An Investigation into the use of Low-Level and Mel Cepstral Frequency Features for Audio Signal Classification

Munsanje Mweene (MWNMUN001)
Claude Betz (BTZCLA001)
University of Cape Town
EEE4114F

23 April 2018

***Abstract* — Audio signal classification (ASC) is a two-step process of extracting descriptive features from a sound sample and feeding them into a classification algorithm. The specific approaches applied depend on the audio categories being examined. For example, Mel-frequency cepstral features have been found to be useful for speech classification. This work limits itself to classification of audio samples as either speech or music. Two categories of audio features are considered: low-level signal features, such as the signal RMS value, and the percentage of low energy frames, and Mel-frequency Cepstral features. In terms of classification, both unsupervised and supervised approaches are considered. It is found that, using these features, unsupervised approaches grossly underperform supervised approaches. It is hypothesised that this is due to the non-linearity of the relationship between the speech and music classes in the feature space. To examine the effect of combining the low-level features and the MFCC features, a Support Vector Machine is fitted to the feature sets individually, as well as the feature sets combined. The model fitted on the combined features yielded the best performance.**

1 Introduction

Audio classification is a trivial task to the human ear. It can distinguish between a track playing through the radio, a knock on the door, or a conversation in the next room. With the increasing push for fully automated systems, it is desirable to build systems that can accurately perform these classifications without human supervision. Applications are numerous, ranging from the diagnosis of respiratory disease [1] to voice activated personal assistants. This work is concerned with building an automatic speech-music discrimination system that takes as input a raw audio signal and outputs a classification. The raw signal is first reduced into a short vector of features before feeding it into the classifier. This process is called *feature extraction*, and significantly impacts the quality of the classifier produced. In this work, two primary classes of features are used: low-level signal features and Mel-frequency Cepstral features. These are chosen due to their simplicity of implementation. The separability of the audio classes in the feature space is assessed by applying the K-means clustering algorithm and assessing its performance. Given reasonable performance, the K-means centroids can then be used to classify new samples. Finally, a classifier is built by training a Support Vector Machine on the data. An overview of the system is shown in Figure 1.

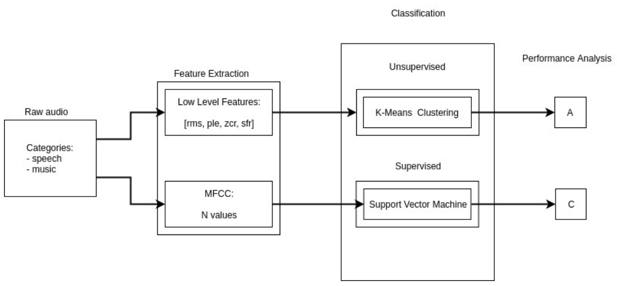

Figure 1: block diagram of system implementation

2 Literature Review

There has been much research conducted in the field of Audio Signal Classification. A common problem in this field is the development of speech-music classifiers, where the objective is to classify an audio signal as representing speech or music. The question can be broken down further to varying levels of granularity, such as identifying the origin of the speaker, the language being spoken, the genre of music and even word transcription. Breebaart et al [3] look at the classification of popular music, classical music, speech, noise and crowd noise. In addition to low-level signal parameters and MFCC features, they consider perceptual features such as roughness and loudness, as well as temporal envelopes that attempt to model processing in the human auditory system. They partition their data into a training and test set and fit a Gaussian mixture model to the training set. To evaluate the model, they assign each sample in the test set to the class that maximizes its class membership likelihood via Bayes theorem. Scheirer [2] similarly looks at low-level signal parameters to build a speech-music discriminator. They consider multiple classification models, including a Gaussian maximum a posteriori estimator, a Gaussian mixture model, and a nearest neighbor classifier. In terms of performance, they report an aggregate classification error of 6.8% when using all features.

3 Apparatus

3.1 Programming Tools

Feature extraction and classification was conducted using the Python programming language. In particular, the **NumPy** and **SciPy** libraries were used for data manipulation and the implementation of signal processing operations. The **Matplotlib** library was used for visualisation.

3.2 Data

The data used in this research consists of 2 audio categories: speech, music. Each audio snippet is a single-channel 16-bit track sampled at 22050 Hertz. The audio snippets are 5 seconds in length. There are 384 speech samples and 384 music samples to make a total of 768 audio samples. The data was collected from the open source project, Marsyas, maintained by students and researchers worldwide [8].

