have a single input), a series of hidden layers, and an output layer, although any general feed-forward structure is applicable.

If the activation function is continuous, as the commonly chosen *sigmoid* function is, then the overall output functions (given by the nodes in the output layer) are differentiable and *gradient descent* (through the *backpropagation* algorithm) can be used to learn the optimal weights $\mathbf{w}$ for the entire network by minimising its error function $\mathrm{E}(\mathbf{w})$. This describes the divergence of the classifier's results from the labels of the training set [3, pp. 227–245].

Neural networks cope well with non-linear separations of timbre space, are quick to apply once trained, and have the ability to provide multiple outputs. Each could describe the degree with which the classifier believes the corresponding label to be correct. There are also variations on the basic *multi-layer perceptron* described above, providing tradeoffs of training time against applicability [5].

2.3.2 Bayesian Classification

The Bayesian approach to classification attempts to learn a distribution for labels over examples – the *posterior* – through the application of Bayes' Rule:

$$\mathrm{P}(K|X) = \frac{\mathrm{P}(X|K)\mathrm{P}(K)}{\mathrm{P}(X)} \qquad \text{or} \qquad \text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{evidence}}$$

over classes K and examples X. Our training set contains concrete examples of each label that allow us to estimate the *likelihood*, and we can provide an estimate of the *prior* by how prevalent each label is in the underlying population.

However, the number of labelled examples required to build a general probability model (where features may be dependent) would be prohibitive. Even if we restrict $\mathbf{X}$ to being a boolean vector of n features, for K having k possible labels the number of probabilities $p_{ij} = P(\mathbf{X} = \mathbf{x}_i | K = k_j)$ that we would need to estimate is $\mathrm{O}(k \cdot 2^n)$, $k - 1$ distinct probabilities for each possible feature vector.

This is where the *Naïve Bayes assumption* comes in – assuming that each of the features X_m in our example is conditionally independent of others, then $P(\mathbf{X} = \mathbf{x}_i | K = k_j) = \prod_{m=1}^{n} P(X_m = \texttt{true} | K = k_j)$ and for the above example we need only estimate $\mathrm{O}(k \cdot n)$ probabilities [9]. Even when the independence assumption is an approximation, it brings the learning algorithm into the realm of practicality. However, a Bayesian classifier is fundamentally a linear classifier, which may impede its effectiveness in handling the harder-to-recognise limits of an instrument's range.

2.3.3 Alternative Techniques

Plenty more established machine learning techniques exist, and varied implementations with standardised input, output, and evaluative tools can be found in machine learning toolkits:

- One of the simplest is the *nearest neighbour* classifier, where an example is classified by the majority vote of its neighbours in feature space (calculated by some distance metric, such as the Euclidean distance). This means its main drawbacks are that all computation is left until the classification stage and that irrelevant features can seriously impede its performance.
- A *decision tree* algorithm attempts to partition feature space one feature at a time, learning a series of questions to ask about each example. This allows different subsets of features to be used to decide a sound's class in different parts of timbre space, reducing the influence of confounding features. It also leads to an appealing human-readable description of the classification algorithm. However, finding the optimal decision tree is an NP-complete problem, meaning our choice of decision heuristic is vital.
- A *fuzzy rules* classifier makes use of fuzzy logic, a framework for approximate reasoning. While in Boolean logic set membership is absolute, fuzzy logic assigns to each set a *membership function* from elements to the real interval $[0, 1]$. If an example satisfies a learned rule's conditions, its degree of membership of a label's set is increased by some weighted amount. The classifier can thus emit don't know labels for hard-to-classify sounds [11]. This could be a better outcome than misclassification for a live application of a timbral classifier (such as musical performance).

2.4 Requirements Analysis

For an experimental project of this size, it helped to develop a set of goals to steer the software design process and highlight some appropriate techniques of software engineering.

2.4.1 Principal Goals

From the outset, I was aware that the end result would rely as much on evaluation as implementation. The final feature set and choice of classifier would depend mostly on data collected during evaluation, but the overall structure of the project could be planned in advance:

- Develop a *feature extraction pipeline*, capable of summarising a sound's timbral properties as a feature vector.
- Implement a variety of *statistical features*, and source different types of *classifiers*. Evaluate combinations in terms of how they affect classification accuracy, and suitability for a live application.
- Compile a representative *training set* for a five-instrument domain.
- *Integrate* these into a system able to label novel sounds from the above domain correctly, with at least 75% accordance with human judgement.

2.4.2 Implementation Considerations

A key concern that arose during planning was that of *modularity*, as features and classifiers would be swapped in and out of the system as evaluation dictated. Applying good principles of object-oriented design (detailed in the Implementation section) and employing the use of a machine learning toolkit featuring multiple classifier implementations would ensure that the project could proceed smoothly, and that future refinement would be as straightforward.

This coincided well with the experimental nature of the project, as I could interleave the exploration of feature efficacy with the construction of the feature extraction pipeline itself. Once a basic version of this was complete, I could begin to trial classifiers. All this could be performed in tandem with compilation of a training set, and data collection on human classification judgements. This led to a rather forgiving dependency graph, shown in Figure 2.6.

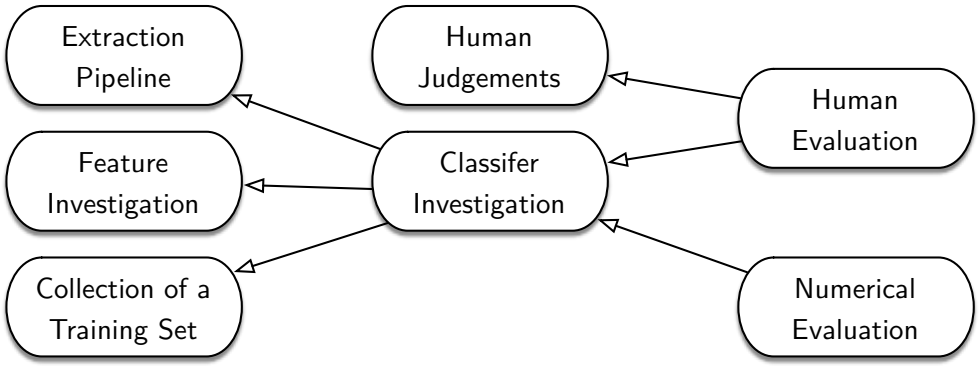

Figure 2.6: A high-level dependency graph for the project, with arrows representing the *depends-on* relation.

2.5 Languages, Libraries and Tools

I chose to code the feature extraction portion of the project in C++ principally due to its reputation for compiled speed and the availability of efficient FFT libraries, while still allowing the use of (some form of) object-oriented design.

I coded the experimental software in Java, primarily due to my familiarity with the audio libraries and Swing as a GUI framework, and also because its strong typing would help to speed up implementation.

The experimental software wrote its results into a local MySQL database, meaning I had to learn the basics of the SQL language to interact with it. For both portions of the project I used the familiar Eclipse development environment, with the appropriate language plug-ins.

To investigate the efficacy of the various classification algorithms, I mainly made use of two machine learning toolkits: Orange and KNIME, described later. Both provided data pre-processors, a range of classification algorithms and automated evaluation tools.

To perform my experimental analysis and produce graphs for this dissertation, I learnt the R language for statistics. I also improved my knowledge of Bash scripting and LaTeX typesetting.

To guard against loss of code and my experimental results, and also to provide version control, I maintained a Git repository on my PWF space, which I would push updates to at regular intervals. To back-up my training set, and provide an additional level of redundancy, I took regular backups of my entire hard-drive using the automated backup facility of Mac OS X.

Chapter 3

Implementation

In this section, I describe in detail the design and construction of the project's core, the feature extraction pipeline. The collection of a training set and concrete experimentation with classifiers is also summarised. Finally, I cover the construction of the stimulus presentation software used for the human evaluation.

3.1 Design Decisions

As described in the last chapter, the investigative and experimental nature of the project meant that little could be known about the actual features and classifier implementations that would end up being used in the finished product.

However, the structure of the project was clear: a feature extraction pipeline would accept uncompressed audio, and output a compact feature vector representation; this would then be passed on to a ready-trained classifier, which would output a label for the given sound. Training of the classifier would take place off-line, with the same feature extraction pipeline being put to use to generate the labelled training set.

Features would be either temporal or spectral, and thus had well-defined inputs (the temporal envelope and Fourier transform of a sound respectively) and outputs (a single numerical value per feature). I would also be the one implementing them, and so could fix the relevant interfaces beforehand.

On the other hand, C and C++ classifier libraries came in all shapes and sizes, with wildly different interfaces and some older examples (such as MLC++[1]) having their own subtly incompatible implementations of parts of the C++ Standard Library! Libraries existed for individual or small collections of algorithms, demanding a different approach be taken for each variety of classifier.

[1] `http://www.sgi.com/tech/mlc/docs.html`

If I was to fully integrate a number of algorithms into the system for testing, I would have had to write wrappers for each of the new libraries introduced. As such, I decided instead to encapsulate the feature extraction pipeline into a command-line tool, and use a stand-alone machine learning toolkit to perform classification, allowing me to swap and evaluate algorithms with (relative) ease.

3.1.1 Structure of the System

The resulting system is structured as in Figure 3.1. `WAV`-format files of a single instrument are input into the feature extraction pipeline (shown on the left), producing a single (unlabelled) feature vector corresponding to a summary of that sound's timbral properties. This is then fed to a trained classifier, which outputs its labelling of the sound. The classifier is trained by supplying the relevant Learner of the machine learning toolkit with labelled examples, generated by running the training set audio through the feature extraction pipeline.

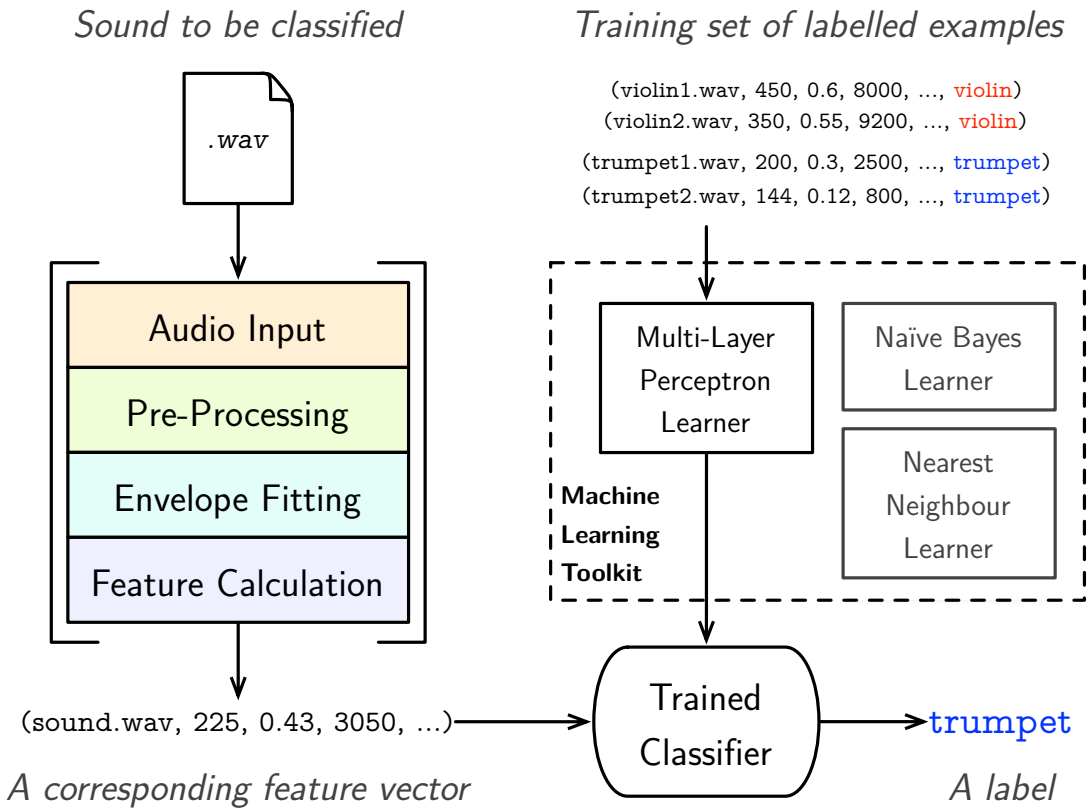

Figure 3.1: A high-level diagram of the system's operation.

