Speech segment classification on music radio shows using machine learning algorithms

Tim Scarfe and Yuri Kalnishkan

Computer Learning Research Centre and Department of Computer Science, Royal Holloway, University of London, Egham, Surrey, TW20 0EX, United Kingdom
{tim,yura}@cs.rhul.ac.uk

Abstract. In this paper, we describe audio "texture" features based on the Short Time Fourier Transform (STFT). We use these features in combination with three popular learning machine algorithms to classify spoken voice segments of a popular Electronic Dance Music radio show "A State of Trance", which is produced by the current world number 1 DJ; *Armin van Buuren*.
The aim was accurately to distinguish when Armin van Buuren was talking regardless of background silence, music or other voices (sung or spoken).
We achieved strong empirical results which could be further improved with some basic domain-specific heuristics or compromises on the feature parameters. SVM and Bayesian Logistical Regression produced particularly encouraging results both yielding $\approx 98\%$ overall classification accuracy and $\approx 99\%$ F-score on the speech class on the highest model where we increased the verbosity of the underlying feature set. SVM however provided the most robust performance given several feature variations, significantly out-performing the others given less verbosity on the feature set.

1 Introduction

There is a wealth of time series style data such as audio that does not possess effective "temporal" metadata relevant to the purposes of information retrieval and processing. Audio is different from video because it can not be quickly scrubbed and indexed by a human in the same way a video can. One can scrub through most videos on YouTube and quickly ascertain the gist. Often, audio data must be assimilated much more slowly because the only way to interpret the content and structure is to listen in real time. The metadata itself could be the name of the current track, the genre, or simply whether the DJ was talking.

Automatic methods to add temporal metadata to audio data would be ideal and could open up the possibility of real time applications such as blocking commercials or spoken voice on a radio.

The problem of distinguishing music from speech can be approached by introducing some explicit criteria; for example, speech having distinct spectral characteristics over time. The issue though is that no single type of structural analysis would be sufficient to distinguish speech. Rather it is better to use a learning machine which will identify high level patterns amongst features in the audio. In this paper, we have applied this approach which is typical of machine learning. We create a training set by labelling some part of a music file and then train a learning algorithm to distinguish music from speech.

The design of descriptive features (structure extraction) is one of the main challenges when building pattern recognition systems. Features that describe intervals of non-stationary data are even more challenging. This paper builds on methods recently used by Tzanetakis et al [10][11] and Foote et al [4].

The learning algorithms which we use for this binary classification task are; Support Vector Machines (SVM), Bayesian Logistical Regression (BLR) and C4.5.

We were particularly interested in finding applications of supervised machine learning techniques (especially SVM) to audio segmentation and classification. Most papers were concerned with unsupervised scenarios, genre classification or fingerprinting.

Lie et al [6] did very similar work to us implementing an RBF SVM predicting 4 segment classes (3 of which were speech based). They also achieved encouraging results, 96.65% average accuracy on speech/non-speech (we got 98.3%). They cited classification performance and the complicated distribution of audio data in feature space as their motivations for using SVM.

R.Shantha [3] used an SVM with a combination of wavelet features and Fourier features to predict 16 audio classes from a database of 534 sounds. At the highest level of detail on their features, they achieved 91.6% classification accuracy.

Jia Ching-Wang [5] used an SVM in combination with spectral features to predict 15 audio classes from 677 sound files achieving an accuracy of 91.7% in the best case. He discusses a novel framed based approach to combine the constituent features from one file into one big feature, rather than taking the means and standard deviations. We do the latter in this paper although the difference is that our features only represent one second of time which is perfectly acceptable (we are doing segmentation).

We were not able to find examples of BLR or decision tree algorithms used for audio classification. However we have shown these algorithms to be quite weak compared to SVM.

2 Digital Signal Processing

Audio data starts out in the time domain and meaningful analysis cannot be performed easily in the time domain (with a notable exception in our case - the Zero Crossings feature). While a single sinusoidal function such as $\sin(x)$ is instantly recognisable; any kind of composite such as $\sin(x) + \sin(2x)$ would not look like a sine wave any more and some decomposition into the frequency domain would clearly be required before analysis. Fourier Analysis represents any function as a set of multiple integer oscillations of trigonometric functions. To use an analogy, think of a prism decomposing a magenta-coloured light into its constituent colours (red and blue). Ultimately we need rich short-time feature vectors to pass into the Support Vector Machine.

In this paper we are only interested in the *discrete* Fourier transform because the audio data has been discretely sampled (usually at $44,100Hz$).

2.1 Discrete Fourier Transform

The sequence of N complex numbers $x_0, \ldots, x_{N-1}$ is transformed into the sequence of N complex numbers $X_0, \ldots, X_{N-1}$ by the DFT according to

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn}, \quad k \in \{0, \ldots, N-1\} \tag{1}$$

where i is the imaginary unit and $e^{\frac{2\pi i}{N}}$ is a primitive Nth root of unity i.e. complex numbers on the edge of the unit circle that move around counter-clockwise when raised to the Nth power. Note that x is our audio signal which will be real-valued for our purposes.

The DFT result vector X_k can be converted back into the time domain with the Inverse Discrete Fourier Transform (IDFT) given as

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{\frac{2\pi i}{N} kn}, \quad n \in \{0, \ldots, N-1\} \tag{2}$$

.

