Detecting Advertising in Radio using Machine Learning

November 8, 2007

Robin Müller-Cajar
rdc32@student.canterbury.ac.nz
Department of Computer Science and Software Engineering
University of Canterbury, Christchurch, New Zealand

Supervisor: Dr. Brent Martin
brent.martin@canterbury.ac.nz

Abstract

We present an algorithm that can distinguish between advertising and music without understanding it, by extracting key attributes from a radio audio stream. Our method combines advanced filtering of an audio stream with machine learning algorithms to recognise the filtered variables. The result is lightweight enough to run on an embedded processor, and could thus be used to create a device that gives the listener the ability to filter advertising from radio broadcasts.

Contents

1 Introduction

Advertising has become ubiquitous and pervasive in almost every aspect of life. Whole companies like Google depend on it to provide income [11]. Several reports state that the average American person is subjected to 500 to 3000 advertisements every day [3, 6]. Not only does this have implications on our health [31, 28], but it is quite simply annoying. Our project focuses on mitigating this problem by creating the Intelligent Advertising Detector (IAD). This application recognises advertising in radio broadcasts, making it possible to avoid it.

We chose Radio as the target domain of our application because of the passive way it is enjoyed. Unlike vision, humans cannot disable their hearing. This, combined with the fact that people ususally perform other tasks while listening to the radio, makes it very hard to avoid advertising on the radio. In contrast the viewer can simply mute television commercials, or ignore printed advertisements. Radio is the only medium where the listener first has to perform a task switch to avoid any advertisements. Also, as there are no visual cues, there is no way to know when the advertisements have finished.

Our problem is: How do we recognise advertising, without understanding it? It is easy for a human to distinguish between different audio contents as they can assign meaning to what they hear. Although speech recognition has advanced markedly to the stage where basic automatic transcriptions of news broadcasts have become possible [10] this approach is still computationally expensive. Thus our method cannot rely on understanding the content it receives. Instead we focus on identifying key features in the audio stream that are only present in advertising. These features need to be consistent across most, if not all, advertisements, while also being simple to calculate.

Thus our research is split into several key steps:

- Research and discover features that differ between advertising and music
- Implement algorithms that can extract attributes corresponding to these features
- Use machine learning algorithms to create a function with these attributes as inputs, and the class (advertising or music) as output

In the background and related work section we discuss the approaches of other researchers that have also attempted to classify an audio stream, either to discover speech or also to detect advertising. We also describe some machine learning algorithms central to our project.

Our method chapter starts with the analysis of advertisements. This results in the discovery of key features that can be used to distinguish between advertising and music. We implement algorithms to extract these in section 3.3, before describing the design and implementation of several different machine learning algorithms in section 3.4.

The results of testing the machine learning algorithms are described in chapter 4, along with the quality of the extracted attributes. Our discussion chapter then attempts to explain each classifier and its performance. Finally we discuss our conclusions and future work in chapter 6.

2 Background and Related Work

2.1 Advertising

The principle of advertising is as old as trade itself. It has been defined as:

> Making known; calling public attention to a product, service, or company by means of paid announcements so as to affect perception or arouse consumer desire to make a purchase or take a particular action. [19]

There have been enormous numbers of studies that have analysed the content of advertisements, both for radio and other media [1, 7, 14, 17]. The problem of these studies is that they concentrate on the effectiveness of advertising by analysing its information content. What they do not do is differentiate advertising from the media surrounding it, by defining or describing the features that differ.

Thus researchers that want to recognise advertising need to create their own analysis of what distinguishes it from the remaining media. Most researchers did this following the path of least resistance, identifying one key feature that consistently identified all advertisements. Duan et al. for example focus on the fact that commercials on television do not contain a station logo [8], while Saunders relies entirely on the fact that radio advertising contains speech [25]. Herley does not specifically detect advertising but instead all repeat content, but notes that it is possible to identify it by its length [12].

2.2 Extracting features from an audio stream

Clearly we cannot use the raw audio stream as input to any machine learning algorithm, as this would certainly overwhelm it. Instead we will extract those features that can identify advertising, by using algorithms that measure these features.

Pfeiffer, Fischer and Effelsberg have managed to automatically extract some content and meaning from audio [21], as part of the University of Mannheim Movie Content Analysis (MoCA) project. They provide an audio analysis framework that can be used to extract several key attributes from music. They use the framework to detect violence, such as screams and gunshots in movies. Apart from extracting the volume, pitch and frequencies, they also look at the onset and offset of an audio signal, which measures how fast a hearing cell responds to a signal.

Of more interest to us is their attempt to classify music by determining the fundamental frequencies of the chords played. This could be of interest as these funda-

mental frequencies are less likely to be present in advertising, which often does not contain chords. But their method is not yet advanced enough to provide consistent frequency resolution [21], nor has it been tested on speech to see how well it could handle no chords being present.

There has also been some research into the detection of speech in any audio source. Scheirer and Slaney have created a classifier that can accurately distinguish between speech and music when both are separate, but only to 65% accuracy when distinguishing between speech, music and simultaneous speech and music [26]. The attributes they extract are also volume, the energy distribution of the frequencies, the spectral "Flux", the Zero-Crossing Rate (ZCR) and a simple metric to measure the beat of the audio. The spectral "Flux" measures the change in frequencies over time, while the ZCR represents "the number of time-domain zero-crossings" [26]. Both of these are differ between music and speech and could thus be used to recognise advertising.

Srinivasan et al. manage to classify mixed audio to an accuracy of 80% [29] using frequency analysis and the ZCR. A similar approach is used by Itoh and Mizushima, who used the identification of speech to reduce noise in hearing aids [15]. The latter also use the spectral slope of the input signal, which is particularily useful to distinguish between speech and background noise. Another algorithm relying on the ZCR is presented by Saunders. He uses it to distinguish between music and speech to a 98% accuracy across the training data.

Lu, Zhang and Jiang have also created a method that can classify several different types of audio (speech, music, silence, environment sound) and can even distinguish between different speakers [16]. This method advances on the work done by Scheirer and Slaney [26] and Pfeiffer et al. [21] as it manages to produce satisfactory results even when more audio classes than speech and music are used. They measure the variation of the ZCR, the number of silent frames in speech and the spectral flux before using several machine learning algorithms to classify the measured attributes.

Another system that detects the presence of speech even when mixed with other audio has been proposed by Hoyt and Wechsler [13]. They use both Radial Basis Function (RBF) classifiers and a formant tracking algorithm with mixed results. Although they propose combining their algorithms to improve the result, they do not actually do so.

2.3 Detecting advertising

Duan et al. have attempted to detect advertising in Television, using semantic content analysis [8]. By creating algorithms that could accurately detect the boundaries of commercials and using Support Vector Machines to classify them they managed to correctly classify commercials with up to 85.8% accuracy. Most of the features extracted focused on the video stream, thus making them of little use to our project.

Herley has also created a method to find advertising in radio broadcasts [12]. His method focuses on the fact that advertisements are repeated. Thus by building a database of known advertisements using fingerprints, it is possible to avoid the repeat content. Combined with knowledge on how long each segment is, this can be used to avoid advertisements but not repeated music. The main advantage of this method is that it avoids the fundamental problem of distinguishing between

the DJ and advertising, as the DJ does not produce repeat content.

2.4 Statistical Classification and Machine Learning Algorithms

Effectively our method concentrates on separating the audio stream into two main classes: Advertising and Music. This is a simple statistical classification and as such requires machine learning algorithms capable of solving this problem. We discuss some of the algorithms that can and have been used, along with WEKA a framework that makes it easy to quickly test different algorithms.

2.4.1 WEKA

Instead of implementing every machine learning algorithm manually, it makes better sense to use a standard framework of implemented algorithms for training. Those algorithms that produce promising results can then be implemented and used in our application. The Waikato Environment for Knowledge Analysis (WEKA) project by Frank and Witten [32], provides this framework. It is an environment where one can rapidly prototype different machine learning algorithms, using a standard data format. Their framework is better suited to train algorithms than it is to use the algorithms as classifiers; therefore any resulting function needs to be ported into another program for testing.

2.4.2 Linear Regression

Linear regression models the relationship between a dependent variable and several independent variables. In our case the dependent variable is the output class c, while the dependent variables are the attributes a filtered from the audio stream. It works by creating a linear function that best separates the output classes.

Essentially it creates a linear equation of the type:

$$c = \beta_0 + \beta_1 a_1 + \beta_2 a_2 + \ldots + \beta_n a_n \tag{2.1}$$

Where each β_i is a constant derived from the training data. WEKA uses the Akaike information criterion [2] to determine the goodness of fit of the resulting equation.

This algorithm by definition can only learn linearly separable data. Although its simplicity may mean that it cannot fit the model, it is ideal as a benchmark for data that is possibly linearly separable.