3.1.2 Software Engineering

During the design and implementation phases of the project, principles of software engineering proved useful in managing the process' inherent complexity. *Modularity* has already been mentioned as a key concern, the high-level separation of feature extraction and classification allowing me to fully exploit existing classification algorithms.

On a lower level, the problem of feature extraction is *decomposed* into the tractable stages described in detail in what follows. By separating these stages we ensure that each has a well-defined purpose (*tight cohesion*), and should rarely, if ever, need to interact with any other stage to perform it (*loose coupling*). Thus, we promote their *functional independence*, ending up with communication between modules that is simpler (and thus easier to understand).

Separation also facilitates the object-oriented principle of *information hiding*: by hiding the design decisions for a particular component within a standard interface, we limit the impact of any changes in implementation on the behaviour of other modules. For instance, each spectral feature extends the pure virtual `SpectralFeature` class, which implements a `calculate()` method. This method call is all that is needed to produce the result of any feature.

To improve the resilience of the code to boundary and invalid cases, I employed techniques of *defensive programming*, sanitising and checking user input. This was especially relevant in C++, a language without implicit array bounds checking, and helped to eliminate most bugs encountered in development. Good C++ style was also employed: by separating function and class declaration and definition (with header and source files); using an object-oriented style throughout (i.e. no global variables or functions); fastidiously de-allocating objects as soon as possible; and encapsulating all code inside a project-specific namespace.

3.2 Feature Extraction Pipeline

The feature extraction pipeline is the core of the system, and was implemented in C++ due to considerations of compiled speed and the availability of a high-performance Fourier transform library (as described later on). It is conceptually divided into four stages, described in detail in the sections that follow.

3.2.1 Audio Input

The *audio input* module of the pipeline is responsible for reading in uncompressed audio (in the form of monophonic `WAV` files) and coercing them into a format more

amenable to analysis, an array of `doubles`. This format is specified as the `Audio` class, a container for the raw sound data and its associated metadata.

For a C++ program, if an exception is thrown in an object's constructor, that same object's destructor is *not* called[2]. Since the `Audio` object would be carrying around its sonic payload in heap-allocated `double` arrays, this could have led to a memory leak if any error had occurred in between reading the file in and finalising the object.

Instead, `Audio` objects are formed from a `WAV` file using a `static fromWav()` method. This takes a `string` filename, opens the file for reading, verifies the `WAV` header and populates a new `Audio` object with its data. If at any point the operation is forced to fail, a `FileException` or other runtime exception is thrown and the `Audio` object under construction – at that point, still local to the function – is destroyed as it falls out of scope.

New `Audio` objects can also be created as a subrange of an existing instance using the `static windowed()` function, which also applies a windowing function to the resulting audio. This is described later, in section 3.2.4, but simply put: by fading the start and end of a sound sample to silence, it improves the quality of the resulting Fourier transform.

3.2.2 Pre-Processing

The *pre-processing* section of the pipeline is responsible for homogenising the input sounds, so that non-essential differences such as recording volume and silence do not influence the subsequent stages. A pure virtual `Preprocessor` class was specified with a single `process()` method taking a reference to an `Audio` object, so that any pre-processing task could be realised by extending this class and implementing this method.

The `process()` method of implemented pre-processors should alter the supplied `Audio` object in-place, and so the application order of these transformations is relevant. The two implemented pre-processors are `Normalize`, which maximises the peak volume of the sound, and `TrimSilence`, which removes anything below a specified amplitude threshold at the beginning and end of the sample.

3.2.3 Envelope Fitting

The *envelope fitting* module is key to the approach outlined in section 2.2.4, where the most salient intervals of the sound – the attack phase, the sustain phase and the decay phase – are identified and subjected to further analysis. It fits a

[2] `http://www.parashift.com/c++-faq-lite/exceptions.html#faq-17.10`

temporal envelope to the sound, a description of its variation in peak amplitude over time, as shown in Figure 3.2. This provides a compact and easily queried estimation of the energy present in the system – that is to say, the resonating instrument – at any given time.

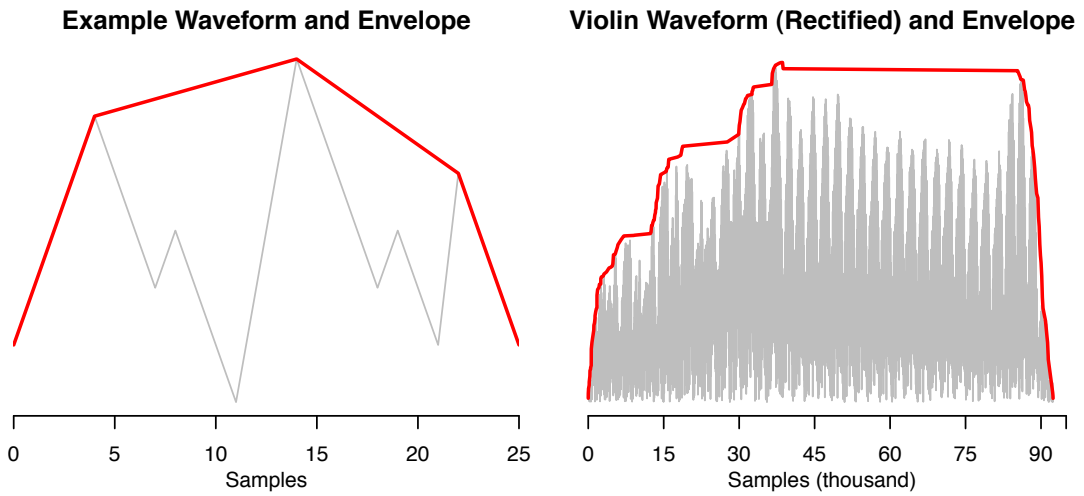

Figure 3.2: Two example waveforms, with their temporal envelopes generated by the `findPeaks` method of my `EnvelopeFitter` class.

The traditional approach to envelope-fitting, and the only one that I could find significant documentation of, is to take a moving average (actually a moving *root-mean-square*) of the signal [2, pp. 9–10]. This presented an awkward tradeoff: too large a window and the important transient behaviour of a short recording was lost, too small a window and the value fluctuated too much to be useful.

Instead, I developed an algorithm that would provide a convex (over-)estimate of a sound's temporal envelope, in the form of a list `peaks` of peak points in the sound that yield the envelope when interpolated. From this the attack, sustain and decay intervals would be apparent, and the half-power rise and fall points – where additional feature calculation could take place – would be easy to extract.

Amplitude perception in humans is in a way similar to the moving average method. Muscles in the middle ear attenuate sudden and loud sounds, much like the iris contracts to save the retina from damaging light intensities [1, pp. 7–8]. However, this adaptation occurs on the scale of tens of milliseconds, and so transient behaviour is still important to the perception of sound.

Regardless, we are interested primarily in identifying interesting points in the sounds to analyse further using spectral methods.

Fitting the Envelope

The algorithm operates on the sound's *full-wave rectified* waveform, obtained by taking the absolute value of each sample. This is because amplitude peaks may occur when the raw signal value is negative. The algorithm computes a finite-difference approximation to the gradient of the rectified waveform, kicking into action when this turns from positive to negative: a possible peak is found.

At these points, we add the current peak to the list of peaks seen so far, then ask: "does this peak have a higher amplitude than the previous peak?". If so, reading backwards through `peaks` we can remove all in between the current one and the last seen peak of a higher amplitude. If there are none higher, we remove all peaks between the current one and the highest seen so far. This ensures that the resulting waveform is convex, while conserving a good amount of detail in the attack and decay segments.

Initialising the algorithm with a single peak at the first sample of the sound, the generated set of peaks can then be queried to build up a set of key points. In our case, we are interested in the half-power rise and fall points: the first and last points where the (normalised) envelope value passes `0.5`. For the first of these, we look for the first peak over `0.5` and interpolate between it and the previous peak to find an estimate of the half-power rise point. The half-power fall point is handled similarly, looking for the last peak over `0.5`.

Initially, I had thought to fit a more abstract *attack-sustain-decay* model to the resulting peaks, resulting in the *proportions* between attack time, sustain time and decay time as temporal features. However, sustained notes brought with them a host of ambiguity problems. It was often hard to tell where sustain ended and decay began (such as for a trumpet like in Figure 2.4, where amplitude tends to fall during the sustain section).

I found that the half-power rise point provided a good indication of the attack sharpness anyway, and that decay was too variable a measure to provide much discriminatory information, and so kept the simpler model.

The `Envelope` and `EnvelopeFitter` Classes

The envelope data itself – in this case, the half-power rise and fall points, and the raw peaks – is encapsulated in a `Envelope` class. This is generated by the pure virtual `fitEnvelope()` method of the `EnvelopeFitter` class, which also has (as a `protected` method) the `findPeaks()` algorithm described above.

Extending this pure virtual base class is the `RiseFallEnvelopeFitter`, which follows the interpolation method described above to identify the half-power rise and fall points after calling `findPeaks()`. A `LinearEnvelopeFitter` was also

developed, attempting to fit the more abstract attack-sustain-decay model described above, recording the transition points and proportions of the three phases in the `Envelope` class.

3.2.4 Feature Calculation

The core of the feature extraction pipeline is the *feature calculation* stage. The choice of features used to describe a sound has the greatest influence over the data passed to the classifier, and thus over the maximum accuracy of classification. Like the other stages a `SpectralFeature` class with a pure virtual `calculate()` method was designed, so that all constructed spectral features could share the same interface. This class also has a `name` parameter, allowing features to be identified in the output (described later).

The `calculate()` method takes a `Spectrum` object, another container class bringing together the complex and power spectra of a given sound. This is in turn constructed using a `static fromAudio()` method, much like an `Audio` object (discussed in section 3.2.1). Thus, the Fourier Transform is computed only once for a given `Audio` object, regardless of the number of features used. Using this spectral data, a `double` value is returned as the feature value for a given sound.

To calculate the Fourier transform of a slice of audio, I made use of a library called FFTW[3] which purported to be the "Fastest Fourier Transform in the West". This C library could perform arbitrary-size FFTs (Fast Fourier Transforms) with optimisations for real-valued input (which, of course, my sounds would be). These include taking advantage of the Nyquist limit and Hermetian symmetry ($X(f) = X(-f)$ for a transform X of a real-valued signal) to compute coefficients only for frequencies between 0 and half the sampling frequency.

Its speed could be improved further by guaranteeing input and output arrays aligned to the `double` boundary in memory, and by making use of *wisdom* – saved data on how best to compute transforms of a certain size, accumulated over a number of runs. To ease operation with C++ I also made use of FFTW++[4], a wrapper which manages these organisational aspects.

Windowing

Each output bin of the (discrete) transform is centred about an integer multiple of f_s/N, where f_s is the sampling frequency and N is the size of the (full, un-optimised) transform. The Discrete Fourier Transform (and thus all FFTs)

[3] `http://www.fftw.org/`

[4] `http://fftwpp.sourceforge.net/`

assume that their input represents a single period of a periodic signal of infinite temporal extent. So, for any of these bin frequencies this periodic extension causes no discontinuities and a strong peak is seen at the respective bin.