2.2 Short Time Discrete Fourier Transform

On large audio samples such as radio shows; it is not sufficient just to use the DFT. For our purposes we are interested in short-time features in the audio data. We effectively break up the signal up into sliding, overlapping frames by multiplying x_i by window function $w(t, s, p)$ where t is the start index of the window, s is the requested position in the window and p is the window length. STFT windows are only non-zero for a short period of time (for an interval of $[1, p]$). The reason STFT frames are overlapped (see Fig. 1) is to reduce boundary artifacts. The window function itself is often hill shaped but centred around $t + \frac{p}{2}$ such as the Hann window given as

$$w(t, s, p) = \left\{ \begin{array}{ll} \frac{1-\cos\left(\frac{2\pi s - t}{p}\right)}{2} & t \in (s, s+p) \\ 0 & \text{elsewhere} \end{array} \right\} \tag{3}$$

. The main reason for this is to reduce spectral leakage.

When spectral leakage is less of a concern, the rectangular window is often used, given as

$$w(t, s, p) = \left\{ \begin{array}{ll} 1 & t \in (s, s+p) \\ 0 & \text{elsewhere} \end{array} \right\} \tag{4}$$

.

The DFT is then calculated for these windowed frames as before. The end result (known as the STFT) is given by

$$S_k = \sum_{n=0}^{N-1} w(t, s, p) x_n e^{-\frac{2\pi i}{N} kn}, \quad k \in \{0, \ldots, N-1\} \tag{5}$$

. In this paper we have applied the Hann window function to the STFT.

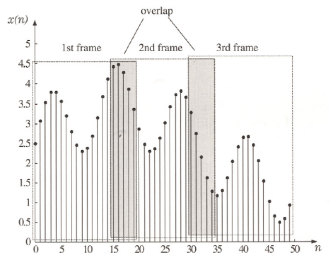

Fig. 1. Illustration of the overlapping frames in STFT (from [13])

2.3 Feature Extraction

From a pattern recognition perspective; the goal is to get a relatively small set of highly descriptive features. Clearly the learning machine (in this case SVM) would perform better in terms of performance and classification accuracy.

Discretely Binning the STFT Vectors Assuming our STFT windows are of length 512, the resulting frequency vectors will also be of length 512. This amount of information is extraneous and qualitatively too high to feed directly into an SVM considering these radio shows are often hours long. The first thing we did to the output of the STFT was to reduce the dimensionality to a much lower number (64, 32, or 8) depending on the model. We did this by dividing the resulting STFT vectors into x bins and taking the average value of each bin. In the following section we will be talking about a further discretisation into bands but this will be across these STFT bins, not the original STFT result vectors.

Downsampling In our results section we will consider 3 working empirical models. On 2 of the models (B & C) we downsample the audio information to $22050Hz$ before it even goes into the DFT. Note that this has the effect of reducing the highest representable frequency to $\frac{22050}{2}$ (Nyquist Theorem [9]). This also means that for those two models, each texture window will only comprise of about 44 analysis windows.

In addition to using the raw STFT output, we would also like to design some transformations that will succinctly distinguish the classes we are trying to identify speech vs. non-speech.

2.4 Frequency Domain Features

This section will describe the features that operate in the frequency domain. See Fig. 2 for a comparison of speech and music in the frequency domain. Most of them operate by first splitting the binned STFT frames up into log-spaced bands. If the bands parameter is set to 4, these features will have 4 outputs (e.g. Fig. 4) each effectively representing a log-spaced segment of the STFT frame which has already been binned (see Section 2.3). The rationale here is that there is more "interesting" stuff going on in the lower frequencies so we need to examine those closely while on the upper frequencies the information is less verbose and can be described with wider bands.

Let $F_i = f_i(u), u \in \{0, \ldots, M\}$ be the STFT of the ith frame. M is the index of the highest frequency bin (see Section 2.3). Let $F_{ib} = f_i(u_b), u \in (l_b, u_b)$ where l_b and u_b are edges of the band b. For clarity, $f_i(u)$ describes a band b on the ith STFT frame. The spectral features that follow are extracted for each binned STFT frame.

The Spectral Centroid given as

$$SC_{ib} = \frac{\sum_{u=l_b}^{u_b} u \cdot |f_i(u)|^2}{\sum_{u=l_b}^{u_b} |f_i(u)|^2} \tag{6}$$

is perhaps the most common spectral feature referred to in research papers and often cited as being highly effective when combined with a textural feature ([10]).

The Spectral Centroid characterises the audio spectrum's shape (see Fig. 4) and brightness - indicating its "centre of mass" i.e. central tendency. It is calculated using the weighted mean of the frequencies present in the spectrum.

The Spectral Bandwidth feature given as

$$SB_{ib} = \frac{\sum_{u=lb}^{ub} (u - SC_{ib})^2 \cdot |f_i(u)|^2}{\sum_{u=lb}^{ub} |f_i(u)|^2} \tag{7}$$

is the weighted average of the distances between spectral components.