2.4.3 Decision Tree algorithms

The Iterative Dichotomiser 3 (ID3) decision tree algorithm, and its later advancement J48 are well suited to the problem domain. Invented by Ross Quinlan [22], the algorithm works by splitting the search space (the space of all possible input values) along the attributes that provide the highest information gain. This procedure is repeated as the algorithm descends down the tree of attribute splits until all test values (where possible) are accounted for. J48 improves the final tree by pruning it to remove branches that have too little information gain. Clearly this means that the final tree will no longer fit the training data exactly, but in general this improves performance especially when used with noisy data, as it prevents the tree from overfitting to the data.

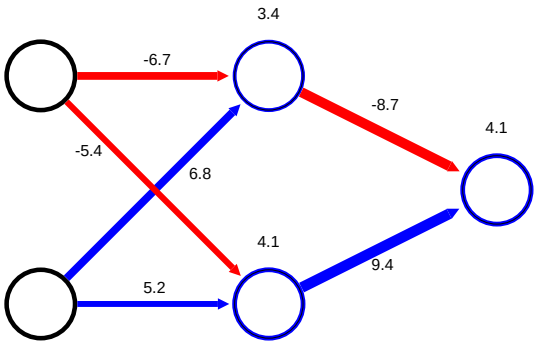

Figure 2.1: Feed forward neural network trained on the XOR function. The value of each weight and threshold is represented by its colour and thickness.

This is a greedy algorithm, as it does not search for the best possible tree, but only the attribute with the highest information gain to branch on at each point in the tree. This can be a problem as this attribute may not be the best attribute to split on. This is because the information gain is calculated by subtracting the entropy of an attribute from the entropy of the data set. Yet the entropy of an attribute tends to be low when the attribute has many different values. For example if we were to include the length of the audio segment analysed so far as an attribute to ID3 this would produce a different value every time. ID3 would split first on this attribute, producing a useless result.

If we ensure that the attributes passed to this algorithm are meaningful, this algorithm should be well suited to this task.

2.4.4 Artificial Neural Networks

These attempt to simulate the function of the brain, which makes decisions based on the collective output of billions of neurons. Each neuron fires whenever its inputs exceed a certain threshold value. The output of each neuron then is used as input to the next neurons, which repeats the process. ANN and their predecessors Perceptrons have been studied since 1957 [24]. Although a single neuron can only learn linearly separable equations [18], a feed-forward network using hidden layers can learn approximate every possible continuous function [4]. Note that it is a NP-complete problem to actually design a network for each arbitrary function.

Several different flavours of ANNs exist. We will concentrate on a simple multilayer feed-forward network trained using the back-propagation algorithm. An example network is schown in figure 2.1.

This type of network consists of an input layer, any number of hidden layers and an output layer. Each hidden layer can have any number of units, each of which is a simple perceptron. From each unit in each layer there are a number of connections to the next layer. In a feed-forward network all connections move strictly towards the output layer. Each connection has a weight associated with it, which is used to emphasise certain connections. Additionally each unit has a threshold which defines at which summed input value it begins to produce output.

Thus to train a feed-forward network we provide a set of example inputs and the expected output associated with them. We run the inputs through the network

and compare the actual output with the expected output. The difference (or error) is then used by each unit in the network to scale the weights of its inputs, by attributing some of this error to each input. This scaling of weights starts at the output layer and is then back-propagated through the network to the input layer.

The output (activation value a_i) of each unit i in this network is defined by sigmoid function of the input in_i. This function (shown in 2.2) is necessary as the backpropagation algorithm requires the inverse of the activation function to calculate the change in weights.

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

The network we are using cannot retain any state, because the activation from previous time steps plays no part in the current time step [20]. This means it has no internal state other than its weights. It is well suited for the problem at hand, as neural networks are good for continuous inputs and are noise tolerant [20].

2.4.5 Hidden Markov Models

Unlike the algorithms above hidden Markov models (HMMs) work on a fundamentally different premise. They are commonly used in tasks such as speech-recognition [23]. Unlike in ANNs where one network is used to produce every possible output class, we create a separate model for each output class. The model is either trained using some form of training algorithm [5], or created to fit every possible output that a class can create. To classify unknown output we then compare the output to each model and calculate the probability that this model has produced the given output. Clearly the model with the highest probability is then taken to be the producer of the output and therefore the class of the output.

Hidden Markov models are a doubly stochastic model [23]. Each model consists of a set of states. Each state has two sets of probabilities: The probability set of possible outputs, and the state transition probabilities.

At each point in time the model will be in a certain state. It then produces some output dependent on its output probabilities before changing state dependent on its state transition probabilities. Note that there is nothing to stop the state transitioning to itself as the next state. An example model is shown in figure 2.2. Here $b_{i1}, \ldots, b_{ij}$ correspond to each value in the output probability set of state i, while $ai1, \ldots, aij$ corresponds to each value in the state transition probabilities of state i.

An example of the use of HMMs is by Duan et al. [8], who train them to detect audio scene changes in television commercials.

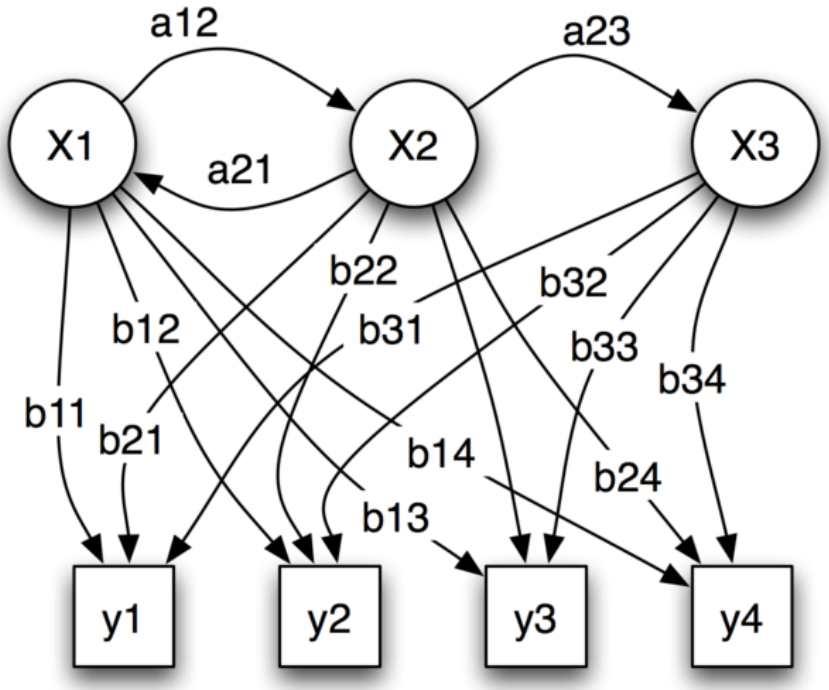

Figure 2.2: An example Hidden Markov Model. The circles represent stated, the rectangles possible outputs. (From [30])

3 Method

Our project focused on creating an application framework that could solve the problem outlined above: How can we detect advertising without understanding it? To do so we first analysed advertising and attempted to find several attributes that would be unique when compared to other audio. Using these attributes we then created an application split into two main parts. First we created audio analysis software containing several algorithms that extract the attributes discovered earlier. Then using WEKA we first prototyped several machine learning algorithms, before building several applications that could read the output from the audio analyser and decide on what type of audio it came from.

Finally we tested these classifiers both separately and in combination to produce an application that can accurately separate music and advertisements.

3.1 Analysing Advertising

We decided to perform a small study on advertisements to explain the distinguishing features of advertising when compared to music, as there has been little previous research to find these features. Using advertising recorded from several different radio stations, we were able to find several attributes that differed from music.

Our study focused on commercial radio stations that could be received in Christchurch. We made the decision to focus only on contemporary music, as this makes up the bulk of music on the radio. We recorded several hours of radio from 91.3 ZM, More FM, The Edge and The Rock as test data. These recordings were taken at different times of day between 8am and 5pm in the first week of June. The advertisements that were captured in this manner were then analysed and compared to music in the same segments.

This gave us the following key attributes:

Speech This was the key variable that was present in almost all advertisements. Obviously to sell something the advertisers had to spread their message using verbal information. Noteworthy are advertisements for Halswell Country Florists and Team Hutchinson Ford; Both of this consisted entirely of music, with the text sung in tune. In contrast a significant portion of advertisements contained only speech, while some contained either music or other sound effects in the background.

Rhythm of Speech Compared to music and the DJ, advertisements sometimes appeared rushed, as traders attempted to squeeze as much information into as little time as possible. There were some advertisements that did not follow

this rule, such as the Lincoln University ad, which was spoken in a very low and slow pitch.