However, any components of the sound with a period not equal to one of these bin frequencies would be riddled with discontinuities at the period boundaries if the signal were to be repeated. Thus, naïvely taking the transform of a sound will more often than not lead to *leakage* of energy from the nearest output bin into neighbouring bins, reducing the accuracy of the transform.

We can improve the transform for an aperiodic segment of sound by using a better windowing function – one which smoothly forces the signal to zero at its beginning and end, and thus avoids the introduction of discontinuities at the imposed period boundaries. I chose the *Hann window* shown in Figure 3.3, which is a good general-purpose windowing function which reduces the effect of leakage while introducing little distortion:

$$w(n) = 0.5\left(1 - \cos\left(\frac{2\pi n}{N-1}\right)\right) = \sin^2\left(\frac{\pi n}{N-1}\right) \quad \text{for sample } n, \text{ window size } N.$$

The `Audio` class' `static windowed()` method creates a new instance from a subrange of an existing instance, multiplying the new audio with the right-hand form sample-by-sample. A comparison of an FFT of an unwindowed and windowed input vectors is shown in Figure 3.4, where some reduction of leakage can be seen in the lower frequencies.

A clear explanation of the phenomenon of leakage, taking advantage of the convolution theorem, can be found in [10, pp. 80–87].

Features Implemented

Three principal spectral features were implemented (apart from pitch, described in the next section). These were the `SpectralFlatness`, `SpectralSlope` and `SpectralCentroid`, all of which are mathematically quite simple but correspond well with identifiable changes in perceived timbre.

As described in section 2.2.3: *flatness* is a measure of the similarity of the spectrum to white noise, and thus decreases with the *tonality* (or "spikiness") of the power distribution; *centroid* describes the centre of mass of the spectrum, where most of the signal energy is concentrated; and *slope* indicates the prevalence of higher harmonics, giving an indication of how much of a sound's energy is in the high-frequency components (or perceptually, a sound's *brightness*).

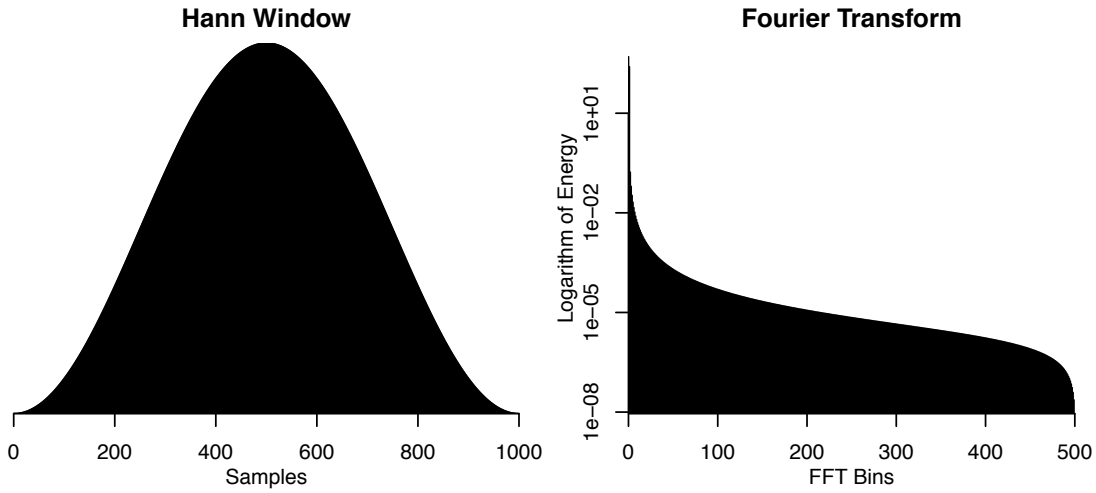

Figure 3.3: The Hann windowing function, and its Fourier transform. Note how quickly the power decays, leading to reduced leakage.

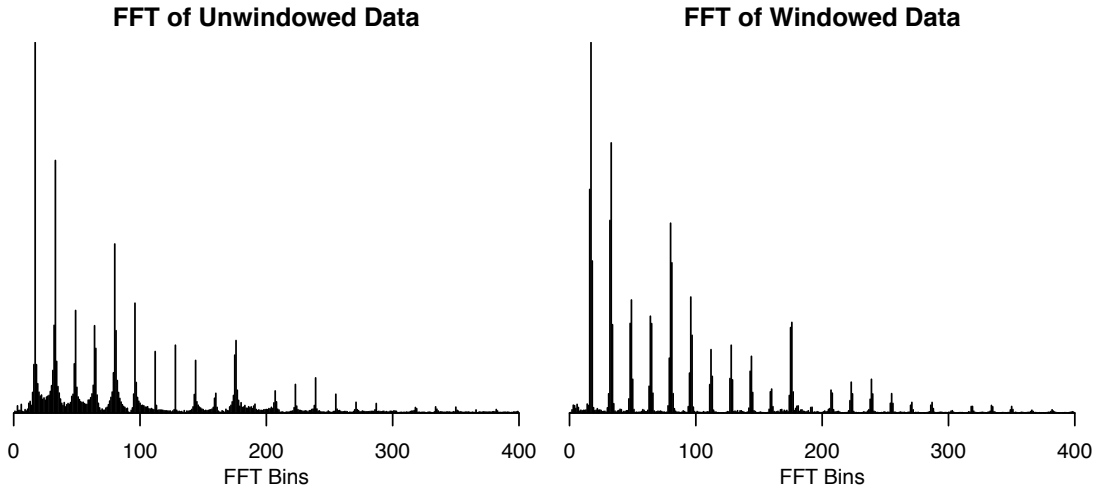

Figure 3.4: A comparison of FFTs of windowed and unwindowed data. Note how less leakage can be seen around the lower frequencies, and how the amplitudes of the peaks are minimally distorted.

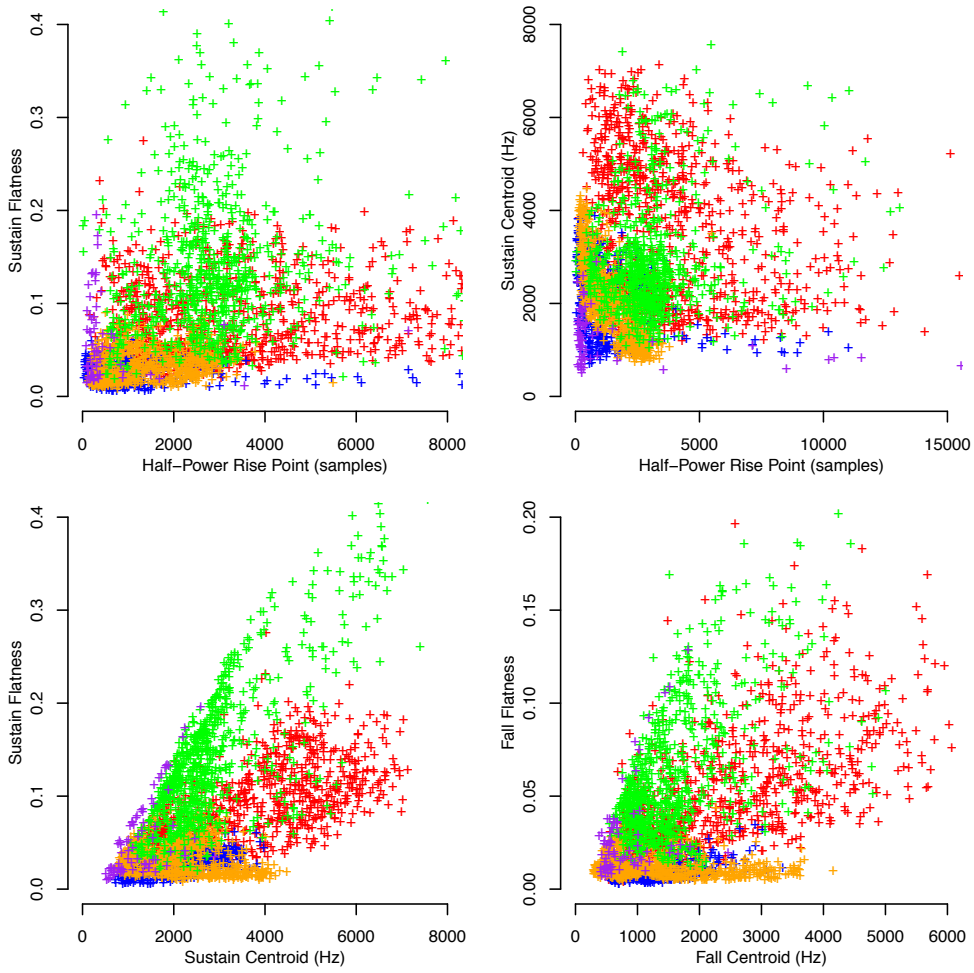

Figure 3.5: Plots showing various 2D projections of timbre space, generated from the data output by my feature extraction pipeline. Point colour corresponds to instrument class.

After inspecting the separation power of each pair of features (with plots such as in Figure 3.5), I assembled an initial feature vector, made up of half-power rise point, fall centroid, fall flatness, sustain centroid, sustain slope, and sustain flatness. Initial trials with a trained multi-layer perceptron subjected to cross-validation gave over 98% classification accuracy for a two-instrument training set – trumpet and violin – and around 70% accuracy for a five-instrument set (five-instrument results are described in much more detail later).

This set of features was efficient to compute – relying on a single pass through the spectral data each – provided good separation between instrument classes, and was perceptually meaningful.

Pitch as a Feature

Pitch is a perceptual rather than a physical phenomenon, and has a complex and decidedly non-linear relationship to the fundamental frequency f_0 and harmonics of a sound. Many sounds will still have a pitch corresponding to f_0, despite this first harmonic having less energy than higher harmonics or being absent altogether (the spectrum of the trumpet in Figure 2.3 shows this phenomenon).

As such, a simple algorithm that looks for the highest peak in a Fourier transform (implemented as `SpectralPeakPitch`) will often not find the pitch. Temporal methods using the sound's *autocorrelation* function can be used, and have a fairly good success rate. However, having the power spectrum already available, I investigated spectral methods of pitch detection, hoping to use pitch as a feature (or at least as a normalising factor for others).

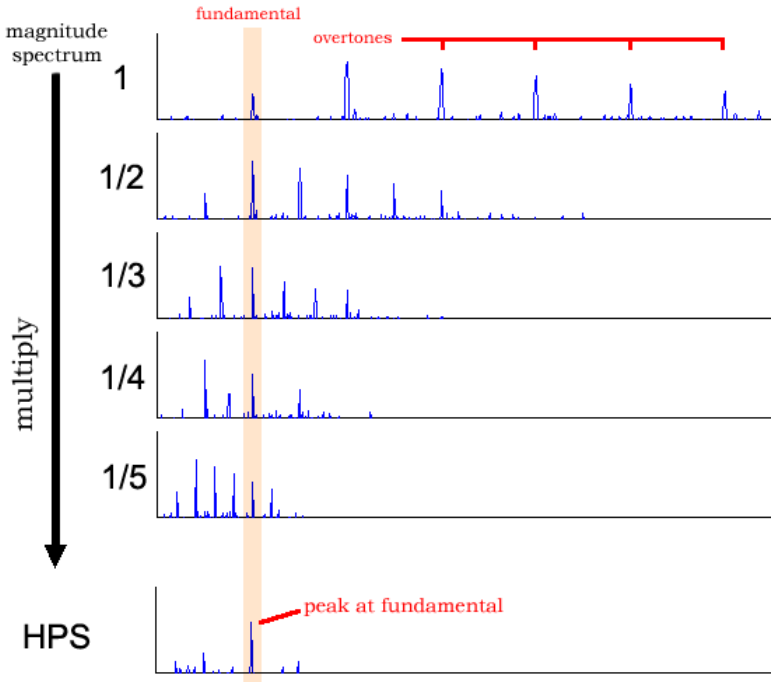

Figure 3.6: A procedure for calculating the Harmonic Product Spectrum. From `<http://sv.mazurka.org.uk/MzHarmonicSpectrum/>`

The *harmonic product spectrum* is built up by downsampling the frequency resolution of a sound's power spectrum by a factor of 2, and multiplying the result by the relevant part of the original power spectrum. This is then repeated for increasing downsampling factors, up to our chosen *harmonic limit*. As such,

the contribution of the higher harmonics will be multiplied into the fundamental, hopefully making it the dominant peak in the HPS.