The Spectral Band Energy feature given as

$$SBE_{ib} = \frac{\sum_{u=l_b}^{u_b} |f_i(u)|^2}{\sum_{u=0}^{M} |f_i(u)|^2} \tag{8}$$

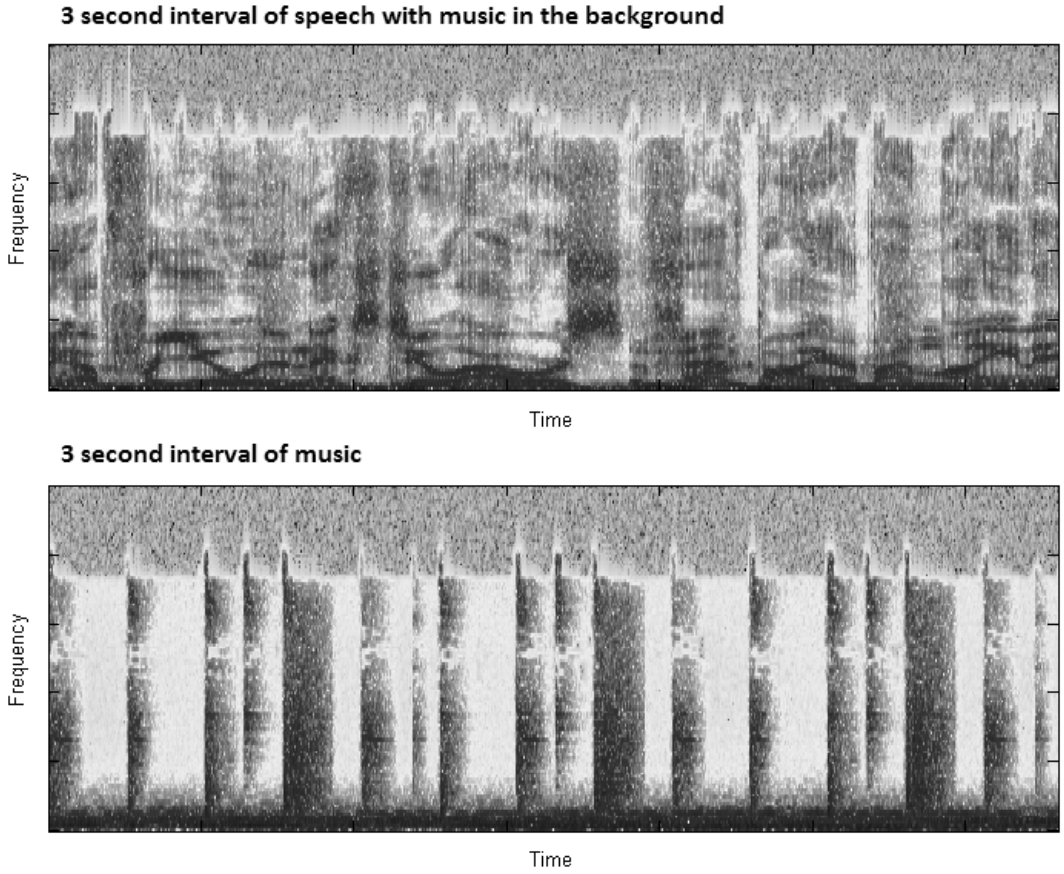

Fig. 2. An illustration of two spectrograms showing 3 second sample intervals from A State of Trance. The top one is speech on top of music, the bottom is just music. It is worth clarifying at this point that the DJ never talks without some kind of music in the background. The thing that really characterises speech is the harmonic "stripes" in the spectrogram which are frequency "peaks" changing with time. Most musical instruments produce a harmonic series of frequency peaks where all peaks are constant multiples of the first peak (which is known as the fundamental frequency). The human voice is also a musical instrument. The reason that musical instruments all produce a harmonic series is simply because it is more pleasing to listen to than a single tone because it stimulates several critical bands of hearing in our ears. The reason for the existence of harmonics is quite intuitive too, for example when a guitar string vibrates it recursively subdivides on itself causing resonance at constant multiples of its fundamental frequency. Different instruments are characterised by the shape of their harmonic series. The music segment at the bottom is showing a particularly percussive piece though (think drums and "boum-boum"). Percussive instruments are not harmonic but rather have full spectral coverage, they are not constant in time which explained the gaps. These particular percussive beats do have some visible harmonic content incorporated though because electronic dance music is created with synthesisers and thus doesn't accurately represent traditional instruments.

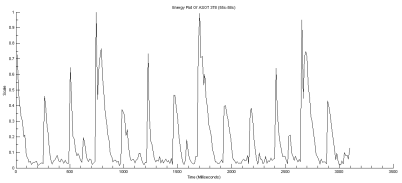

Fig. 3. Plot of the spectral energy feature (with one band shown) on a 3 second segment of show 378.

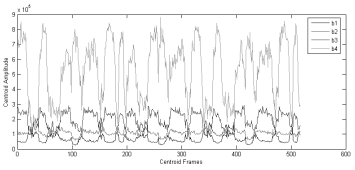

Fig. 4. Illustration of the spectral centroid feature with 4 bands on a short music segment.

describes energy in the frequency bands normalised by the energy in the entire spectrum (see Fig. 3).

The Root Mean Square feature given as

$$RMS_{ib} = \sqrt{\frac{\sum_{u=l_b}^{u_b} f_i(u)^2}{u_b - l_b}} \tag{9}$$

is a measure of the loudness of a frame ([10]).

The Spectral Flatness Measure given as

$$SFM_{ib} = \frac{\left[\prod_{u=l_b}^{u_b}\right]^{\frac{1}{u_b - l_b + 1}}}{\frac{1}{u_b - l_b + 1}\sum_{u=u_b}^{u_b} |f_i(u)|^2} \tag{10}$$

quantifies the flatness of the spectrum and distinguishes between noise and tone-like signals.