Frequencies and Frequency Distribution Advertisements contain fewer frequencies, but those frequencies have been compressed to make the audio appear louder. The range of frequencies tends to be smaller than for music, when the advertisement consists of pure speech. For all advertisements the frequency balance is skewed more towards lower frequencies.

Volume and Volume Distribution The silent gaps between words mean that the average volume of speech is lower. Also the change in volume over time is higher for advertisements for the same reason.

Time Advertisements are usually aired in chunks and in front of information the listener is interested in. Thus the likelyhood of advertisements being aired increases before the full hour, as this is before the news broadcast. Interestingly even when no news broadcast is aired, the commercials appear at the same time.

Length of Segment An advertisement is never longer than a minute, while music is significantly longer. Radio DJ announcements can be any length, but also tend to be longer than a single advertisement.

By extracting these attributes (where possible), it should be possible to train a machine learning algorithm on them. This should enable the accurate classification of advertising in radio broadcasts.

3.2 The Intelligent Advertising Detector

To classify advertising we created a software framework that can analyse audio and then classify the audio based on the extracted attributes. The nature of the task led to a two-tiered approach. By building two separate applications, one to analyse the audio and one to classify it, we can easily test several algorithms and classifiers without creating a mess of the source code.

This reasoning led to the software seen in figure 3.1: We have created an audio analyser and controller, and several machine learning and statistical analysis applications. The audio analyser receives the audio stream, filters out attributes and passes these to the machine learning applications. The machine learning applications then classify the received attributes and return a class to the audio analyser. Depending on the return value, the audio analyser then performs an appropriate action. In this test version the playback volume is simply muted whenever the return value indicates that the current audio is advertising.

3.3 Audio Analyser

For each of the distinguishing features of advertising that we have identified earlier, we needed to create an algorithm that can extract this feature. Some of these algorithms used the temporal domain of the audio, while others relied on the frequency domain. Therefore we converted all the audio input using the Fastest Fourier Transform in the West (FFTW) [9] library.

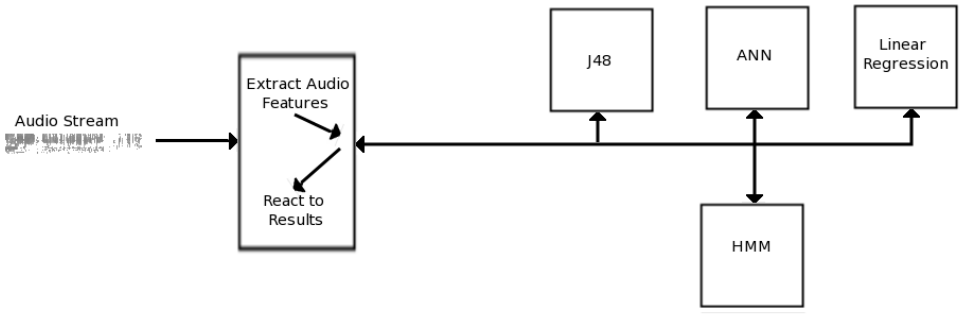

Figure 3.1: Overview of the Intelligent Advertising Detector

Due to the logarithmic nature of music the resulting frequencies need to be recalculated to obtain a linear distribution. To obtain a note one octave higher than the current note, its frequency has to be doubled. As western music is based entirely on tones and semitones, we rescale the frequencies so that each frequency value corresponds to a quartertone. This means that higher notes contain a larger range of frequencies than lower notes. Appendix A contains the C++ code necessary to convert any logarithmic frequency array into a linear one.

Audio was split into samples of duration 100ms, at a sampling frequency of 44100Hz. This sampling time was taken to ensure that every audible (down to 20Hz) frequency has at least two full cycles in each sample, while at the same time remaining fine grained enough to quickly notice changes in frequencies. The statistical classifier algorithms where then passed one set of attributes every ten samples. This means that they have enough time to return a result before the next set of attributes arrives.

3.3.1 Obtaining Relative Rating

One of the major problems of this project was the fact that most attributes filtered by the audio analyser were only important relative to each other. For example the volume of an audio clip is only significant if we can compare it to other volumes at different positions and therefore state that it is louder or softer than the rest of the audio clip. We cannot use a statically defined threshold, because different radio stations transmit at different strengths.

Thus every attribute was scaled relative to its average (mean) value. When training and testing the algorithm the mean values of the entire training set were used. Clearly this is not an option when listening to music that has not been prerecorded. In this case we use the mean of all the values extracted since the application has started. This mean attribute value ($\overline{a}$) is easy to calculate using the formula:

$$\overline{a}_t = \frac{(t-1)\overline{a}_{t-1} + a}{t} \tag{3.1}$$

Where t is the number of samples that have been received so far, and a is the value of the attribute. When using this mean, the first few scaled attribute values are inaccurate as the mean is too close to the current sample value. After approxi-

mately 100 samples, the running mean $\overline{a}$ has approached the true mean enough to produce meaningful results.

3.3.2 Volume

This attribute simply describes the power of the input signal. The volume v of any sample consisting of an array of n individual values x, is calculated using the function:

$$v = \sum_{i=1}^{n} x_i^2 \tag{3.2}$$

This value is used to detect speech and the rhythm of the audio sample. The average volume for speech tends to be lower than for music, as speech contains pauses, which music does not. Volume patterns tend to be more constant for music than for speech, as the latter contains many more peaks and troughs in each sample period. By summing the number of significant peaks, we also have a crude measurement of the pace of the audio sample.

3.3.3 Frequency Distribution

We extract several attributes to obtain a measure for how evenly frequencies are distributed across the spectrum. This again targets the speech in advertising, as speech has a narrower frequency spread than instrumental music.

Energy Balance

The energy balance compares the sum of the frequencies above 8000Hz with the total sum of all frequency values. This value is significant because speech contains few frequencies above 8000Hz while instruments contain significant amounts of frequencies in this range. This does not always hold as certain music genres such as rap often contain only speech and drums, and therefore very few high frequencies.

To calculate it we first averaged the values of ten consecutive samples in the frequency domain together before compressing the resulting array of quartertones into six individual values to obtain the frequency array f.

The energy balance b is then calculated using the formula:

$$b = \frac{\sum_{i=5}^{6} f_i}{\sum_{i=1}^{6} f_i} \tag{3.3}$$

Number of silent frequencies

This value targets the volume distribution apparent in pure speech. By summing the number of empty frequencies, we can again obtain a value for the frequency distribution of the received audio.

The value does not consider every frequency recorded but instead picks 32 significant frequencies from the 300–2000Hz frequency range. The array of frequencies fs is calculated using the equation:

$$fs_i = 2^{\frac{i}{2.89}} + 300 \tag{3.4}$$

Where $0 \leq i < 32$. This equation has been chosen because at $i = 0$, $fs_i = 301$ and thus 300Hz, while at $i = 31$, $fs_i \approx 1995$ and thus close to 2000Hz. The function is exponential and as it is used over the uncompressed frequency band it ensures that we use more low frequencies than we use high frequencies. This is important as the higher frequencies are less significant (see section 3.3 for an explanation of western music and human hearing).

The frequency range has been chosen as it contains most of the significant frequencies of the audio stream. Due to noise we consider all frequencies that have a value below 15% of the average frequency energy ($\overline{f}$) to be empty. The advantage of such a high threshold is that it is very noise tolerant, and can even ignore quiet music in the background of some advertisements.

Mathematically the number of silent frequencies n_s is extracted from the array of uncompressed frequencies f as follows:

foreach fs_i *in* fs **do**
  if $fs_i < 0.15\overline{f}$ **then**
    n_s++;
  end
end

Algorithm 1: Calculating the no. of silent frequencies

Extreme Frequency Values

This value again measures the volume distribution of the audio stream. Advertising contains some very strong frequencies because it tends to heavily emphasise the speech frequencies, something that music usually does not. By summing the number of frequencies above and below a certain threshold we obtain a good estimation for the general volume distribution.

The algorithm used to calculate the number of frequencies n_{ex} in the quarter-note frequency array f, not close to the average frequency value $\overline{f}$ is as follows:

foreach f_i *in* f **do**
  if $f_i < 0.2\overline{f}$ *or* $f_i > 3\overline{f}$ **then**
    $n_{ex} + +$;
  end
end

Spectral Flux

This is the value proposed by Scheirer and Slaney [26]. Again this attribute focuses on the fact that music usually uses more frequencies than speech. These frequencies tend to change rapidly as new notes are played. This contrasts with speech where the speaker tends to remain within a thin frequency spectrum, with most frequencies remaining empty. Thus if we sum how much each frequency changes from one sample period to the next, we can gather a value for how much the audio

has changed in that period. This value will be higher for music or noise than for speech.