I coded the HPS algorithm up in R, a statistical language that proved useful in prototyping mathematical and feature code, and producing exploratory plots (about which more will be said in section 3.2.6). It produced good results for a given instrument and harmonic limit, allowing pitch to be picked out easily from the spectrum. However, two major downsides emerged when I implemented the algorithm in C++ as `SpectralHpsPitch`, and applied it to my training set:

- choosing the right value of the harmonic limit was difficult, as it seemed to vary depending on the instrument. Too high a value and higher harmonics would begin to compete with the fundamental, too low and the fundamental would not be boosted enough.
- as the harmonic limit was increased, the maximum frequency included in the resulting harmonic product spectrum was reduced. So abusing the HPS technique would restrict me in the pitch range that I could identify.

I also investigated the *cepstrum* of a sound, essentially a measure of the periodicity of the Fourier transform. This is obtained by taking the transform of the log-power spectrum, and from my investigation was too noisy (and computationally expensive) to be of any real use. Regardless of these difficulties, applying what successful pitch measures I obtained as a feature or a normalising factor did not lead to appreciable increases in classification accuracy (or even separation of investigative plots) so pitch was abandoned as a feature.

3.2.5 Putting the Pipeline to Work

The `Pipeline` class assembles the various stages and modules of the system into a working pipeline. with the `addPreprocessor()`, `setEnvelopeFitter()` and `addSpectralFeature()` methods. Once this is done an `Audio` object can be passed into the `apply()` method, which outputs a `vector<double>` of features. Applying the pipeline to an `Audio` object causes:

- the `Preprocessor`s to be applied,
- an `Envelope` to be fitted,
- and the `SpectralFeature`s applied to `Spectrum`s generated from:
  - intervals around the half-power rise and fall points, extracted with the `Audio::windowed()` function and informed by the `Envelope`,
  - and the sustain portion of the sound.

If the attack segment of the sound is shorter than our preferred window size, then we do not compute the features corresponding to the half-power rise point, as I found that they were too noisy to be of use. Instead, they are output as `NaN` values, to be handled by the machine learning toolkit as appropriate. The same behaviour applies to very brief decay periods.

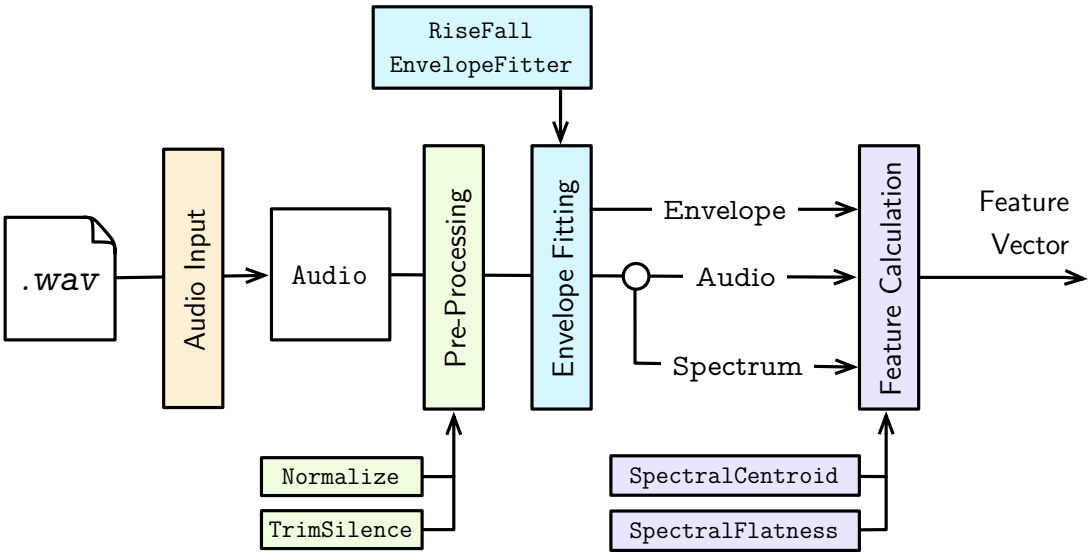

Figure 3.7: An example instantiation of the `Pipeline` class, and its operation.

An overview of the pipeline can be seen in Figure 3.7. The resulting feature vector is then pushed to `stdout` as a tab-separated list of values. Before running the pipeline, the `getFeatureNames()` method is called to generate a header list of tab-separated feature names, to identify the subsequent row values.

The `Pipeline` object is instantiated and populated in a `main` method, which also handles command-line interaction with the tool, and conversion of `WAV` files into `Audio` objects. Thus, the pipeline can be applied to files with the syntax:

```
timbral trumpet/*.wav violin/*.wav ...
```

giving an output (dependent on the choice of features) such as:

```
Filename        Length  RisePoint  RtoFCentroid ...
trumpet/t1.wav  24668   1495       21245
violin/v1.wav   36178   10164      31630
```

3.2.6 Additional Tools

To address bugs, and also to play with the raw audio, envelope and spectrum data produced, I created a number of utility functions that would dump these into tab-separated files on disk, with the syntax: `timbral -dump example.wav`. I could then use the statistical language R[5] to visualise this data, and also to trial features and algorithms without having to debug them in C++. This also allowed me to verify the correctness of their operation (relative to my R implementation).

R is an open-source functional language, with powerful built-in statistics and graphics libraries that allow illuminating plots to be produced either quickly or professionally (but not both). Most of the graphs in this dissertation were produced in R. Like another mathematical environment, MATLAB, the fundamental data structure is that of a *vector*, and most operations can be succinctly performed on a vector of values.

For example, the code for an anonymous function computing an arbitrary-length Hann window is:

```
function(length) {
  sin((pi * 0:(length-1))/(length-1))**2
}
```

Note the `m:n` notation for quickly creating a sequence from `m` to `n`. R also has powerful array-indexing capabilities for selecting and transforming relevant elements. I put R to use in the analysis of results from my human evaluation, detail of which is given in the next chapter.

3.3 Comparing Classifiers

With a feature extraction pipeline encapsulated as a command-line tool, I could freely explore the various classifier toolkits available. My very first experiments were with Weka[6], implemented in Java and with an excellent selection of Bayesian, neural network and more exotic algorithms. Unfortunately, it regularly ran out of memory when training neural networks on large example sets (despite increasing the JVM's maximum heap size into the gigabytes). I quickly switched away, as this seemed to be a recurring problem with Weka[7].

[5] `http://www.r-project.org/`

[6] `http://www.cs.waikato.ac.nz/ml/weka/`

[7] `http://weka.wikispaces.com/OutOfMemoryException`

Next, I moved on to Orange[8], implemented in Python and with a good set of evaluation and data visualisation features. Through the medium of visual programming, a network of data inputs, transformations, learners and evaluation modules could be constructed and rearranged with ease. This allowed for the rapid comparison of the supplied machine learning algorithms, which unfortunately were few in number – and importantly, did not include neural networks.

Seeing as my first experiments with neural networks gave good results, I was keen to explore their performance further. I switched for a final time to KNIME[9], written in Java and the most fully-featured package yet. It offered a similar visual programming interface and plenty of machine learning algorithms, but also allowed for greater control over data handling – including dealing with `NaN` values and normalising training data to improve learning performance.

3.4 Training Set

It would seem that the underlying distribution of instrument labels over any chosen subset of feature space is highly complex. As such, the performance of any classifier applied to the problem of inferring this labelling will be dependent on the size and quality of its training set.

I was fortunate to find a large collection of unrestricted and free single-instrument sounds recorded in an echo-free environment at the website of the Philharmonia Orchestra[10], from which I was able to build a suitably large set of labelled examples. To support my evaluation, I chose a domain of five orchestral instruments: clarinet, flute, trumpet, bowed violin and acoustic guitar.

The training set contains over 2800 samples, covering the full pitch range of each instrument multiple times, and where appropriate a number of different *articulations* (variations in performance technique) and *dynamics* (variations in the sustain length and volume of each note). Included also are extremes, at the very limits of pitch range or volume where they would not usually be played.

However, while performing my evaluation, I realised that drawing the entire training set from a single (comprehensive) source may have led to degraded performance when testing the classifier on an independent dataset. This is covered in Section 4.3.2. Labelled single-instrument sounds without royalty restrictions were extremely hard to find, so I was unable to add other significant sources to my already sizeable training set.

[8] `http://orange.biolab.si/`

[9] `http://www.knime.org/`

[10] `http://www.philharmonia.co.uk/thesoundexchange/make_music/samples/library/`

Once the samples have been organised into a folder per instrument, the feature extraction pipeline can be run on the samples with the command:

```
timbral -train violin trumpet ...
        -f violin/*.wav trumpet/*.wav ...
        > training.csv
```

and the example feature vectors are produced with corresponding labels. Due to the standard tab-separated output format, this file can be directly used by any of the machine learning toolkits mentioned above to build a classifier.

3.5 Human Evaluation Software

Originally, I had hoped to make use of existing software to conduct the human portion of my evaluation. However, my research had turned up few viable packages that were free, supported audio and presented a GUI, and of these none wrote their output into a standard and convenient format.

The experiment would present stimuli (and record responses) in a random order, but with certain sounds repeated to measure the subject's consistency of classification, as described in Section 4.4.1. Subjects would be required to listen to the entire sound at least once before being allowed to choose their labelling.

This made a number of packages unsuitable. For instance, the venerable and comprehensive PsyScope[11] (designed at Carnegie-Mellon University in the early 90s) would have required the use of an arcane scripting language and audio format to support the experiment. PEBL[12] sported a familiar, R-like (but still somewhat awkward) scripting language but came with dubious support for audio stimuli. Both systems recorded each subject's output into a proprietary, text-based format. This would make automation of the analysis with R more difficult.

Rather than spend time working around deficiencies of existing packages, I decided to implement my own stimulus presentation software in Java. This would present instructions and sound samples to the subject, collecting the results in a centralised, easily indexed format amenable to further analysis. Two screenshots of the software are shown in Figure 3.8, one presenting instructions to the subject and the other collecting responses.

[11] `http://psy.cns.sissa.it/`
[12] `http://pebl.sourceforge.net/`

Structure of the Software

The class structure of the resulting system is shown in Figure 3.9. A given `Experiment` is made up of multiple `Parts`: a `TextPart` provides instructions to the subject; a `StimulusPart` presents stimuli one-by-one and collects responses; and a `PracticePart` allows the subject to familiarise themselves with the format.

I adopted a Model-View-Controller structure, separating the user interface from the experimental control logic. The `Experiment` and its `Parts` correspond to the Model, containing the structure and content of the experiment.

For example, a `StimulusPart` is constructed with a `List<File>` of directories which contain the relevant stimulus WAV files, and has a method `selectStimuli()` to provide a subset of those stimuli in a random order with some elements repeated. More detail on the structure of the human experiment, and how the software supports it, can be found in the Evaluation section. Some actual implementation code can be found in Appendix A.