The Spectral Crest Factor feature given as

$$SCF_{ib} = \frac{\max\left(|f_i(u)|^2\right)}{\frac{1}{u_b - l_b + 1}\sum_{u=u_b}^{u_b} |f_i(u)|^2} \tag{11}$$

measures the tonality of a signal.

The Shannon Entropy feature given as

$$SE_{ib} = \sum_{u=l_b}^{u_b} |f_i(u)| \log_2 |f_i(u)| \tag{12}$$

measures the spectral distribution of a signal.

The Renyi Entropy feature

$$RE_{ib} = \frac{1}{1-r} \log\left(\sum_{u=l_b}^{u_b} |f_i(u)|^r\right) \tag{13}$$

is also a measure of spectral distribution but can be of a higher degree. In all experiments on this paper we have used degree 2.

The Spectral Rolloff feature

$$RO_{ib} = \sum_{u=l_b}^{u_b} f_i(u) \leq 0.85 \left(\sum_{u=lb}^{u_b} f_i(u)\right) \tag{14}$$

is characterised as the frequency below which 85% of the magnitude distribution is concentrated (Tzanetakis et al [11]). The rolloff is another measure of spectral shape.

Spectral Flux is the sum of the square difference between successive binned STFT frames (taken from [8])

2.5 Time Domain Features

Zero Crossings Rate The Zero Crossings Rate is a measurement of how many times the zero boundary is crossed within an interval (also 512 samples).

2.6 Textural extraction

Ultimately we want feature vectors that represent 1 second of audio. This is for simplicity primarily but also qualitatively to ensure that there is enough information to discern whether or not the DJ is talking.

We could simply set a STFT window size of 44100, but we want to have an even richer temporal picture of what is happening within that one second. To achieve this; we set the STFT window size to be 512 samples which is roughly $\frac{1}{88}$th of a second (small enough to assume stationarity of the signal) and we combine them back together into 1 second feature vectors by taking the arithmetic means and variances. This is really important to do because speech contains vowel and consonant sections which have very different spectral characteristics ([11]).

The width of the windowing function $w(t, s, p)$ determines whether there is a good frequency resolution, or good time resolution (see Fig. 2.6). Clearly, the shorter the window, the better the time resolution and thus the time of changes in frequencies can be accurately represented. A wider window though will result in much more accurate representation of what the frequencies actually are. This can be explained using the Nyquist theorem. Assuming we take a window of N samples at a sampling rate of S. The underlying DFT will produce N complex coefficients. Half of these get discarded on a real valued signal and the remaining $\frac{N}{2}$ coefficients represent the discrete frequencies $\{\frac{\frac{S}{2}}{\frac{N}{2}} \cdot \{0, \ldots, \frac{N}{2}\}\}$. Clearly, as N increases, so does the frequency resolution. There are other methods of signal analysis such as Wavelets that do not have the time/frequency tradeoff because they operate in a different domain (time/scale domain). For our purposes though we felt that the STFT was perfectly acceptable because for example at a sampling rate of 44100Hz, a 2048 sample window might give us $\frac{44100}{2048} \simeq 22.5$ discrete frames per second and 1024 frequency blocks representing increments of $\frac{22050}{1024} \simeq 21.5$Hz. This is actually overkill which is why we bin the STFT vectors to further reduce their dimensionality.

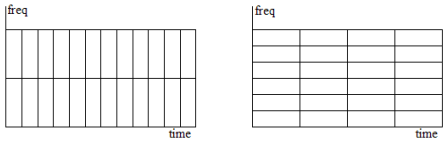

Fig. 5. Illustration of the time/frequency resolution trade off.

We call these *textural features* inspired from Tzanetakis et al 2002 [11]. Tzanetakis found that musical genre classification accuracy rapidly increases with the number of analysis frames within

texture windows. From 40% at 1 frame to 55% at 20 frames. It finds a limit very quickly after about 500ms of represented underlying audio. This is intuitive considering the limit is reached at the point when the genre could be reasonably ascertained.

Other alternative methods have been used to incorporate the temporal information into the stationary STFT outputs such as time derivatives [14].

3 Learning Machines Overview

3.1 Support Vector Machines

Support Vector Machines are a set of supervised learning methods used for binary classification (pattern recognition). SVM works by placing the training data in a multidimensional real value space and aims to find an optimal linear separating hyperplane that maximises the margin between the nearest examples of the two opposite classes while simultaneously minimising the extent of error (examples falling on the wrong side of the hyperplane). The induced hyperplane becomes a binary decision rule for classifying new examples. One of the reasons SVM performs so well is that its objective function has a "regularising" term with the combined objectives of simplifying the decision rule as well as minimising error on the training set. This allows SVM to deal well with "overfitting" which is where the training set is learned too tightly thus jeopardising general performance.

Primary form of SVM We have a training set of examples with their respective labels of the form (x, y) and we want to separate them with a hyperplane of the form $x_i \cdot w + b$. In the primary form the SVM can be described as the following optimisation problem

$$\begin{aligned} &\frac{1}{2}(w \cdot w) + C\left(\sum_{i=1}^{l} \xi_i\right) \to \min \\ &\text{s.t.} \quad y_i((x_i \cdot w) + b) \geq 1 - \xi_i \\ &\qquad\qquad i = 1, \ldots, l \end{aligned} \tag{15}$$

.