To calculate the Spectral Flux S_f we perform the calculation:

foreach *Sampled array* f **do**
    $S_f += \sum_{j=1}^{n} |f_{j,t-1} - f_{j,t}|$
end

3.3.4 Detecting Audio Edges and Segment Length

Here we use a similar measure to the spectral flux discussed above. Instead of measuring the change in frequency since the last sample period, we compare the change in frequency to the average frequencies of the last ten sample periods. If this value is high, so is the chance of an edge. By remembering the time since the last edge we can find the length of the current audio clip, giving us the ability to sort clips by length.

The edge value e_v is calculated using the formula:

$$e_v = \sum_{j=1}^{n} |\overline{f}_j - f_{j,t}| \tag{3.5}$$

Where $\overline{f}_j$ is the mean of the j^{th} frequency for the last ten samples; $f_{j,i}$ is the value of the j^{th} frequency at time t.

This value is then used as a basis for calculating the length of the current segment. This length value is simply a counter that increments every second and is reset whenever an edge occurs.

3.4 Machine Learning Algorithms - Design and Implementation

Once we have extracted key audio characteristics, we need to decide what class the audio stream they were extracted from belongs to. Theoretically we could manually create a function to do this, but practically this is too hard to be realistic. Thus we train machine learning algorithms with some training data, and then use these algorithms as classifiers.

The training data consisted of 200 attribute sets, half of which originated from music, the other half from advertising. Each algorithm was initially trained and tested on this set.

The machine learning algorithms are chosen by comparing their performance over the training set with the performance of the linear regression algorithm. Any algorithm with a higher accuracy over the training set is used. We chose linear regression as a benchmark because initial results showed our data to be almost linearly separable (shown in figure 4.1). Another prerequisite to any algorithms chosen is that it has a high noise tolerance. This is necessary due to the analog nature of radio broadcasts, and because of the large amounts of data that cannot be uniquely classified (DJ talking over top of a song, advertising for a concert or new CD).

We used WEKA to test most algorithms. This algorithm was tested and implemented purely in C, because WEKA does not have an implementation of Hid-

den Markov Models (HMMs). The other algorithms were then implemented using C/C++, communicating with the audio analyser using Linux FIFO pipes. This makes it possible to 'plug in' different classifiers to the audio analyser — it does not know which classifiers are being used, it only knows the output that is returned.

Below we describe the implementation of each algorithm that passed all the required prerequisites.

3.4.1 Artificial Neural Network

Our ANN was implemented by rewriting code by Paras Chopra. It is a simple feed-forward network that is trained using the backpropagation algorithm. We used a fully connected network, because connections that are not needed will receive a low weighting when the network is trained.

Our design was based on Occam's razor — the simplest network that works is probably the best one. This method also meant that we would avoid the ANN overfitting itself to the training data. By starting with a simple perceptron with an input for every audio property (six in total), we expanded the network until it could train itself on the training data provided.

Surprisingly the resulting network performed poorly across the test set (76% accuracy). Thus we continued increasing the network size and retraining until we reached a plateau in result accuracy across the test set. The resulting network is shown in figure 3.2. The complexity of this network makes it more likely that the data is overfitting to the training set, but its improved performance across the test set indicates otherwise.

3.4.2 Decision Tree

The decision tree algorithm used was J48, an advancement of the ID3 algorithm. We chose J48 over ID3 because it enables the use of continuous data and it includes pruning to provide a better fit to noisy data.

The tree was built using WEKA, and validated using 10-fold cross validation. The resulting decision tree was then hard coded into a C++ application using if-else statements.

3.4.3 Linear Regression

Once again the actual function was created and cross-validated using the Weka framework. The resulting function was then transferred into C++ and used as a benchmark to compare the performance of any other algorithms against.

The function implemented calculated the output o as follows:

$$o = -0.2889v + -1.0235b + 1.5603n_{ex} + -1.5777S_f + 0.6224 \tag{3.6}$$

Here v is the volume at the point in time, b is the energy balance, n_{ex} the number of extreme frequency values, and S_f is the Spectral Flux.

3.4.4 Hidden Markov Models

These were implemented by utilising the Generalised Hidden Markov Models library by the Max Planck Institute for Molecular Genetics, Germany [27]. They were

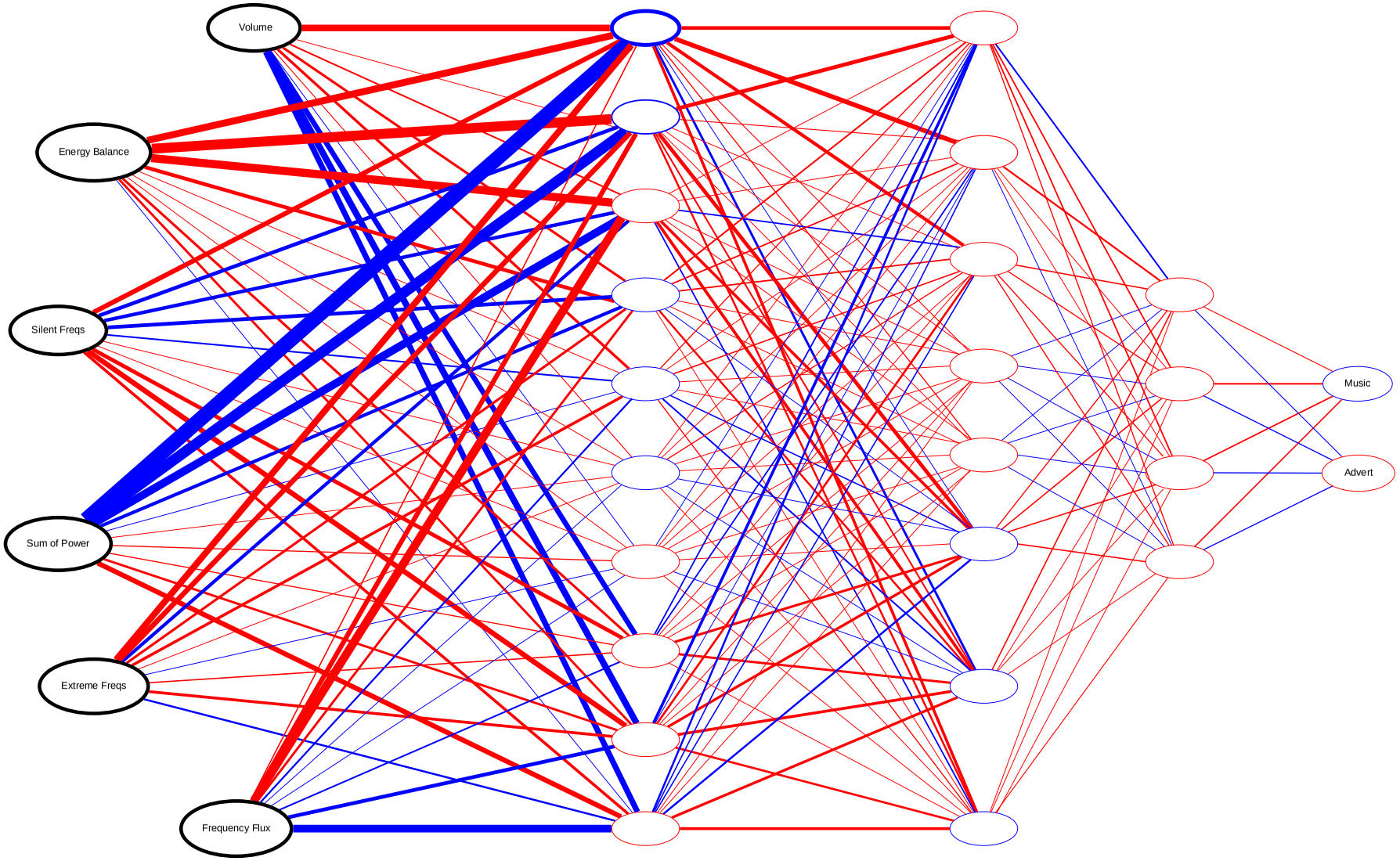

Figure 3.2: The Final ANN used to classify audio data: The thickness of each dendrite corresponds to its weight. Red indicates a positive value, blue a negative value.
Note that some of the inputs have significantly higher weightings than others, these correspond to the attributes with the highest information gain. Also note that the outputs are simply mirror images of each other.

trained on the same classes as all the other algorithms: advertising and music. For each class we trained one HMM using the Baum-Welch algorithm.

The input data into the models is the sequence of volume levels in the audio stream. Every sample period the current volume is measured and stored. Hidden Markov models require discreet outputs, so the volume levels were scaled into ten equally sized containers. The resulting array of discrete volumes in the range of one to ten is then passed into both Markov models every second. Using the Viterbi algorithm, the classifier application decides which model is more likely to generate the input sequence of volumes.

As with the ANN, our models were designed by starting with the simplest possible model — a single state. The number of states was then increased linearly, until the model stopped improving across the training set. Although both models were designed independent of each other they both ended up with the same number of states (see figure 3.3). Thus both models use seven states, but have completely different connection and emission probabilities.