Each `Part` has a corresponding `PartPanel` which displays its contents to the subject (and is thus a View) and dispatches their actions to a `PartPlayer`, which contains the relevant control logic (such as the `StimulusPlayer` being able to play audio, making it the Controller). The experiment as a whole is controlled and sequenced by the `ExperimentPlayer`, which records results using the `DataStore`, an interface onto a backing database.

Database interaction and audio playback capabilities are included in Java's core libraries, which along with familiarity was a key reason for choosing to implement in it. The GUI was implemented in Swing, Java's built-in toolkit, and the database backing the software is a free version of MySQL. The `DataStore` connects to the database using the MySQL JDBC (Java Database Connection) connector. These technologies proved appropriate to the task and were also chosen due to familiarity, enabling rapid development.

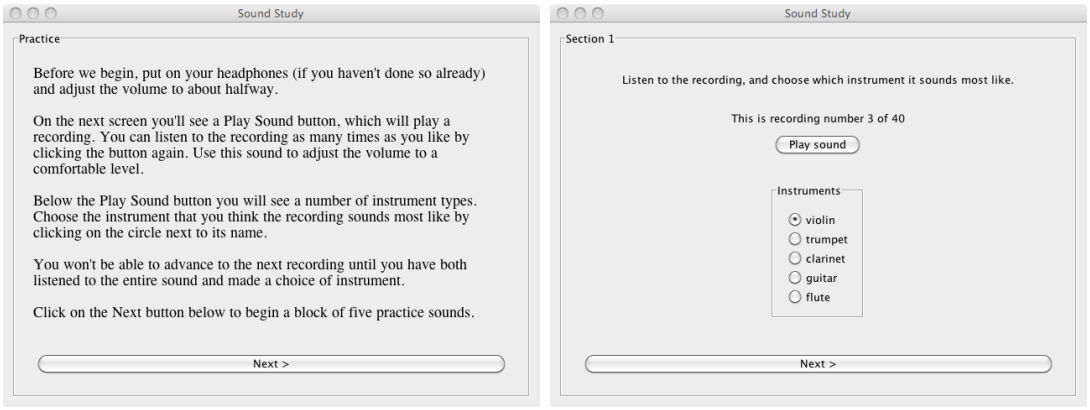

Figure 3.8: Screenshots of the stimulus presentation software.

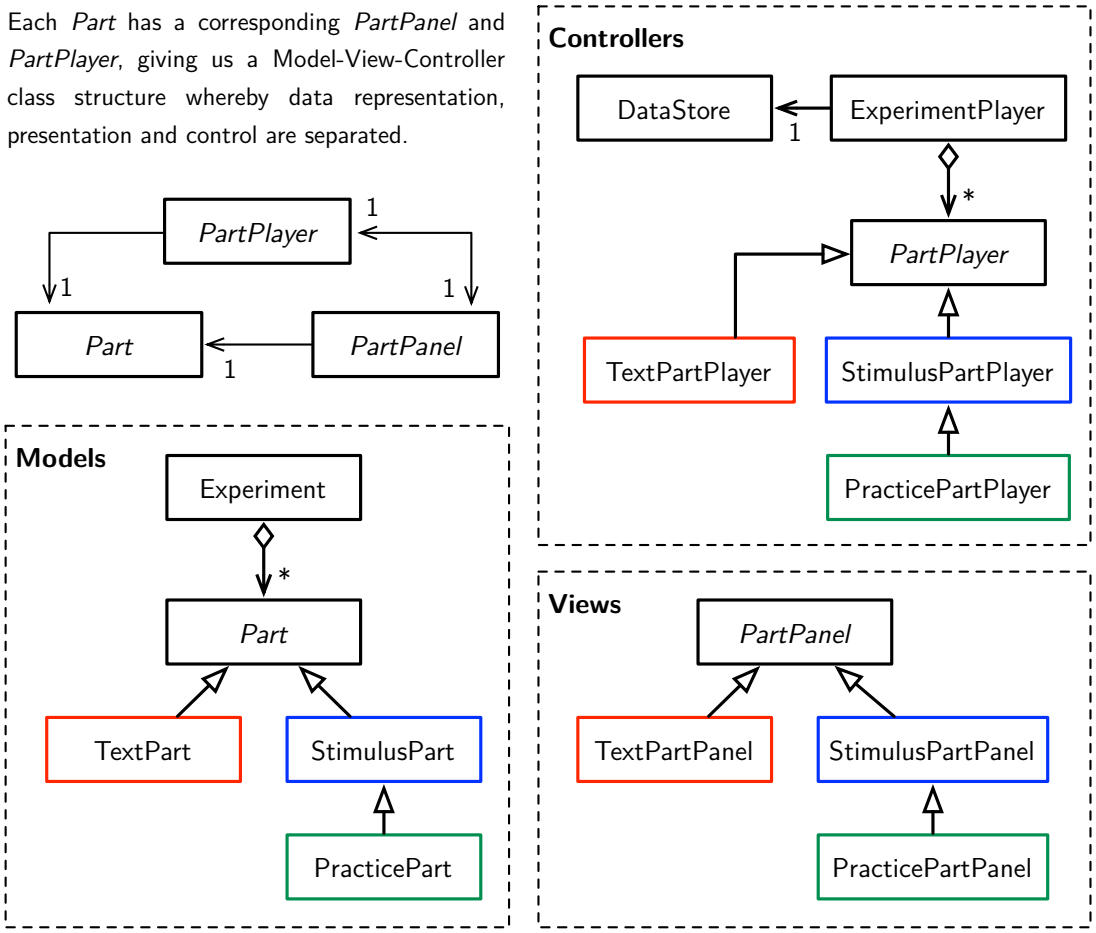

Figure 3.9: A partial UML diagram of the software's class structure.

Chapter 4

Evaluation

In this section, I describe measurements of the classifier's effectiveness, obtained by cross-validation on the training set and a human experiment on an independent test set. I also summarise my experience with the machine learning toolkit.

4.1 Measuring Classifier Performance

Success in classification has many facets, and for a given label we can define measures of success in terms of the four possible outcomes of any classification decision. A *true positive* is the most useful outcome, the correct identification of an example's label. A *true negative* is almost as good, as we have rejected an erroneous label. We then encounter *false positives*, and *false negatives*, spurious identifications and rejections respectively. From these we derive various measures of a classifier's performance:

- *precision* describes how many positives are true positives: $P_{\mathbf{T}}/(P_{\mathbf{T}} + P_{\mathbf{F}})$
- *recall* describes how effectively sounds of a given class were successfully identified, essentially how thorough the classifier was: $P_{\mathbf{T}}/(P_{\mathbf{T}} + N_{\mathbf{F}})$
- *accuracy* describes overall how many decisions were correct: $(P_{\mathbf{T}} + N_{\mathbf{T}})/n$, for total number of decisions n.

Practical classifier design often hinges on the tradeoff between these measures, such as a false negative being far more dangerous than a false positive in disease diagnosis. Of course, these are all measures of binary classification performance, i.e. whether or not an example belongs to a certain class.

Since we are dealing with a multi-class problem, and one where a misclassification has no life-or-death implications (as of yet) it would be most constructive

to use accuracy – defined as the total number of correct classifications, over all classes – as our primary measure. We can extend the notions of true and false positives and negatives to multiple classes with a *confusion matrix*. This is shown in Table 4.1, which describes how a total of 30 examples are classified into three classes. Each column shows how examples with a certain *actual* class are assigned *predicted* labels by the classifier.

Class labels		Actual		
		A	B	C
Predicted	A	9	0	1
	B	2	8	0
	C	0	3	7

Table 4.1: A confusion matrix for a three-class problem. The sum of the diagonal divided by the sum of the entire matrix gives our *accuracy*: 80%.

An accuracy figure for any trained classifier is also dependent on the test data. A classifier is very good at picking up the peculiarities of a particular training set, so using this same set to test the classifier would result in inflated performance figures. We seek a classifier that has successfully *generalised* a model from the training data, and has not succumbed to *over-fitting*. Similarly, if our classifier has user-adjustable parameters, we may want a separate *validation* set on which to play with these.

Since we require labelled data to test our classifier (which is tedious to collect), *cross-validation* is something of a concession. By dividing the full training set into $n + 1$ equally-sized and randomly-populated slices, we can train our classifier on n of these, leaving one on which to test. This procedure can then be repeated for each combination (not permutation) of n training slices, and the mean accuracy calculated. This provides the necessary separation of test and training data, and an approximation of our statistic for a classifier trained on the full training set. This technique was used for the numerical evaluation.

Of course, the best way to test the generality of a classifier is to use a truly independent test set, which my human evaluation described below did.

4.2 Using the Machine Learning Toolkit

As described in the previous section, I made use of the KNIME machine learning toolkit to allow experimentation with a broad variety of classification algorithms.

After the feature extraction pipeline had been completed, I was able to move back and forth between tweaking feature code and measuring its impact on classification accuracy. The filtering features of KNIME then enabled rapid comparison of feature set and classifier combinations.

KNIME also provided methods of treating the training data that were meant to improve the learning process for some classifiers (such as neural networks). These included normalisation, which would remove the pressure on weights to combine values of different magnitude into a meaningful result, and SMOTE: Synthetic Minority Over-sampling TEchnique. This synthesises new example instances for under-represented classes (much like the rotation and skewing of handwriting examples in optical character recognition), but prunes over-represented examples [12]. Normalisation consistently improved classification accuracy, but SMOTE would often reduce it (perhaps due to the procedure removing key examples from over-represented classes) and so was not put to use.

I would consistently obtain better performance from KNIME classifier implementations than from the Orange implementations, with a 5–10% increase in classification accuracy even for a nearest-neighbour classifier. As a point of example the accuracy of the KNIME MLP (multi-layer perceptron, a neural network) was unstable over successive training stages, which may well have been due to a non-deterministic training procedure. KNIME also allowed more customisation of the intricacies of a classifier's behaviour, but even when these parameters were identical it would still perform better than Orange.

4.3 Numerical Evaluation

Numerical evaluation proceeded by ten-fold (i.e. $n + 1 = 10$) cross-validation on my five-instrument training set, consisting of: bowed violin, trumpet, clarinet, flute and plucked guitar. Each feature was normalised using the *z-score* method, where a value is expressed as distance from the mean in standard deviations. From these ten accuracy figures, the mean was computed. Classifier parameters were then tuned to improve performance, resulting in the final accuracies shown.

4.3.1 Results

The accuracies of various feature set and classifier combinations are presented in Table 4.2. This includes a baseline classifier which chooses the most common class (the violin in my training set, making up 32.6% of the examples), and a feature set that does not employ my temporal model (for comparison). For the Fuzzy Rules classifier, *don't know* results were treated as misclassifications.

		Feature sets		
		Sustain flatness, centroid and slope	+ rise time	+ fall flatness and centroid
Classifiers	Baseline	32.6	32.6	32.6
	Naïve Bayes	41.6	48.7	49.0
	Fuzzy Rules	63.3	65.7	67.6
	Nearest Neighbour	61.1	69.3	70.9
	Multi-layer Perceptron	68.1	73.1	75.2
	Decision Tree	69.4	73.6	**76.0**

Table 4.2: Cross-validation results for a variety of classifiers and feature sets. Cell values give the mean *accuracy* of the combination, in percent.

4.3.2 Discussion

Each of the trialled classifiers and feature sets exhibited an accuracy above the baseline, with the best performance coming from the Decision Tree supplied with a full temporal feature set at 76.0% accuracy. This is just above the 75% level set out in my success criterion.

The addition of my temporal model – in the shape of rise time, and spectral features taken at the half-power fall point – made an appreciable difference to the performance of each classifier, with an average 7.0% increase in accuracy over the simple spectral feature set.