The primary form gives us the optimal separating hyperplane w_i. ξ_i is the cumulative error, C is the "complexity" coefficient, which weights the amount of error that will be tolerated (and thus how simple the resulting decision rule is).

Classifying non-linear data Perhaps the most powerful feature of SVM is the implementation of the "Kernel Trick". SVM is a linear classifier and in its original form can only perform well on linearly separable data. If you had a set of 2-dimensional data which looked like a cluster of negative examples surrounded by a circle of positive examples, the SVM would fail to produce good results. However the "trick" is that we first transform the data using a higher degree function and then find a linear separating hyperplane in the transformed space. Clearly; the resulting decision rule would need to incorporate the same transformation within it so that new examples are first transformed into the same space before the classification is made.

The kernel trick transforms any algorithm that solely depends on the dot product between two vectors (which is replaced with the kernel function). The SVM can be described in dual formulation to allow us to use kernels. Using kernels, a linear algorithm can be transformed into a non-linear algorithm. This non-linear algorithm is equivalent to the linear algorithm operating

in the range space of ϕ. However, because kernels are used, the ϕ function is never explicitly computed (where $\phi(x)$ is an inner product space, such that $K(x, y) = \varphi(x) \cdot \varphi(y)$). This is desirable, because the high-dimensional space may be infinite-dimensional (as is the case when the kernel is a Gaussian).

The kernel trick was first published by Aizerman et al [7].

The dual form is found from representing the SVM problem as a Lagrangian. When the Lagrangian is rearranged to be only in terms of the Lagrangian multipliers α_i, the training examples are also in an inner product. The inner product is then replaced with a kernel function

$\sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j K(x_i, x_j) \to \max$

such that $0 \leq \alpha_i \leq C, \quad i = 1, 2, \ldots, l$ and $\sum_{i=1}^{l} y_i \alpha_i = 0$.

The classification of a new x is given by $\operatorname{sgn}\left(\sum_{i=1}^{l} \alpha_i y_i K(x_i, x) + w_0\right)$.

Radial Basis Function Kernel We will be using one of the most popular kernel functions – the Gaussian Radial Basis Function Kernel

$$k(x, x') = e^{\left(-\gamma \|x - x'\|^2\right)}$$

where $k(x, x')$ describes a cell in the Kernel matrix and γ is the width of the Gaussian function. We tried several kernels and got the best results from this one.

3.2 Bayesian Logistical Regression

Suppose we are given a data sample $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_T, y_T)\}$, where $x_i \in \mathbb{R}^d$ are signals and $y_i \in \{-1, 1\}$ are labels. Consider the set of probabilistic models parametrised by $\beta \in \mathbb{R}^d$, where $\Pr(y = 1 \mid x, \beta) = \psi(\beta' x)$ and $\psi(r) = e^r/(1 + e^r)$, or, in other terms, $\Pr(y = s \mid x, \beta) = (1 + e^{-s\beta' x})^{-1}$, $s = \pm 1$, and labels are assumed to be independent given their signals.

The Bayesian approach is used to fit a model, i.e., to find β. One needs to minimise the negative logarithm of the posterior density of β given D, which is equal to $\sum_{i=1}^{T} \ln(1 + e^{-y_i \beta' x_i}) - \ln p(\beta)$, where $p(\beta)$ is the prior density. For the prior density we take either the Gaussian (in this paper we always used the Gaussian prior) or the Laplace distribution centred at 0 with independent components, so that $-\ln p(\beta)$ equals, up to a constant, to $\sum_{j=1}^{d} \beta_j / \sigma_j^2$ or $\sum_{j=1}^{d} \lambda_j |\beta_j|$. See Genkin et al[2] for more detail.

Once a β is found, we can work out classifications for new unseen signals by setting a threshold.

3.3 C4.5

C4.5 is an algorithm used to generate a decision tree developed by Ross Quinlan [12]. C4.5 is an extension of Quinlan's earlier ID3 algorithm.

At each node of the tree, C4.5 chooses one attribute of the data that most effectively splits its set of samples into subsets enriched in one class or the other. Its criterion is the normalized information gain (difference in entropy) that results from choosing an attribute for splitting the data. The attribute with the highest normalized information gain is chosen to make the decision. The C4.5 algorithm then recurs on the smaller sublists.

The algorithm is based on Occam's razor i.e. it prefers smaller decision trees (simple rules) over larger ones. However, it does not always produce the smallest tree, and is therefore a heuristic. Occam's razor is formalised using the concept of information entropy

$$E(S) = -\sum_{j=1}^{n} f_S(j) \log_2 f_S(j)$$

where $E(S)$ is the information entropy of the subset S; n is the number of different values of the attribute in S (entropy is computed for one chosen attribute), $f_S(j)$ is the frequency (proportion) of the value j in the subset S.

Gain given as

$$G(S, A) = E(S) - \sum_{i=1}^{m} f_S(A_i) E(S_{A_i})$$

quantifies the entropy improvement by splitting over an attribute, and higher is better where $G(S, A)$ is the gain of the subset S after a split over the A attribute, $E(S)$ is the information entropy of the subset S, m is the number of different values of the attribute A in S, $f_S(A_i)$ is the frequency (proportion) of the items possessing A_i as value for A in S, A_i is i^{th} possible value of A, S_{A_i} is a subset of S containing all items where the value of A is A_i.