3.5 Improving Initial Performance

After testing the algorithms across some our test data we noticed several key issues.

The output of our machine learning algorithms can flip from one second to the next, because none of the algorithms remember their past state. This was especially a problem in borderline data. Thus we decided to smooth the output of each algorithm, by using its running average as return value. A true running average of order n would have meant caching n past results, so we calculated an approximation $\overrightarrow{o}$ as follows:

$$\overrightarrow{o}_i = 0.9\overrightarrow{o}_{i-1} + 0.1o \tag{3.7}$$

Where o is the true output of the algorithm. This was close enough to the true running average for our purposes. This was done for each algorithm, at which point they were retested.

By averaging the output we obviously produce errors everytime the class changes, for example at the beginning or the end of an advertising break. But as these class changes are reasonably rare, the introduced errors should be offset by the increased accuracy during a class.

The ANN, J48 and Linear Regression produced very similar results, which is not surprising as they all received the same input sets. Their outputs tended to err towards false positives (music classified as advertising). The HMM tended to misclassify advertising as music (false negative), which meant that it was more likely to play advertising, but much less likely to skip music. To balance this result we decided to combine the results of several algorithms to provide a more accurate classification of music and advertising. The two algorithms that were combined were the ANN and the HMMs. The ANN was chosen because it produced the most accurate results over the training set, while Markov models were chosen to counterbalance the ANN.

To combine the algorithms we used linear regression to weight the output of our combined algorithms. We trained the regression algorithm with a training set consisting of the outputs of the ANN and HMMS, as well as the expected output at each point in time. The resulting formula (equation 3.8) was then added to the

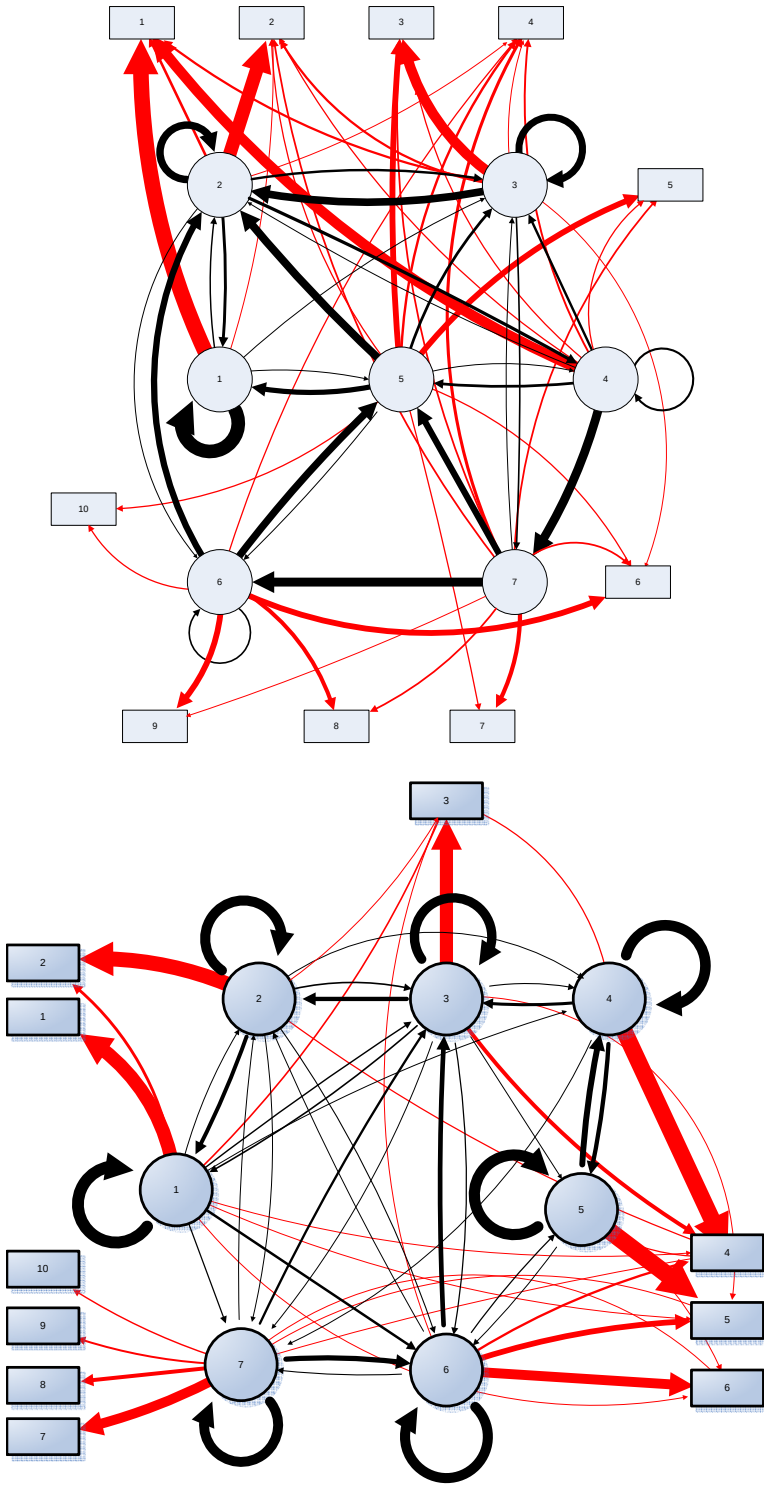

Figure 3.3: The trained Hidden Markov Models. the speech model is at the top, music is at the bottom. Circles are states, rectangles the volume outputs. The thickness of each line is proportional to the probability of this line being followed.

source code and tested.

$$o = -1.1349\overrightarrow{o}_{ANN} - 0.1589\overrightarrow{o}_{HMM} + 1.1265 \tag{3.8}$$

Note that we are using the averaged output of the ANN and HMMs.

3.6 Comparing the different machine learning algorithms

As we had now implemented several algorithms theoretically capable of classifying the audio data, we decided to run a full comparison on the data. The training and test data was 30 minutes of audio recorded from the More FM radio station on the 11th of June. 200 seconds (100 seconds music and 100 seconds advertising) were then taken as training data, using twenty ten second excerpts from throughout the segment. To test the algorithms we used the entire thirty minute segment, playing it from beginning to end and comparing the output of the algorithm with our transcription of the audio.

The test set contained approximately ten minutes of advertising and twenty minutes of music. For the purposes of the experiment the DJ has been classified as advertising. There are several situations where the machine learning algorithms are expected to fail. Examples include two advertisements that are music: An advertisement for the Hopscotch Finance Plan, and another for Christina Aguilera in concert. Also some of the music contained speech, either at the end of a song when the DJ talks over top of the music, or at the beginning of some songs (for example "Grace Kelly" by Mika). Thus the accuracy of each algorithm will be similar to those experienced in real life, where there will be always situations that cannot be classified correctly based on the attributes we extract.

4 Results

4.1 Analysing Audio

Most of the data filtered from the audio stream is almost linearly separable (shown in figure 4.1). Although there are values that cross due to borderline data, most values cluster in regions.

The attributes with the highest information gain as calculated by the ID3 algorithm are volume and the energy balance. In fact using only the volume v attribute separates music from advertising with a 86% percent accuracy across the training set. Energy balance b can be used to separate the classes with a 79.5% percent accuracy, followed by extreme frequency values n_{ex} and spectral flux S_f (both 71.5%). The least accurate indicator is the number of silent frequencies which can be used to classify 68.5% of the training set correctly.

The edge detection algorithm failed to provide meaningful results. Thus the length attribute had to be dropped as its information gain was non-existent.

4.2 Algorithm Results

Our resulting neural network had three hidden layers. As we can see in figure 3.2, the attributes with the highest information gain also had the highest weights as inputs to the network. Thus our algorithm was able to distinguish between the importance of attributes.

Unsurprisingly the speech model had a much higher probability of outputting low energy values than the music model for our Hidden Markov models. Thus its probabilities were skewed towards this. The chance of outputting the lowest possible volume is the highest in the speech model. In the music model (figure 3.3) we can see that there is a high chance of each state returning to itself after each iteration. This indicates that the volume levels do not change as much nor as quickly in music and tend to remain reasonably constant. The probability of outputting a medium volume is also much higher in the music model than the speech model. Appendix B has the actual figures for the model.