Adding the slope feature at the half-power fall point seemed to decrease classification accuracy. This may have been because an instrument's sound tends to lose its high-frequency components in the decay phase, and the calculated slope at this point would be similar for most sounds.

I did not include spectral features taken at the half-power rise point in the supplied feature sets. This was due to a large number of training examples having very brief attack phases, from which little spectral information could be extracted. Since my pipeline emits `NaN` values for the rise features when this occurs, there was little use in including them in the analysis, especially when the test set may have contained such examples. The half-power rise point could still be calculated and supplied as a feature, the *rise time*.

At this stage, I realised that my training set had been recorded in echo-free and near-ideal surroundings. Furthermore, the sound samples for an instrument class would have been taken from a *single instrument* – i.e. all 945 violin samples would have been played on the same violin. This raised the spectre of over-fitting, as my human evaluation would be conducted on a fresh set of samples, taken from a variety of instruments recorded in disparate environments. As such I expected real-world performance to degrade from the cross-validation ideal.

4.4 Human Evaluation

Timbre being a perceptual phenomenon, a true measure of the classifier's effectiveness would come when measured against the prowess of a human listener. First, I would have to investigate human performance at sound labelling, for which I collected an independent set of forty test sounds drawn from the same five classes as my training set. These came from Freesound[1], a repository of user-contributed royalty-free audio.

I also developed stimulus presentation software to conduct the experiment, described in Section 3.5 and below. The human results could then be compared to those of my trained classifier run on the same sounds.

However, I was also interested in whether the classifier would agree with human judgement in situations where a ground truth did not exist. That is to say: faced with a forced choice between my five instrument classes, how would a human classify a sound which belonged to *none* of those classes, and – if a consensus existed – how well would my classifier's judgements accord with theirs?

As an example application, a user could approximate an instrumental sound by non-instrumental means, such as through beat-boxing or a synthesizer. If this sound was convincing or similar enough to cause other humans to apply an instrumental label to it, then an appropriate label should be emitted by the classifier. In this way, the classifier's labels could be put to use in (for example) generating a drum pattern from the user's beat-boxed input sounds.

A positive result here would tell me that my classifier was approaching timbre in a way compatible with human perception. To test this, I collected and produced a further forty sounds, ranging from barnyard animals to synthesized tones. As I had expressed my project's success criterion in terms of accordance with human judgement, this would provide the most abstract test of all.

[1] http://www.freesound.org/

4.4.1 Format of the Experiment

Given my five original sound labels – bowed violin, trumpet, clarinet, flute and plucked guitar – the human experiment would seek to answer these questions:

- For sounds drawn from one of these classes:
  - How accurate is human classification in relation to ground truth?
  - How well does my classifier accord with human judgement?
- For sounds not drawn from any of these classes:
  - Do the humans agree on the classification of any of these sounds?
  - For sounds where humans agree, does my classifier also agree?

The experiment would be conducted by stimulus presentation. Subjects would sit in a controlled environment (in this case, my freshly tidied room) wearing a standard set of acoustically sealed headphones. In this way, the influence of room acoustics and the filtering effect of the head and outer ear would be minimised.

The experiment would take place in two parts of forty sounds each, with a break in between: first, participants would classify sounds drawn from one of the five classes specified; next, they would be forced to label (with the same five labels) sounds not drawn from any of these classes.

The subject would use my stimulus presentation software to assign a label to each of the presented sounds. Thirty-eight sounds would be drawn randomly from my test set of forty for each part. Two randomly chosen sounds in each part would then be repeated later on in the experiment, to test consistency of the judgements of individual humans.

The subject would not be able to advance to the next stimulus until they had listened to the *entire* sound at least once, and chosen a label. The software would also provide an initial practice session, and guidance on the format of the experiment.

Afterwards, they would answer a short questionnaire asking them to describe their musical experience, any hearing problems, and their method for classifying the non-instrumental sounds. A total of fourteen subjects took part in the experiment, males and females of university age (18–21) with varying levels of interest and skill in music.

4.4.2 Part I: Instrumental Sounds

For this section of the experiment, eight sounds from each of the five classes in my training set were selected. These were presented for classification to my subjects, and the results of the experiment compared to machine performance.

Human Performance

My human listeners scored a mean accuracy of 88.6% accordance with ground truth, with half of the subjects scoring 95% and above. Conversely, a sound was correctly labelled 88.9% of the time (on average), with only half of the stimuli having no incorrect labellings.

Performing a chi-squared (χ^2) goodness-of-fit test on the distribution of labels for each stimulus, we can determine which sounds my subjects were able to agree on a consistent classification for. The χ^2 test produces a *p-value* corresponding to the probability that the null hypothesis is true, i.e. that class labels are uniformly distributed. I took my significance level as being a p-value below 5%, which resulted in 35 of 40 stimuli having being labelled consistently by humans.

This is a strong result, but it also demonstrates the fallibility of human perception. Individual humans made mistakes, with the clarinet and flute being most often confused for each other. Indeed, there were three such examples that were correctly classified less than 50% of the time! Three subjects even managed to give two inconsistent classifications for the same sound (as mentioned, my software presented some sounds twice to test this very behaviour), suggesting that context plays an important role in sound perception.

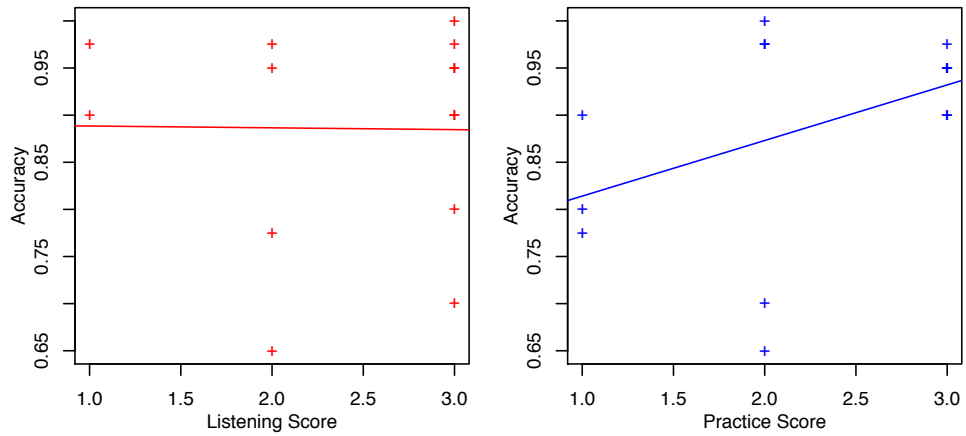

Figure 4.1: Participants' accuracy in Part I shows no correlation ($c = -0.01$) with interest in listening to music (Listening Score), but shows significant positive correlation ($c = 0.43$) with musical skill (Practice Score).

An interesting result that emerged was that classification ability was not correlated with interest in listening to music, as shown in Figure 4.1. Instead, I found that performance increased with musical skill, which demands repetitive

practice and attention to detail. I was able to determine this by scoring each subject's questionnaire answers with a *listening* score and a *practice* score (each from 1–3 with 1 meaning no interest, 2 meaning some and 3 meaning plenty). More detail on this process can be found in Appendix B.

Comparison of Machine Performance

Using the optimal feature set determined by the cross-validation stage (the full temporal model), each of the trialled classifiers was run on both the full 40-sound test set, and the 35-sound set with consistent human classifications (according to the χ^2 test). Again, any *don't know* results emitted by the Fuzzy Rules classifier were regarded as incorrect classifications, more specifically as false negatives. The results are presented in Table 4.3.

Classifier	All sounds	Sound succeeding χ^2
Naïve Bayes	27.5	22.9
Multi-layer Perceptron	55	54.3
Nearest Neighbour	57.5	54.3
Decision Tree	60.0	60.0
Fuzzy Rules	**67.5**	**68.6**

Table 4.3: Classifier accuracy (in %) for the instrumental test set.

Although all better than a baseline classifier choosing the most common set, which would have yielded 20% accuracy in both cases, these results were considerably less pleasing than those for cross-validation. This may well have been due to some over-fitting of the classifier on the training set. The best result was yielded by the Fuzzy Rules classifier, giving me a 68.6% accuracy on the set of sounds for which consistent human labellings existed. This is below, though approaching, the 75% laid out in my success criterion.

Inspecting the Fuzzy Rules classifier's performance further showed that most guitar sounds were causing *don't know* results, perhaps due to their relative under-representation in the training set (making up only 3% of examples).

4.4.3 Part II: Non-Instrumental Sounds

For this section of the experiment, forty sounds were either sourced or produced by myself which were not drawn from any of the five instrument classes present in my training set. These included instruments which produced sound in similar

ways to the included classes (such as the Chinese *erhu*, a bowed string instrument), but also farmyard animals, notes from a synthesizer and the human voice.

The object was to investigate whether the classifier would make similar forced choices to the human subjects, when a consistent human classification existed. In a similar way to the first part of the experiment, these were first presented to humans, and then to my system.

Human Performance

As there was no ground truth (with respect to the available labels) against which to evaluate my subjects' performance, I instead moved straight to the χ^2 test for significance. With the same 5% significance threshold, only 26 out of the 40 sounds were consistently classified by my subjects as being of one of the five available classes. From these I assembled a consensus labelling against which to evaluate machine accuracy.

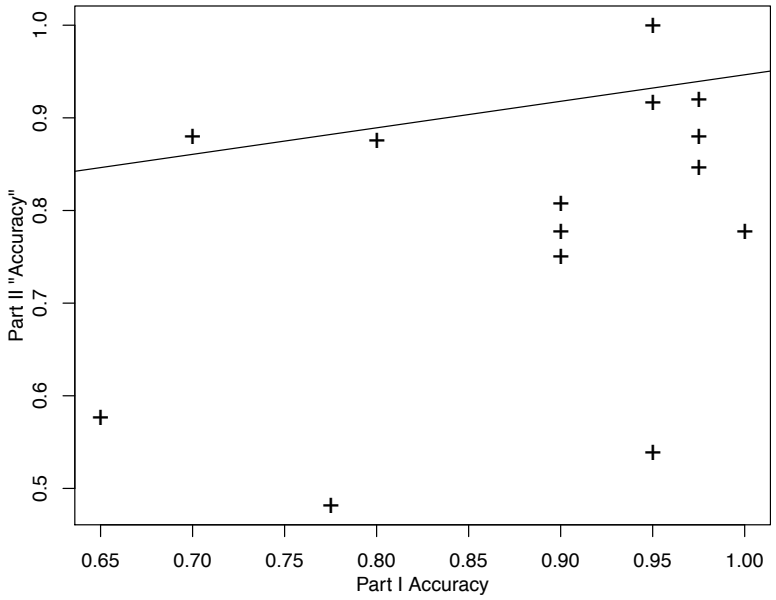

Figure 4.2: Participants' accuracy in Part I shows a very slight positive correlation to their accordance with consensus in Part II.

Human accuracy in this context has a different meaning: since it represents how often my human subjects agreed with the consensus, it essentially represents how sure my subjects were of their collective classification. The mean human agreement on the samples for which a consensus labelling existed was thus 78.8%, an impressive score considering the unusual nature of the task. However, human

"accuracy" on this part was only slightly correlated with accuracy on the first part (with coefficient 0.40), as shown in Figure 4.2.

Much inconsistency was seen for individual subjects: ten out of the fourteen participants gave two different labels to one sound over the course of the experiment. This is not surprising, as from the questionnaire results the participants found it difficult to express the method by which they made their forced choices. Some imagined the mechanism by which the sound could be produced (i.e. a brass, string or wind instrument) while others tried to compare timbre to that of known sounds directly.