C4.5 incorporates one relevant improvement for our purposes and that is pruning trees after creation. C4.5 goes back through the tree once it has been created and attempts to remove branches that do not help by replacing them with leaf nodes.

4 Experimentation Methodology and Final Feature Vectors

Episodes of a State of Trance are normally 2 hours long. For experimentation we decided to extract nearly 5 minute (299 seconds) intervals from 9 different shows. Usually these intervals were from the beginning of the show because the DJ is almost guaranteed to talk although sometimes they were taken from a random index in the show where the DJ talked. We used one of these 5 minute intervals as training data, and the rest were concatenated together to form a large single test set.

As with any binary classification task, true/false labels were assigned to all of the data – intervals featuring DJ Armin van Buuren talking were labelled true, other regions were labelled false (even if they contained speech or singing from other people). The purpose was to specifically distinguish when Armin van Buuren was talking.

All of the labelled data and corresponding MP3 files are available to download from the address given at the end of the paper (in CSV and Weka format). Show 374 was used for the training set, and shows; 368, 369, 370, 371, 372, 373, 375 and 378 (in that order, concatenated into one) were used for the test set.

The training set had 28 instances/seconds of speech (271 non-speech). The test set had 2392 instances, comprising; 291 speech, 2101 non-speech.

Feature extraction summary STFT was performed on overlapping sliding frames of discretised sampled audio (as previously discussed in this paper). The source audio was stereo but we simply took the left channel and discarded the right. The STFT frames were binned to reduce the dimensionality and various frequency domain features were then run with these bins as an input. One feature (zero crossings) was run in the time domain i.e. no STFT is applied.

Textural features corresponding to one second of source audio were created by taking means and variances of the underlying raw STFT bins and output of feature vectors. The interval of the means and variances would of course alter depending on the STFT capture window i.e. a capture window of 512 samples would mean that $\frac{\text{samplerate}}{512}$ features would be combined into one. The cardinality of the feature vector doubles at this stage because for every "textural" feature there is a mean and variance for an interval of underlying "normal" features.

See Table 2 for an illustration of how the features are set out for use with the learning machines. Clearly the size of the feature matrix will change depending on the parameters used on the feature extractors i.e. number of bands and the number of bins on the STFT.

Data preprocessing For the SVM and BLR algorithms the data was first linearly scaled onto the interval [0,1]. These algorithms have been shown to perform better when the data is scaled to this interval.

Selecting optimal parameters for the learning machines Selecting optimal parameters for learning machines can be a bit of a black art. The approach we used was essentially trial and error using the Weka data mining software ([1]). We were interested in optimising for the F-measure on the speech class first, while bearing the overall classification accuracy in mind. These objectives were never conflicting.

Please see Table 1 for an illustration of which parameters we used on the learning machines, for each model.

	Model A	Model B	Model C
SVM Complexity (C)	5	5	5
SVM RBF Bandwidth (γ)	0.07	0.08	0.09
C4.5 Tolerance	0.25	0.25	0.25
BLR Tolerance	0.8	0.8	0.5

Table 1. Table showing which learning machine parameters were used in each model.

Mean	Zero Crossings x 1
Mean	STFT x Bins
Mean	Spectral Flux x Bands
Mean	RMS x Bands
Mean	Rolloff x Bands
Mean	Centroid x Bands
Mean	Renyi Entropy x Bands
Mean	Shannon Entropy x Bands
Mean	Spectral Crest Factor x Bands
Mean	Spectral Band Flatness x Bands
Mean	Spectral Band Energy x Bands
Mean	Spectral Bandwidth x Bands
Variance	Zero Crossings x 1
Variance	STFT x Bins
Variance	Spectral Flux x Bands
Variance	RMS x Bands
Variance	Rolloff x Bands
Variance	Centroid x Bands
Variance	Renyi Entropy x Bands
Variance	Shannon Entropy x Bands
Variance	Spectral Crest Factor x Bands
Variance	Spectral Band Flatness x Bands
Variance	Spectral Band Energy x Bands
Variance	Spectral Bandwidth x Bands

Table 2. Illustration of how the features are laid out for input into the learning machine.

See Fig. 6 for a visualisation of the feature matrix used for Model A (models are explained in the results section).

5 Model parameters explanation

- **STFT Bins** is the number of bins discretised from the STFT vectors as described in Section 2.3.
- **Number of Bands** the number of log-spaced discretised bands rendered from the binned STFT frames.
- **Effective Sample Rate** is the effective sample rate post-downsampling (originally all samples were sampled at 44100Hz).
- **STFT Window Length** is the length of the STFT window. Assuming the sample rate was 44100Hz, 22050Hz samples would represent a $\frac{1}{2}$ second window.
- **STFT Window Overlap Factor** is how much the STFT windows overlap, expressed as a factor. 1.3 would represent a 30% overlap.

	Model A	Model B	Model C
STFT Bins	64	32	8
Number of Bands	6	4	4
Effective Sample Rate	44100	22050	22050
STFT Window Length	2048	1024	512
STFT Window Overlap Factor	1.1	1.5	1.3

Table 3. Table showing which feature parameters were used in each model.