4.3 Algorithm Comparisons

When testing the algorithms using the training set each performed similarly well. The neural network performed best with 96.5% accuracy. It had 3% false negatives (fN) and 4% false positives (fP). The Hidden Markov Models (HMMs) followed with 93% accuracy (8% fN, 6% fP). The J48 decision tree (90%, 9% fN, 11% fP) and the linear regression algorithm (88%, 7.5% fN, 16.5% fP) performed the worst. The combined ANN/HMMs algorithm managed receive a 100% accuracy over the

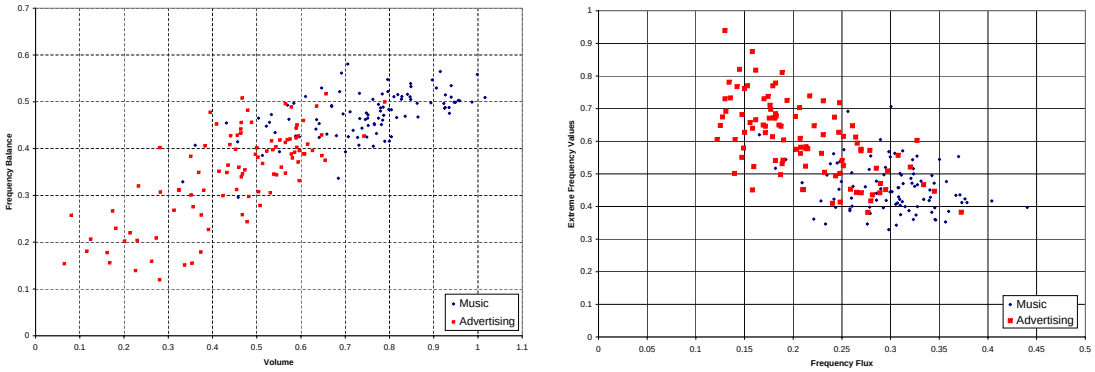

Figure 4.1: Graphs showing the filtered attributes with the highest information gain from the training set. Each point corresponds to the attribute values of one second of audio

	% Overall	% Music	% Speech/Ads
Linear Regression	87.5	91.2	83.8
Linear (Averaged)	88.1	93.5	82.7
J48	88.5	93.9	83.1
J48 (Averaged)	89.7	91.0	88.4
ANN	85.7	86.8	84.6
ANN (Averaged)	89.6	90.7	88.5
Markov	85.8	91.1	80.5
Markov (Averaged)	93.1	96.4	89.8
Combined Algorithm	91.2	92.5	89.9

Table 4.1: Accuracy comparison between different machine learning algorithms. The output for the averaged algorithms was obtained by using a running average of order 10 on the true output.

training set.

When compared with the test set the results are quite different. The combined algorithm does not perform best with its 91.2% accuracy (see table 4.3). In fact the most efficient algorithm is the Markov model with averaged output with a 92.7% accuracy. It is worth noting that the HMMs took advertising as music more often than the J48 and ANN algorithms.

By averaging the outputs we are able to improve the results of all algorithms. This improvement is very small for linear regression and J48 at around 1%. In contrast the HMMs improvem by 6.9% and the ANN improvemes by 3.9%. This also greatly improves the detection of advertising of the markov model to 89.9%.

5 Discussion

The results show that there is enough difference between advertising and music that an algorithm that does not understand the content of the audio can still separate them.

Most of the algorithms performed similarly well, because most of the extracted data was linearly separable. This was because even though the more advanced machine learning algorithms could model functions of a higher complexity this was simply not necessary. Thus even the linear regression algorithm worked almost as well as the other algorithms, with a 88.1% percent success rate. Although all the extracted attributes were reasonably good at separating advertising and music, most were used to detect speech. Thus borderline cases were borderline for most attributes, which meant that there was little gain in combining the attributes.

The large size of the ANN is surprising. As can be seen by the differing input weights in figure 3.2 the network clearly emphasised the values of some inputs more than others. Also there were several weights that were basically non-existent. Perhaps using some form of genetic algorithm to create a not fully connected network would have produced a better result than relying on the back-propagation algorithm to remove weights that seem unnecessary.

The different set of attributes passed to the HMMs caused them to produce significantly different results. The attributes passed to the HMMs were simply an array of volume levels. This meant that the HMM could not detect if the volume came from speech or music. Thus the HMM was very good at detecting the volume patterns of music, but could not determine consistently if there was speech in the foreground, causing a higher number of false negatives. Thus this algorithm was much more likely to classify the input audio as music than as advertising. This contrasted with the other algorithms of which some tended to overemphasise advertising. This could also have been caused by the input attributes which were all primed to detect speech. Thus any music containing speech was misclassified as advertising.

The results show a much lower accuracy for recognising advertising than for music. This was caused by one advertisement for a Christina Aguilera concert which consisted almost exclusively of sound clips of her songs. Obviously every algorithm classified this advertisement as music. This advertisement contributed approximately 10% of the total advertisement time, which explains why none of the algorithms could classify advertising with an accuracy higher than 90%.

Thus the heavy reliance on attributes that detect speech meant that most algorithms failed in the same set of situations:

- Any advertisement that consisted of music was classified as music. In essence

this is not wrong, but at the same time it meant that advertisements were wrongly classified.

- The DJ was consistently classified as advertising. This is something that cannot be fixed by introducing new attributes. Although theoretically there are some attributes that distinguish the DJ from advertising (such as the rhythm of speech), these are not very consistent, and also very hard to classify.

- Some songs simply consisted of speech and music, exactly like advertising. At the same time some advertisements had music included. The IAD managed only about a 56% accuracy with these borderline cases (using HMMs as classifier), which is only slightly better than an algorithm based on pure chance.

Part of the problem was the fact that the entire application focuses on attributes that can be extracted from the audio stream directly. This means that large amounts of meta-data and domain knowledge are lost. Examples of this meta-data are:

Time of day The probability of advertisements increases at certain times of day and certain time of the hour. Simply by recording the time at which advertisements were played over a certain period it should be possible to implement an accurate probability distribution of advertisements corresponding to time. But this distribution would most likely be different for every radio station, and would also have to be dynamically updated, making it hard, if not impossible to implement.

Length of audio segments This attribute is theoretically easy to measure using an excellent edge detection algorithm. Perhaps using the algorithm proposed by Herley [12] instead of the one used would make this possible.

Length of Ad-break This follows on from the attributes above. If we can measure the length of each segment and therefore know how for long advertisements have been played, we can clearly decrease the probability of another advertisement after a certain amount of time. This relies on the fact that the ad-break tends to be around five minutes long.

The issue with using these attributes is that they require the domain knowledge to be correct. There is nothing that stops radio stations only broadcasting two minute ad-breaks every five minutes, which would make the last attribute useless. Similarly, simply moving the time of ad-breaks by a certain amount of time would make the time of day attribute completely useless. Thus these attributes can be easily circumvented or broken, immediately decreasing accuracy.

Using attributes that do not rely on speech would also have helped separate the borderline cases described above. The problem is that no other feature is so common in advertising. Although some advertisements use sound-effects, while others use some form of musical accompaniment, both of these are not common enough to describe a significant portion of advertisements. In fact well over half of the advertisements sampled in our analysis consisted entirely of speech. This also explains why it has been impossible to separate out the DJ from the advertising. As our edge detection algorithm failed to work consistently we could not use the

length attribute which would have been completely independent of speech. This might have improved our accuracy even further.

Combining several machine learning algorithms did not improve the final performance at all. Although the combined algorithm managed a 100% accuracy across the training set, it produced worse results across the test set. Basically the linear regression algorithm used to assign weights to the outputs of the original algorithms overfitted the data to suit the training set, but was unable to generalise to the test set.

By averaging the outputs of the machine learning algorithms we were able to improve some of their performance significantly. This was especially true for the HMMs and the ANN. Especially the advertising detection of the HMMs improved markedly (to 89.8%) when we averaged their output. Thus by averaging their output noisy data was effectively ignored. This result indicates that both algorithms were significantly affected by noise. By averaging the output we are also causing some problems with the classification. Anytime the audio class changes (for example when a song finishes and the DJ starts talking) our averaged output will only slowly adjust to this change, because we do not detect the edge change. Thus we often miss the beginning of a song, and do not regognise the beginning of an advertisement. Once again a good edge detection algorithm would solve this problem, improving the performance of the IAD significantly.

Another possible reason for the need to average results might have been a too short time slice. But using a time slice longer than one second (ten samples at 100ms each) would have meant a significantly longer delay before the IAD reacts to changes in the music.

6 Conclusions and Future Work

We have shown that it is possible to detect advertising in radio broadcasts to a good (93.1%) accuracy. It has shown that the right choice and correct filtering of attributes is much more important than the choice of machine learning algorithm, as all of our algorithms performed similarily well.

The attributes chosen to define advertising in this project were all extracted from the audio stream. This was not a necessity, but makes our application completely independent of outside information. If we had relied on attributes such as the time, our application would by necessity have required another information stream, independent of the audio stream.

We heavily rely on speech detection to classify advertising, as it is the most common identifier of advertising. This is partly because speech detection is well studied in literature compared to other attributes of advertising. But it has the side effect that the radio DJ is consistently (mis)classified as advertisement. By using features from the audio stream that differ between the DJ and speech in advertising this could be avoided, but we have not been able to identify any features that fit this criteria. The reliance on speech was thus both our biggest advantage, but also the reason why a fully correct classification was not possible.