Comparison of Machine Performance

In the same format as for Part I, each classifier was run on the 26-sound set for which consistent human classifications existed (according to the χ^2 test). The accuracy result for each classifier here represents its accordance with human consensus. The results are presented in Table 4.4.

Classifier	Accordance (%)
Fuzzy Rules	28.6
Naïve Bayes	38.5
Nearest Neighbour	38.5
Multi-layer Perceptron	42.3
Decision Tree	**50.0**

Table 4.4: Classifier performance for valid non-instrumental examples.

Again, all classifiers performed better than a baseline classifier choosing the most common label (violin), which would have yielded 25% accuracy. However, the best result – a 50% accordance with human consensus yielded by the Decision Tree classifier – was at almost two-thirds of the 78.8% average accordance of a human listener. The Fuzzy Rules classifier here fell back onto its *don't know* result, which led to very poor performance.

4.4.4 Summary

Overall, human classification performance was strong, and my system was stretched to breaking point by the task of anticipating human judgements rather than physical phenomena. The nature of my training set – each class being drawn

from a single instrument – meant that performance on a truly independent set of test sounds would struggle to equal cross-validation accuracy.

However, a classification system of not insignificant accuracy was produced. This was in part due to the adoption of my temporal model, which represented the time-varying behaviour of the sounds in a compact fashion. My system produced some success on the task of mimicking human judgement when interpreting novel sounds, and also in the context of more familiar tasks.

Over the three sections of evaluation, the *Decision Tree* classifier performed well: 76% accordance with ground truth in cross-validation, 60% accordance with human consensus for instrumental sounds, and 50% accordance for non-instrumental sounds. This begins to approach human performance: nominal 100% accuracy on the training set, 88.6% on instrument sounds and 78.8% accordance with consensus on non-instrumental sounds.

As such, although it is clear that the success criterion of 75% accordance with human judgement I originally set out in my project proposal was not satisfied, significant progress has been made towards it.

Chapter 5

Conclusion

This project's aim was to create a classifier capable of correctly labelling instrumental sounds. More specifically, my concern was for a classifier that reflected human judgement, hence my success criterion and the more abstract part of my human evaluation being expressed in terms of it.

Concerning the principal goals set out in Section 2.4.1, the project was successfully designed, implemented and evaluated: a feature extraction pipeline developed, features and classifiers integrated and experimented with, and a training set covering the gamut of pitch and articulation for five instruments produced.

As for the final performance of my classification system significant progress has been made towards a classifier that accords with human judgement in difficult situations. As a first attempt, 60% classification accuracy on an instrumental test set compares well with human accuracy of 88.6%. An accordance with human consensus of 50% on non-instrumental sounds begins to approach the average human agreement of 75%.

However, machine classification accuracy on these independent test sets does not exceed my success criterion of 75% accordance with human judgement. Perhaps this goal was too lofty. For one thing, I had decided not to restrict the pitch or dynamics of the instrumental sounds used. This would have reduced each instrument class' sprawl over feature space and perhaps have made the resulting classifiers more robust.

I also chose five instruments as the domain in which to classify, and from initial experiments fewer classes would have led to greater success. Even my choice of instruments may have led to some difficulty in classification, with the similar-sounding flute and clarinet causing ambiguity in both human and machine classification.

I elected to produce a simple and tangible feature set derived from human perception of timbre, rather than iterating through statistical features in the

quest for optimality. In subsequent reviews of the literature, I found that these two techniques – restriction of feature space and aggressive optimisation – had led to the formidable results being reported for instrumental classification [5].

However, the project did give me valuable insight into the difficulties associated with tackling an ill-posed problem. Separating philosophical concerns (such as the nature of timbre perception) from factors relevant to the project (developing a reliable system, and investigating statistical correlates of timbre) proved a useful skill. I was also able to design, execute and thoroughly analyse a human experiment, drawing relevant conclusions from the data.

I originally envisioned an application for such a classifier in computer music, allowing live performance where a computer could react to sound input in a way sensitive to its timbre – for example, creating a rhythm from a percussive sound or a melody from a tonal sound. Although the resulting classifier doesn't quite have the required accuracy for such applications, it's conceivable that with an improved training set the architecture produced would perform well.

So, I look forward to advances in machine learning and the understanding of perception, and the new applications they will create for machines sensitive to (and in tune with) human judgement.

Bibliography

[1] R. M. Warren. *Auditory Perception: A New Analysis and Synthesis*. Cambridge University Press. 1999.

[2] N. Collins. *Introduction to Computer Music*. Wiley, Chichester. 2010.

[3] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York. 2006.

[4] A. J. M. Houtsma. *Pitch and timbre: Definition, meaning and use*. Journal of New Music Research 26.2, 1997.

[5] P. Herrera-Boyer, G. Peeters and S. Dubnov. *Automatic Classification of Musical Instrument Sounds*. Journal of New Music Research 32.1, 2003.

[6] H. McGurk and J. MacDonald. *Hearing lips and seeing voices*. Nature 264.5588, 1976.

[7] B. L. Giordano and D. Rocchesso. *The effect of inharmonicity on the perceived quality of piano tones*. Proceedings of the 13th Colloquium on Musical Informatics, 2003.

[8] K. Karplus and A. Strong. *Digital Synthesis of Plucked String and Drum Timbres*. Computer Music Journal 7.2, 1983.

[9] T. Mitchell, *Generative and Discriminative Classifiers: Naive Bayes and Logistic Regression*. Pre-press draft, 2005. Downloaded from `<http://www.cs.cmu.edu/~tom/NewChapters.html>`

[10] M. Kuhn, *Digital Signal Processing*. Lecture notes, Part II Computer Science Tripos. University of Cambridge, 2011. Downloaded from `<http://www.cl.cam.ac.uk/teaching/1011/DSP/slides-4up.pdf>`

[11] M. R. Berthold, *Mixed fuzzy rule formation*. International Journal of Approximate Reasoning 32.2, 2003.

[12] N. V. Chawla, K. W. Bowyer, L. O. Hall and W. P. Kegelmeyer, *SMOTE: Synthetic Minority Over-sampling Technique*. Journal of Artificial Intelligence Research 16, 2002.

Appendix A

Code Excerpts

C++ Example: Feature Calculation Code

```
double SpectralCentroid::calculate(Spectrum& spectrum) {

    double numerator = 0.0, denominator = 0.0;
    for (unsigned int i = 0; i < spectrum.spectrumLength; i++) {
        denominator += spectrum.powerSpectrum[i];
        double frequency =
            ((double)i / spectrum.audioLength) * spectrum.sampleRate;
        numerator += spectrum.powerSpectrum[i] * frequency;
    }

    return (numerator / denominator);
}

double SpectralFlatness::calculate(Spectrum& spectrum) {

    double logGeometric = 0.0; double arithmetic = 0.0;
    for (unsigned int i = 0; i < spectrum.spectrumLength; i++) {
        arithmetic += spectrum.powerSpectrum[i];
        logGeometric += log(spectrum.powerSpectrum[i]);
    }
    arithmetic /= (double)spectrum.spectrumLength;
    double geometric = exp(logGeometric/(double)spectrum.spectrumLength);

    return (geometric / arithmetic);
}

double SpectralSlope::calculate(Spectrum& spectrum) {

    double length = (double)spectrum.spectrumLength;
    double f = length * (length + 1) / 2;
    double ff = length * (length + 1) * (2*length + 1) / 6;
    double a = 0.0, fa = 0.0;
    for (unsigned int i = 0; i < spectrum.spectrumLength; i++) {
```

```
        a += spectrum.powerSpectrum[i];
        fa += i * spectrum.powerSpectrum[i];
    }
    double numerator = length * fa - f * a;
    double denominator = length * ff - f * f;

    return numerator / denominator;
}
```

Java Example: Stimulus Selection Code

```
public List<Pair<Integer, File>> selectStimuli() {
    List<Pair<Integer, File>> result = new ArrayList<Pair<Integer, File>>();
    List<File> flatStimuli = getFlattenedStimuli();

    Random random = new Random();
    int uniqueStimuli;
    if (stimuliToRepeat > 0 && repetitions > 0) {
        uniqueStimuli = totalStimuli - stimuliToRepeat*(repetitions-1);
    } else {
        uniqueStimuli = totalStimuli;
    }

    // Check that the request makes sense.
    if (uniqueStimuli < stimuliToRepeat || uniqueStimuli <= 0) {
        throw new IllegalArgumentException("Impossible stimuli selection! " +
            "Need to choose " + stimuliToRepeat + " to repeat from " + uniqueStimuli);
    } else if (uniqueStimuli > flatStimuli.size()) {
        throw new IllegalArgumentException("Not enough unique stimuli provided: " +
            "need to choose " + uniqueStimuli + " from " + flatStimuli.size());
    }

    // Choose unique stimuli
    List<Integer> seen = new ArrayList<Integer>(uniqueStimuli);
    for (int i = 0; i < uniqueStimuli; i++) {
        int chosen = random.nextInt(flatStimuli.size());
        while (seen.contains(chosen)) chosen = random.nextInt(flatStimuli.size());
        seen.add(chosen);
        result.add(new Pair<Integer, File>(chosen, flatStimuli.get(chosen)));
    }

    // Repeat stimuli
    List<Integer> repeatedIndex = new ArrayList<Integer>(stimuliToRepeat);
    List<Pair<Integer, File>> repeated =
        new ArrayList<Pair<Integer, File>>(stimuliToRepeat*(repetitions-1));
    for (int i = 0; i < stimuliToRepeat; i++) {
        int chosen = random.nextInt(uniqueStimuli);
        while (repeatedIndex.contains(chosen)) chosen = random.nextInt(uniqueStimuli);
        repeatedIndex.add(chosen);
        for (int j = 0; j < repetitions - 1; j++) repeated.add(result.get(chosen));
```

```
    }

    // Insert repeated stimuli at random positions
    for (int i = 0; i < repeated.size(); i++) {
        result.add(random.nextInt(result.size() + 1), repeated.get(i));
    }

    return result;
}
```

R Example: Pitch Detection Experiments

```
downsample <- function(v,factor) {
  if (factor < 2) v else {
    d <- v[seq(1, length(v), factor)];
    for (i in 1:(factor-1)) {
      d <- d + v[seq(1, length(v), factor) + i]
    };
    (d / factor)[1:(length(v)/factor)]
  }
}

hps <- function(spectrum, harmonics) {
  hpsLength <- length(spectrum)/harmonics;
  product <- spectrum[1:hpsLength];
  for (i in 2:harmonics) {
    product <- product * downsample(spectrum, i)[1:hpsLength]
  };
  product
}

hann <- function(length) sin((pi * 0:(length-1))/(length-1))**2

cepstrum <- function(spectrum) {
  fullTransform <- c(spectrum, rev(spectrum));
  logTransform <- log(fullTransform);
  cep <- fft(logTransform);
  cep[1:length(spectrum)]
}
```

Appendix B

Detail on the Human Experiment

Here, I have included additional information on the human portion of my evaluation, specifically the coding procedure for questionnaire answers. Included also is the original Human Subjects Declaration, and the consent form that participants had to sign.

The raw results and R scripts used to produce the analysis have been submitted along with the source code.

The Questionnaire

To extract information amenable to analysis from the questionnaire answers, I made use of a *coding frame*. This is a binning procedure, consisting of a set of groups among which the answers could be divided, and allowing the freeform answers given in the questionnaire to be discretised.

I was interested in how enthusiasm for *listening* to music and instrumental *skill* individually affected an individual's classification performance. To this end, I devised a simple scale from 1 to 3 for each attribute.

For skill, a participant would be scored 1 if they had not mentioned ever learning an instrument, or categorically ruled it out. They would score 2 for mentioning some instrumental practice, or having learned when a child. They would score the maximum of 3 if they mentioned ongoing or diverse practice.