6 Results

A high level perspective of the results is illustrated by Table 4. Refer to Table 5 for a verbose set of results for all three models.

	Model A	Model B	Model C
C4.5	82.4%	70.6%	75.3%
BLR	91.9%	70.2%	75.1%
SVM RBF	**92.6**%	**90.0**%	**80.4**%

Table 4. Table showing high level results on all three models. The results shown here are the F-measure on the speech class only (since we are most concerned with the performance of speech classification). The F-measure is a measure of accuracy taking the harmonic mean of the precision and recall of the speech class $F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$.

SVM with the RBF kernel consistently outperformed C4.5 and BLR. The SVM performance between the models also seemed to emulate the change in the feature parameters i.e. a small loss of information produced a small loss in classification accuracy. C4.5 and BLR had a pronounced drop-off in accuracy going from model A to B. However C4.5 and BLR performed slightly better

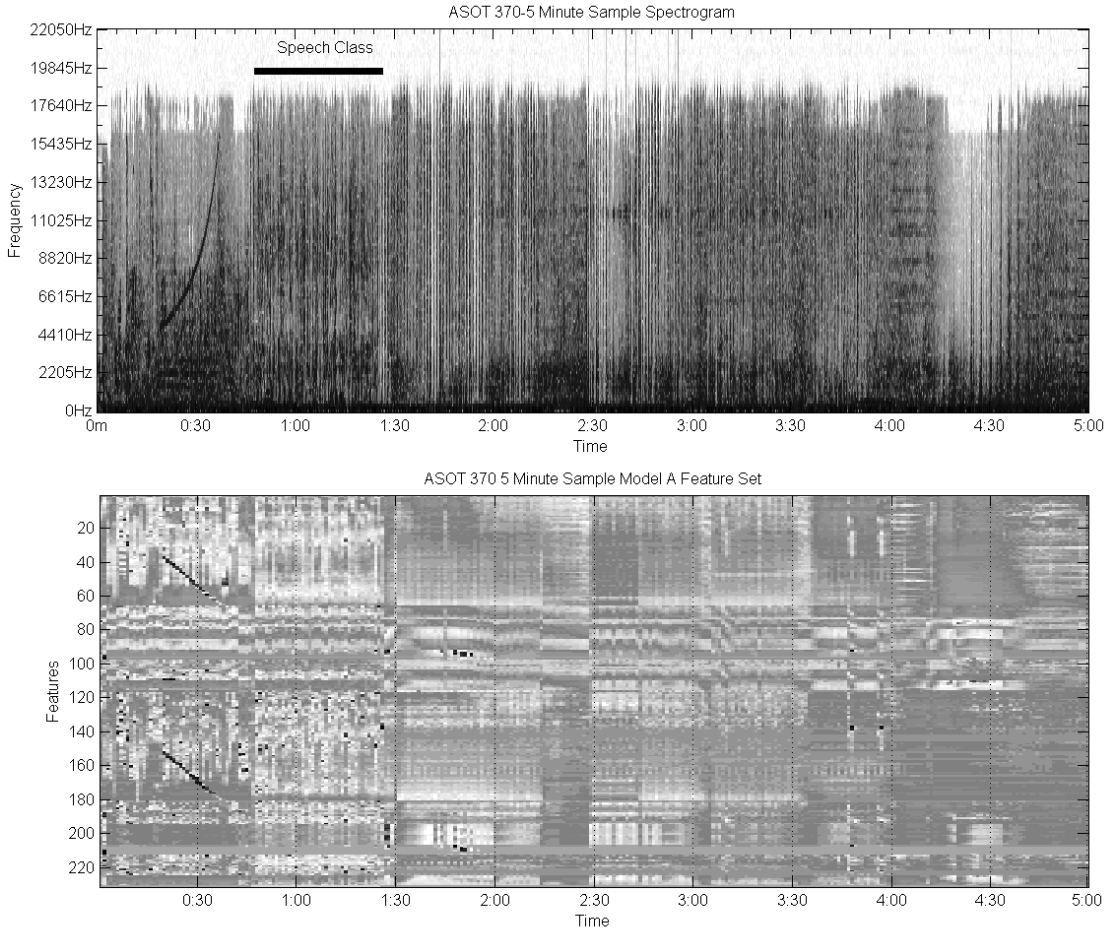

Fig. 6. Visualization of the one of the test show intervals (5 minutes in length). A Spectrogram is shown on the top with the speech class highlighted. Below is the corresponding "textural" feature matrix (Model A with 220 attributes). Imagine a horizontal dividing line going halfway across the feature matrix. The top one is the means, the bottom the variances. This is why it looks slightly like a copy of the top half. The speech class is from about 53 to 90 seconds. One interesting observation is the mysterious exponential-looking black curve on the spectrogram. This corresponds to a "whisping" sound on the radio show introduction. After the log-spaced banding on the STFT features which come first in the feature matrix, it looks more like a straight line.