The type of machine learning algorithm used made little difference to the classification accuracy of the application. This contrasts with the significance of different attributes, some of which could separate up to 86% of audio correctly. Thus our design and identification of audio extraction algorithms provided a much larger contribution to the overall accuracy of the application. Surprisingly a Hidden Markov Model trained only on the sequence of volumes over time performed better than all the algorithms using more complicated attributes. Thus it seems as if the change of volume levels over time says significantly more about the audio class than the average volume and all the other attributes together.

Due to the limited time available for our project, there are several steps still required to make our product usable for the end user. Additionally we have not fully explored every possible method of detecting advertising. In the next few pages we will discuss some of the work that could be done to extend and improve our results.

6.1 Use of Speech Recognition

One of the key assumptions of this project was that the application would not understand the radio broadcast. In principle, with speech recognition advancing constantly this assumption no longer needs to apply. By implementing a speech recognition system as part of the audio analyser and then training the machine learn-

ing algorithms on its output, it might be possible to produce the same accuracy at recognising advertising that humans enjoy.

Possible attributes that the machine learning algorithms could be trained on include key words, such as Sale, Buy, and Hurry. This combined with the attributes we are currently extracting could greatly improve the performance of the Intelligent Advertising Detector (IAD).

6.2 Use of Databases

One of the key abilities of humans is remembering. At the moment our IAD has can only simulate this ability by using machine learning (which remembers key features by creating a function that separates classes based on those features). By implementing a database of known advertisements we can combine the ability of the algorithms with the power to remember. It is probably not computationally and spatially viable to store every advertisement, only storing those advertisements that were not recognised might also be an option.

Instead of only creating a blacklist of advertisements, future research might focus on implementing a whitelist of radio DJs. This would make it possible to give the user the choice of listening to the DJ instead of only listening to music.

Implementing a database would require some input from the user, as by definition only data that has been incorrectly classified needs to be put into a database. Thus some form of dynamic user input would also be required.

6.3 Dynamic Training of Machine Learning Algorithms

Currently our machine learning algorithms are only trained once at the beginning. There are two main reasons to provide a facility to dynamically retrain the machine learning part of the IAD.

The first is concept drift. Although the probability of this occurring is not likely to be high, it is possible that music changes over time, until it is no longer recognised accurately by the machine learning algorithms. Thus by retraining the algorithms this could be avoided.

The second reason is the fact that there are very many types and genres of music. The radio stations our application was tested on only played pop/rock music. If an end-user wanted to use the IAD on a radio broadcast that played classical music, the machine learning algorithms would have to be retrained. Of course this problem could be avoided by training the machine learning algorithms on several genres. But this would make the concept that we are trying to learn significantly more difficult, and possibly mean that the resulting accuracy is lower.

6.4 Use of Meta-data

The use of meta data could also help in correctly classifying advertising (see our discussion section). We do not use any meta-data whatsoever, because our project has completely focused on attributes that can be extracted from the audio stream,.

Future research could focus on identifying data that describes the audio stream that is not present in the audio file itself. The next step would be to somehow extract this data and use it to correctly separate advertising and music.

6.5 More Complex Class Structure

The machine learning algorithms used in this project have all been trained for two classes: music and advertising. Clearly this is not a completely accurate representation of a radio broadcast. By creating more classes it might be possible to more accurately separate the audio stream into its individual segments.

An example of classes that might be used is: music, speech, speech and music, noise, silence, sound-effects. These classes fit the range of sounds that can be heard on a radio broadcast much more tightly.

6.6 User Experience and Software Development

The entire development of the Intelligent Advertising Detector (IAD) is still at a research stage. Thus it cannot be used by an end-user effectively.

Our audio application does not have control over the audio that is played. Thus it cannot change radio channel or play another media file while advertising is present. All it can currently do is mute the audio output whenever advertising is present. By implementing the software on a system that contains a radio card, this problem can be solved.

Although care has been taken to make the memory and processor footprint as small as possible, future research could focus on more efficiently implementing the algorithms used. This could go as far as implementing the software on a microcontroller, thus creating a portable device with the software.

6.7 Extending the Hidden Markov Models

The success of the HMMs could be extended further by passing more attributes to it. Perhaps several of the attributes we are filtering provide more meaning when we consider their change over time, instead of using their mean value every second. At the moment only the volume levels are passed in as attributes, but this could easily be extended by using vectors of several attributes.

Bibliography

[1] ABERNETHY, A., AND FRANKE, G. The information content of advertising: A meta-analysis. *Journal of Advertising 25* (November 1996), 1–18.

[2] AKAIKE, H. A new look at the statistical model identification. *Automatic Control, IEEE Transactions on 19*, 6 (December 1974), 716–723.

[3] ARENS, W., AND BOVÉE, C. *Contemporary Advertising*, 5th ed. Irwin, 1994.

[4] AUER, P., BURGSTEINER, H., AND MAASS, W. The p-delta learning rule for parallel perceptrons. submitted for publication, 2001.

[5] BAUM, L. E., PETRIE, T., SOULES, G., AND WEISS, N. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *Ann. Math. Statist. 41*, 1 (1970), 164–171.

[6] BROWER, M., AND LEON, W. *The Consumer's Guide to Effective Enviromental Choices: Practical Advice from the Union of Concerned Scientists*. Three Rivers Press, 1999.

[7] DOWLING, G. Information content in u.s. and australian television advertising. *Journal of Marketing 44* (Oktober 1980), 34–37.

[8] DUAN, L.-Y., WANG, J., ZHENG, Y., JIN, J. S., LU, H., AND XU, C. Segmentation, categorization, and identification of commercial clips from tv streams using multimodal analysis. In *MULTIMEDIA '06: Proceedings of the 14th annual ACM international conference on Multimedia* (New York, NY, USA, 2006), ACM Press, pp. 201–210.

[9] FRIGO, M. A fast fourier transform compiler. In *PLDI '99: Proceedings of the ACM SIGPLAN 1999 conference on Programming language design and implementation* (New York, NY, USA, 1999), ACM Press, pp. 169–180.

[10] GAUVAIN, J.-L., LAMEL, L., ADDA, G., ADDA-DECKER, M., BARRAS, C., CHEN, L., AND DE KERCADIO, Y. Processing broadcast audio for information access. In *ACL '01: Proceedings of the 39th Annual Meeting on Association for Computational Linguistics* (Morristown, NJ, USA, 2001), Association for Computational Linguistics, pp. 2–9.

[11] GOOGLE. Google inc. annual fiscal report dec. 31, 2006.

[12] HERLEY, C. Accurate repeat finding and object skipping using fingerprints. In *MULTIMEDIA '05: Proceedings of the 13th annual ACM international conference on Multimedia* (New York, NY, USA, 2005), ACM, pp. 656–665.

[13] HOYT, J., AND WECHSLER, H. Detection of human speech in structured noise. In *ICASSP '94: IEEE International Conference on Acoustics, Speech, and Signal Processing, 1994* (April 1994), vol. ii, pp. II/237–II/240.

[14] HUNT, S. Informational vs. persuasive advertising: An appraisal. *Journal of Advertising 5* (1976), 5–8.

[15] ITOH, K., AND MIZUSHIMA, M. Environmental noise reduction based on speech/non-speech identification for hearing aids. In *ICASSP '97: Proceedings of the 1997 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '97) -Volume 1* (Washington, DC, USA, 1997), IEEE Computer Society, p. 419.

[16] LU, L., ZHANG, H.-J., AND JIANG, H. Content analysis for audio classification and segmentation. *IEEE Transactions on Speech and Audio Processing 10* (October 2002), 504–516.

[17] MILLER, D., AND MARKS, L. Mental imagery and sound effects in radio commercials. *Journal of Advertising 21* (December 1992), 83–94.

[18] MINSKY, M., AND PAPERT, S. *Perceptrons*. MIT Press, Cambridge, MA:.

[19] MOTTO, M. J. Definition of advertising, 2002.

[20] NORVIG, P., AND RUSSEL, S. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[21] PFEIFFER, S., FISCHER, S., AND EFFELSBERG, W. Automatic audio content analysis. In *MULTIMEDIA '96: Proceedings of the fourth ACM international conference on Multimedia* (New York, NY, USA, 1996), ACM Press, pp. 21–30.

[22] QUINLAN, R. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA., 1993.

[23] RABINER, L. R. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceeding of the IEEE* (February 1989), vol. 77, pp. 257–286.

[24] ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review* (1958).

[25] SAUNDERS, J. Real-time discrimination of broadcast speech/music. In *ICASSP '96: Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing* (May 1996), vol. 2, pp. 993–996.