For listening, a subject would score 1 if they did not mention listening to music, or mentioned listening to little. They would score 2 if they mentioned listening to some, or a small range of music. They would score the maximum of 3 if they mentioned heavy, diverse musical listening habits.

On subsequent pages, the original questionnaire is reproduced, and the original consent form and Human Subjects Declaration are attached.

Sound Perception Experiment Questionnaire

Please answer the following questions about yourself and the experiment:

What musical experience do you have?
(i.e. do you play an instrument, listen to a lot of music, or even study it?)

Are you aware of any problems with your hearing?

For sounds that were particularly difficult to classify, how did you arrive at your final decision?

Did you have any other difficulties during the experiment?

Please hand this questionnaire back to the researcher.
Thank you again for participating in the investigation.

Sound Perception Experiment Consent Form

Experiment Purpose & Procedure

The purpose of this experiment is to gather data on how sound is perceived and classified.

The experiment is in three sections and it will require about fifteen minutes of your time. In the first section, you will have an opportunity to practice listening to and labelling sounds. In each of the second and third sections, you will be asked to label fifty sound samples with the name of the musical instrument each sounds most like. After the experiment, you will be asked to complete a questionnaire.

Confidentiality

Your labelling of the sounds played to you during the experiment will be recorded. All data will be coded so that your anonymity will be protected in the dissertation that will result from this work, in addition to any further research papers and presentations.

Finding out about result

If interested, you can find out the result of the study by contacting the researcher, Mr Rhodri Karim, after 9th June, 2011. He can be contacted at Churchill College, University of Cambridge, Storey's Way, Cambridge CB3 0DS. His phone number is 07981 986431 and his email address is rk395@cam.ac.uk

---

Consent

Your signature below indicates that you have understood the information about the sound perception experiment and consent to your participation. The participation is voluntary and you may refuse to answer certain questions on the questionnaire and withdraw from the study at any time with no penalty. This does not waive your legal rights. You should have received a copy of the consent form for your own record. If you have further questions related to this research, please contact the researcher.

_____ _____

Participant Date

_____ _____

Researcher Date

Human Subjects Declaration

Since timbre is fundamentally a perceptual phenomenon, measuring the agreement of a timbral classifier with human judgement seems an apt strategy for evaluation. Such an experiment would also offer a ceiling against which to compare the performance of the classifier (i.e. the agreement on classification between humans).

Participants would be played a body of sound samples one-by-one using headphones in a quiet room, and asked to classify each according to some familiar taxonomy (such as the elements of a drum kit). These judgements would then be compared with the classifications arrived at by a timbral classifier trained on the relevant domain. This procedure would be repeated for a number of sound sample domains (such as the sections of an orchestra).

The experiment would be held at the Computer Science department, and would not involve the retention of any personal or uniquely identifiable data. Participants would be asked to provide details of musical experience in case this correlates with the consistency of their judgements. As a reward, a chocolate bar or a small amount of cash may be offered.

Declaration

I will abide by the Computer Laboratory *Human Subjects Declaration* web page, downloaded on the 22nd of October, 2010.

Signed,

Rhodri Karim

Project Proposal

Rhodri Karim
Churchill College
rk395

Computer Science Part II Project Proposal

Real-time timbral classification

22nd October 2010

Project Originator: Rhodri Karim, after discussions with Alan Blackwell

Resources Required: See attached Project Resource Form

Project Supervisor: Alan Blackwell

Signature:

Director of Studies: John Fawcett

Signature:

Overseers: Simone Teufel and Jon Crowcroft

Signatures:

Introduction and Description of the Work

A sound is perceived according to three measures: pitch, loudness and timbre. Pitch is correlated with a sound's fundamental frequency, and provides an ordering from low to high. Loudness is correlated with amplitude, and allows ordering from quiet to loud. This leaves timbre to describe all other perceptual attributes, from the percussive twinkling of a dulcimer to the heady fullness of a bassoon.

Timbre is often described as the *texture* or *colour* of a sound, and descriptions can seem accordingly synaesthetic in quality. This richness of description means there can be no single physical correlate for timbre as it is perceived. Instead, we can examine individual aspects of timbre using an array of mathematical measures, for instance:

- The brightness or richness of a sound varies with the distribution of overtones, components of a higher frequency than the fundamental. The *spectral envelope* of a sound describes how its power is distributed with frequency, and thus allows us to identify and analyse these overtones. The prevalence of overtones is what makes an "ah" vocalisation sound brighter than an "oo".

- The spectral character of a sound as described above is free to *vary with time*. For instance, the sound of a plucked guitar string begins bright and sharp, before tailing off with a more rounded tone. This would correspond to the higher overtones becoming less prevalent over time.

- The *time envelope* describes how the volume of a sound changes over time, often usefully described with phases of attack, decay, sustain and release. A sharp and percussive sound, such as a plucked violin, would be characterised by short attack and decay phases.

Many approaches to pitch detection have been developed, from advanced autocorrelation methods used in the popular Auto-Tune software, to polyphonic pitch detection employing a fast Fourier transform. There has been some success with beat and rhythm detection, and fingerprinting techniques routinely identify copyrighted music in YouTube videos.

However, timbre's ineffability seems to have lead to few applications for its analysis, other than in the resynthesis of instrument sounds and as a component in music information retrieval systems. With a shift towards classifying a sound based on its timbre, I believe new applications in live performance and sample

bank curation could become feasible. It is thus my aim to construct a real-time classifier for sounds, using mathematical correlates of timbre as feature inputs.

Resources Required

In order to train a supervised classifier a labelled corpus of examples is necessary. There are online repositories such as Freesound offering open-source or Creative Commons licensed sound samples. If these prove unsuitable, inexpensive royalty-free sample banks are available.

I will be using my own machine as the primary development environment for this project, which is a MacBook Pro as specified on the attached Project Resource Form. This is equipped with a built-in microphone and sound card, so I will be able to record novel sounds for testing and experimentation.

Starting Point

Knowledge from some of the Part IB courses will prove particularly useful during the project: *Artificial Intelligence I* introduced the perceptron and outlined the operation of neural network classifiers; *Mathematical Methods for Computer Science* developed a rigorous basis for Fourier methods; *Programming in C and C++* introduced the two languages.

Over the summer of 2009, as part of a UROP at the Computer Lab with the DTG, I experimented with steganography and the representation of arbitrary data as sound. This was my first introduction to digital signal processing, specifically the Fourier transform. As part of my IB Group Project, I was able to implement a guitar effects processor on an embedded platform in C++, which gave me valuable experience with the language.

Substance and Structure of the Project

Implementing a timbral classifier entails two major research surveys: first of statistical correlates of timbre, and second of real-time classifier architectures. For instance, in addition to the measures mentioned in the Introduction, perceptual transforms can be applied to the power spectrum so as to better approximate the response of the human ear.

The neural network classifier is one possible choice among many, so I will need to evaluate the suitability of both classifier architecture and implementation. I will also need to consider how to compare the features of two sounds of arbitrary length, or whether instead to specify a fixed size for input sounds.

The project will be coded mainly in C++. This choice of language should provide the performance required for real-time feature calculation, while still allowing the application of object-oriented design. The project can be split into ideally independent modules:

- The *audio input* module will handle the reading in of audio files from disk, including the relevant decompression and format standardization. Eventually, facilities for live audio input could be included as an extension.
- A *preprocessing* module will prepare the audio material for analysis. For instance, by normalizing and then removing any silence from the beginning and end of the sample. As an extension, echo cancellation of some kind could be applied to extract the underlying sound
- The *feature calculation* module will generate a feature vector for the preprocessed sound by applying each of the chosen *feature extraction* procedures. This is the core of the project, and the choice and implementation of features will be key to the efficacy of the classifier.
- The *classifier* module will use an existing classifier implementation, with the feature vector provided by the previous stage as input. A *training set* will also need to be developed mapping sounds to labels.

Evaluation will take two forms. First, the production of numerical measures of the trained classifier's effectiveness on a test set of labelled sounds. Second, an evaluation of how well the system's classifications correlate with a human's perceptual judgements when provided with novel sounds.

For the human portion of the evaluation, novel sounds would be drawn from a single domain at a time, having a significant number of classification categories (i.e. 5 or more, so as to reduce the chance of random agreement). Participants would be played the sounds over headphones in a quiet room, and asked to classify each according to some familiar taxonomy. These judgements would then be compared with those of the classifier (having been trained on the relevant domain).

I will be using the Git version control system for all project files, including the dissertation. This will be mirrored to a backup repository in my PWF space with each change. I will also be updating a project wiki page with ideas, notes and a log. It will be sited at `http://kudos.chu.cam.ac.uk/kwiki/index.php/Rhodri_Karim` and backed up daily.

Success Criterion

For the project to be deemed a success, a classifier should be produced which agrees with 75% of human judgements on a domain with five possible classifications. For instance, if the classifier is trained on the elements of a drum kit, it should be able to differentiate between the sounds of a snare, kick drum, cymbal, tom-tom and closed hi-hat.

Timetable and Milestones

Slot 1: 22nd October – 4th November

Set up development environment, including backup. Research into correlates of timbre. Initial implementation of feature extraction procedures. Implementation of basic audio input and preprocessing modules.

Milestones: Written record of research results. Working pipeline up to feature extraction.

Slot 2: 5th November – 18th November

Research into viable real-time classifiers. Collection of a small training set. Trials of a number of classifiers with current feature extraction implementations. Choose a classifier implementation.

Milestones: The beginnings of a training corpus. Working pipeline up to classification.

Slot 3: 19th November – 2nd December

Begin to plan preparation and implementation chapters of dissertation. Refine choice of features and improve their implementation. Complete Michaelmas supervisions.

Milestones: A working classifier. Outline for preparation and implementation chapters.

Christmas Vacation: 3rd December – 6th January

Continue refining implementation to improve performance. Complete any outstanding implementation milestones. Write preparation and a draft of the implementation chapter. Build up training set. Vacation. Michaelmas term revision.

Milestones: A better-working classifier. A wider training set. First draft of preparation and implementation chapters.

Slot 4: 7th January – 20th January

Choose numerical measures of classifier effectiveness. Begin to plan psychological testing procedures. Review code with supervisor, make any suggested changes.

Milestones: An evaluation plan. Implementation ready for progress report.

Slot 5: 21st January – 3rd February

Write the Progress Report detailing the code produced, some example runs and extensions likely to be incorporated.

Milestones: Progress Report submitted. Project reviewed personally and with overseers.

Slot 6: 4th February – 17th February

Build numerical testing framework for classifier. Collect data for numerical evaluation. Finalize psychological testing procedures. Recruit friends for testing.

Milestones: Numerical evaluation data collected. Participants agreed for psychological testing.

Slot 7: 18th February – 3rd March

Psychological testing and evaluation. Compute statistics based on results. Ensure that the data collected is sufficient to draw conclusions from.

Milestones: Psychological evaluation data collected.

Slot 8: 4th March – 17th March

Detailed plan of the evaluation chapter. Complete the implementation chapter. Complete Lent supervisions and term-time work.

Milestones: Full set of evaluation data. Evaluation chapter planned.

Easter Vacation: 18th March – 21st April

Complete any outstanding evaluation milestones. Write the introduction, evaluation and conclusions. Lent term revision. Vacation.

Milestones: Completed full first draft of the dissertation, with preliminary proofreading.

Slot 9: 22nd April – 5th May

Finalise dissertation, including any diagrams. Review whole project, check the dissertation, and spend a final few days on whatever is in greatest need of attention.

Milestones: Completed final draft of dissertation, proofread multiple times personally, by friends and others.

Slot 10: 6th May – 19th May

Complete any dissertation work left before submission. Easter term courses and exam revision take priority.

Milestones: Submission of dissertation.