	RBF-A	C4.5-A	BLR-A	RBF-B	C4.5-B	BLR-B	RBF-C	C4.5-C	BLR-C
Correctly Classified	**2351 (98.3%)**	2301 (96.2%)	2347 (98.1%)	**2338 (97.7%)**	2244 (93.8%)	2259 (94.4%)	**2288 (95.6%)**	2247 (93.9%)	2257 (94.3%)
Incorrectly Classified	41 (1.7%)	91 (3.8%)	45 (1.9%)	54 (2.2%)	148 (6.2%)	133 (5.6%)	104 (4.3%)	145 (6.0%)	135 (5.6%)
Kappa statistic	0.9165	0.8207	0.9086	0.8869	0.6723	0.674	0.7796	0.6809	0.7197
Mean absolute err.	0.0171	0.0411	0.0188	0.0226	0.0678	0.0556	0.0435	0.0653	0.0564
RMS err.	0.1309	0.1947	0.1372	0.1503	0.2428	0.2358	0.2085	0.2356	0.2376
Relative absolute err.	8.8%	21.1%	9.7%	11.8%	35.5%	29.1%	22.3%	32.1%	29%
Root relative sq. err.	39.9%	59.4%	41.8%	46.6%	75.3%	73.1%	63.6%	70.0%	72.5%
Total Instances	2392	2392	2392	2392	2392	2392	2392	2392	2392
Speech Class									
TP Rate	0.997	0.98	0.995	0.993	0.979	0.996	0.988	0.973	0.977
FP Rate	0.117	0.165	0.12	0.139	0.367	0.441	0.268	0.278	0.299
Precision	0.984	0.977	0.984	0.982	0.953	0.944	0.964	0.962	0.959
Recall	0.997	0.98	0.995	0.993	0.979	0.996	0.988	0.973	0.977
F-Measure	**0.99**	0.978	0.989	**0.987**	0.965	0.969	**0.976**	0.967	0.968
ROC Area	0.94	0.908	0.937	0.927	0.861	0.777	0.86	0.879	0.839
Non-Speech Class									
TP Rate	0.883	0.835	0.88	0.861	0.633	0.559	0.732	0.722	0.701
FP Rate	0.003	0.02	0.005	0.007	0.021	0.004	0.012	0.027	0.023
Precision	0.973	0.85	0.962	0.942	0.798	0.946	0.891	0.787	0.81
Recall	0.883	0.835	0.88	0.861	0.633	0.559	0.732	0.722	0.701
F-Measure	0.926	0.842	0.919	0.9	0.706	0.702	0.804	0.753	0.751
ROC Area	0.94	0.908	0.937	0.927	0.861	0.777	0.86	0.879	0.839
Weighted Average									
TP Rate	0.983	0.962	0.981	0.977	0.938	0.944	0.957	0.942	0.944
FP Rate	0.103	0.147	0.106	0.123	0.326	0.39	0.237	0.248	0.265
Precision	0.983	0.962	0.981	0.977	0.934	0.944	0.955	0.941	0.941
Recall	0.983	0.962	0.981	0.977	0.938	0.944	0.957	0.942	0.944
F-Measure	0.982	0.962	0.981	0.977	0.935	0.938	0.955	0.941	0.942
ROC Area	0.94	0.908	0.937	0.927	0.861	0.777	0.86	0.879	0.839

Table 5. Verbose results for all three models, on all three learning machines.

in model C than B. The reason for this is unclear, it may be that smaller feature vectors translated to stronger classification accuracy with these learning machines.

It was of interest to us to show that relatively strong results could still be obtained with significantly less information (Models B & C). The differences in strength between the learning machines are clearly illustrated here with SVM winning by some margin over the others.

Many of the classification errors were border cases (at the beginning or the end of a speech class but most commonly the beginning) and we believe these could be avoided by using two-second feature vectors or simply left-padding the predictions as a heuristic. It would also be possible to increase the accuracy further by adding in some basic domain specific heuristics such as assuming the speech was not broken within a strong segment of speech.

The data we used and the raw results can be downloaded from http://www.developer-x.com/papers/asot/

References

1. 2007. Weka (Data Mining Tool) http://www.cs.waikato.ac.nz/ml/weka/.
2. David D. Lewis Alexander Genkin and David Madigan. Large-scale bayesian logistical regression for text catergorization. 2004.
3. R.Shantha Selva Kumari et al. Audio signal classification based on optimal wavelet and support vector machine. 2007.
4. J. Foote. An overview of audio information retrieval. *ACM Multimedia Systems, vol. 7, pp. 2-10*, 1999.
5. Cai-Bei Lin Kun-Ting Jian Jia-Ching Wang, Jhing-Fa Wang and Wai-He Kuok. Content-based audio classification using support vector machines and independent component analysis. 2007.
6. Li Lie Lu, Stan Z and China Hong-Jiang Zhang, Microsoft Research. Content based audio segmentation using support vector machines. 2001.
7. E. Braverman M. Aizerman and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning". automation and remote control 25: 821837. 1964.
8. A. Rizzi M. Buccino. Optimal short-time features for music/speech classification or compressed audio data. *IEEE*, 2006.
9. H. Nyquist. Certain topics in telegraph transmission theory vol. 47, pp. 617-644. 1928.
10. George Tzanetakis Perry Cook. Sound analysis using mpeg compressed audio. *IEEE*, 2000.
11. George Tzanetakis Perry Cook. Musical genre classification of audio signals. *IEEE*, 2002.
12. J. R. Quinlan. *C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers.* 1993.
13. Konstantinos Koutroumbas Sergios Theodoridis. *Pattern Recognition, Third Edition.* Academic Press;, 2006.
14. Arun Ramalingham Sridhar Krishnan. Gaussian mixture models using short time fourier transform features for audio fingerprinting. *IEEE*, 2005.