[26] SCHEIRER, E., AND SLANEY, M. Construction and evaluation of a robust multifeature speech/music discriminator. In *ICASSP '97: Proceedings of the 1997 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '97)-Volume 2* (Washington, DC, USA, 1997), IEEE Computer Society, p. 1331.

[27] SCHLIEP, A., GEORGI, B., AND RUNGSARITYOTIN, W. General hidden markov model library.

[28] SMITH, L. Aap raises concern over effects of advertising on children's health. *American Family Physician 75.1* (January 2007), 18.

[29] SRINIVASAN, S., PETKOVIC, D., AND PONCELEON, D. Towards robust features for classifying audio in the cuevideo system. In *MULTIMEDIA '99: Proceedings of the seventh ACM international conference on Multimedia (Part 1)* (New York, NY, USA, 1999), ACM Press, pp. 393–400.

[30] TDUNNING. Hiddenmarkovmodel.png. http://en.wikipedia.org/wiki/Image:-HiddenMarkovModel.png, August 2007. published under GNU Free Documentation License, Version 1.2.

[31] TENG, L., LAROCHE, M., AND ZHU, H. The effects of multiple-ads and multiple-brands on consumer attitude and purchase behavior. *Journal of Consumer Marketing* (January 2007), 27–36.

[32] WITTEN, I. H., AND FRANK, E. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2005.

A Frequency Compression Algorithm

The following algorithm transforms any input array with logarithmic spacing into another array using linear spacing. This is used to linearise frequencies. For example: If the input array contains a frequency at index n_i, the frequency at index $2n_i$ will be an octave higher, while the frequency at index $4n_i$ will be two octaves higher. The output array will then contain the same frequency at n_o, while one octave higher will be at $n_o + x$ where x is relative to the length of the output array.

```
/**
 * @param rawFreqs the array of logarithmic frequencies
 * @param compressedBand the output of linear frequencies
 * @param numRaw the size of raqFreqs
 * @param numCompressed the size of compressedBand
 */
int compressBands(double *rawFreqs, double *compressedBand,
                  int numRaw, int numCompressed) {
  if(rawFreqs == NULL || compressedBand == NULL
        || numRaw < 1 || numCompressed < 1)
    return -1;

  //Start with 1Hz frequency
  double centerFreq = 1.0;
  double lowerFreqBound, upperFreqBound;
  int lowerIndex, upperIndex;

  //Work out N, which is the number of steps per
  //octave in the compressed band
  //If numCompressed is 202 this works out to exactly 24
  //steps per octave i.e. each step is a quarter note.
  double N = (numCompressed-2)/(8.0+1.0/3.0);
  //Note that we skip the first few octaves as these are
  //in a frequency range (0-20Hz) that does not interest us
  int freqIter = (int)N*6;
  double ten_N_log = 10.0*N*log10(2.0);
  for(int i = 0; i < numCompressed; i++) {
    compressedBand[i] = 0;

    //First calculate the frequencies;
    //We start with a frequency around 20Hz
    centerFreq = pow(2.0,(3.0*(freqIter))/ten_N_log);
    freqIter++;
```

```
    lowerFreqBound = centerFreq/pow(2.0,1.0/(2.0*N));
    upperFreqBound = centerFreq*pow(2.0,1.0/(2.0*N));

    //Then convert them to array indices using the formula
    // index = Frequency*arraySize/(maxFreq)*2
    //NOTE max Freq = 22050 as we are sampling at 44100Hz
    lowerIndex = (int) (lowerFreqBound*((double)numRaw)/44100.0);
    upperIndex = (int) (upperFreqBound*((double)numRaw)/44100.0);
    if(upperIndex > numRaw)
      upperIndex = numRaw;
    if(upperIndex == lowerIndex)
      lowerIndex--;
    if(lowerIndex < 0)
      lowerIndex = 0;

    //Finally make the compressed band index be the mean of values
    //in the uncompressed band
    for(int j = lowerIndex; j < upperIndex; j++) {
      compressedBand[i] += rawFreqs[j];
    }
    if(upperIndex != lowerIndex)
      compressedBand[i] /= upperIndex - lowerIndex;
  }
}
```

B Machine Learning Classifiers

B.1 Decision Tree

This is the decision tree produced by the J48 algorithm, and implemented in C.

```
if(sumPower <= 0.656505) {
  if(variation <= 0.570545) {
    if(balance <= 0.422702) {
      if(energyDiffs <= 0.298185) {
        return 1;//: false (29.0/1.0)
      }
      if(energyDiffs > 0.298185) {
        if(energyDiffs <= 0.331309) {
          return 0;//true (6.0/1.0)
        }
        if(energyDiffs > 0.331309) {
          return 1;//: false (3.0)
        }
      }
    }
    if(balance > 0.422702) {
      if(balance <= 0.443866) {
        if(loudVals <= 0.15625) {
          return 0;//: true (6.0/1.0)
        }
        if(loudVals > 0.15625) {
          return 1;//: false (2.0)
        }
      }
      if(balance > 0.443866) {
        return 0;//: true (12.0)
      }
    }
  }
  if(variation > 0.570545) {
    return 1;//: false (63.0)
  }
}
if(sumPower > 0.656505) {
  return 0;//: true (79.0/2.0)
}
```

B.2 Markov Models

This section contains the Markov Models as produced by the Baum-Welch algorithm.
M is the number of outputs,
N is the number of states.
The A matrix contains the state transition probabilities.
The B matrix contains the output probabilities.

```
//Music Model
 HMM = {
M = 10;
N = 7;
prior = -1.000;
ModelType = 0;
A = matrix {
0.66, 0.03, 0.08, 0.01, 0.00, 0.14, 0.07;
0.22, 0.56, 0.09, 0.05, 0.00, 0.06, 0.02;
0.09, 0.24, 0.52, 0.06, 0.01, 0.07, 0.01;
0.00, 0.00, 0.17, 0.59, 0.24, 0.00, 0.01;
0.00, 0.00, 0.00, 0.33, 0.62, 0.05, 0.00;
0.00, 0.05, 0.26, 0.00, 0.07, 0.56, 0.06;
0.00, 0.02, 0.13, 0.00, 0.00, 0.29, 0.55;
};
B = matrix {
0.78, 0.16, 0.03, 0.00, 0.03, 0.01, 0.00, 0.00, 0.00, 0.00;
0.00, 0.83, 0.09, 0.07, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00;
0.00, 0.00, 0.73, 0.22, 0.05, 0.00, 0.00, 0.00, 0.00, 0.00;
0.00, 0.00, 0.01, 0.93, 0.06, 0.00, 0.00, 0.00, 0.00, 0.00;
0.00, 0.00, 0.00, 0.03, 0.96, 0.01, 0.00, 0.00, 0.00, 0.00;
0.00, 0.00, 0.01, 0.13, 0.32, 0.55, 0.00, 0.00, 0.00, 0.00;
0.00, 0.00, 0.00, 0.02, 0.01, 0.01, 0.54, 0.23, 0.11, 0.08;
};
Pi = vector {
0.11, 0.20, 0.25, 0.13, 0.14, 0.12, 0.06;
};
fix_state = vector {
0, 0, 0, 0, 0, 0, 0;
};
};
// Speech Model
HMM = {
M = 10;
N = 7;
prior = -1.000;
ModelType = 0;
A = matrix {
0.84, 0.09, 0.03, 0.03, 0.00, 0.00, 0.01;
0.18, 0.46, 0.15, 0.20, 0.00, 0.01, 0.00;
0.00, 0.47, 0.43, 0.00, 0.00, 0.00, 0.10;
0.00, 0.01, 0.15, 0.13, 0.17, 0.00, 0.54;
0.32, 0.49, 0.17, 0.00, 0.00, 0.01, 0.00;
0.05, 0.39, 0.00, 0.00, 0.45, 0.10, 0.00;
0.00, 0.00, 0.06, 0.00, 0.39, 0.55, 0.00;
```

```
};
B = matrix {
0.96, 0.04, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00;
0.16, 0.83, 0.00, 0.01, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00;
0.14, 0.14, 0.63, 0.04, 0.00, 0.07, 0.00, 0.00, 0.00, 0.00;
0.64, 0.08, 0.05, 0.12, 0.06, 0.05, 0.00, 0.00, 0.00, 0.00;
0.00, 0.05, 0.36, 0.15, 0.33, 0.03, 0.03, 0.00, 0.00, 0.04;
0.00, 0.00, 0.00, 0.03, 0.00, 0.34, 0.00, 0.25, 0.34, 0.03;
0.00, 0.11, 0.12, 0.19, 0.11, 0.07, 0.26, 0.11, 0.00, 0.02;
};
Pi = vector {
0.41, 0.18, 0.15, 0.11, 0.04, 0.05, 0.05;
};
fix_state = vector {
0, 0, 0, 0, 0, 0, 0;
};
};
```