4 Feature extraction

Audio signal classification requires the selection of features that adequately separate the classes of interest in the feature space. The exact nature of the separation is highly dependent on the classification algorithm used. For linear algorithms, the classes must be linearly separable in the feature space for good performance, while for non-linear algorithms they may have more complex relationships [9]. For the extraction of features, it is necessary to break a signal down into small segments, called analysis windows, that are small enough to preserve stable frequency characteristics of the spectrum. In order to capture long term nature of sound, termed "texture", a longer window is defined.

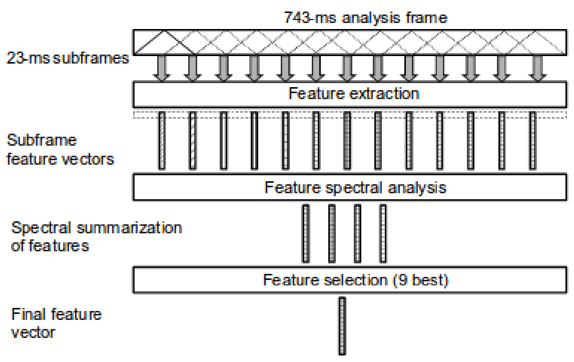

Figure 2: Feature extraction [3]

Drawing upon methodology outlined by [3], this work uses a 25ms analysis window and a 1 second texture window for feature extraction of sounds of length 5 seconds.

The approach of division of features into Low-Level and MFCC feature vectors and the subsequent combination for a multivariate classifier is outlined in [2].

4.1 Low-level features

The low-level features decided upon were Root Mean Square value, Percentage of Low-Energy Frames, Zero-Crossing Rate and Spectral Roll-Off, these were arranged into a feature vector of size 4, which was then used in the classification of the audio signal. Detailed below are the feature definitions, and implementations used:

4.1.1 Root Mean Square

rms is the Root Mean Square (RMS) value of a signal. For a digitised signal, you can calculate it by squaring each value, finding the arithmetic mean of those squared values, and taking the square root of the result.
Loosely speaking, it represents the average "power" of a signal. [2]

$$rms_n = \sqrt{\frac{1}{N}\sum_{i=1}^{N}|x[i]|^2} \tag{1}$$

where:

n = analysis window in signal
N = number of frames in analysis window
$x[i]$ = value of signal in frame i of analysis window

```python
def rms(raw_audio, analysis_window=441, texture_window=50, overlap=0.25):
    """return the variance of rms values averaged over texture widows """
    # step1: calculate rms values analysis windows
    rms_values = [rms_window(raw_audio[i:i+analysis_window]) for i in range(0,len(raw_audio),round((1-overlap)*analysis_window))]

    # step2: local variance of rms in texture window
    var_values = [lc_variance(rms_values[i:i+texture_window]) for i in range(0,len(rms_values),texture_window)]

    # step3: return mean of texture window rms
    return np.mean(var_values) #feature

def rms_window(window):
    """return the rms value of a window"""
    return np.sqrt(1/len(window)*np.sum(window*window))

def lc_variance(l_window):
    """return local variance of texture window"""
    return np.var(l_window)

```

4.1.2 Percentage of Low-Energy Frames

The percentage low energy frames determines how many frames in a sound sample fall below a certain energy threshold.

To calculate the percentage of low energy frames in the signal, the rms value defined in 1 is compared to a threshold that is used to distinguish high energy frames from low energy frames over all frames within the overall signal. Rather than have a fixed threshold, it is better to use a trailing moving average, to account for the dynamics of the signal. This was chosen to be the length of the texture window at second.

```python
def ple(raw_audio, analysis_window=441, texture_window=50, overlap=0.25):
    """return percentage low energy frames of a signal"""
    # step1: calculate analysis window rms values
    rms_values = [rms_window(raw_audio[i:i+analysis_window]) for i in range(0,len(raw_audio),round((1-overlap)*analysis_window))]

    # step2: threshold = 0.5 x average rms value
    of the last second
    mean_values = [past_mean(rms_values[i-
    texture_window:i]) for i in range(
    texture_window,len(rms_values))]

    # step3: filter rms according to past mean
    rms_adjusted = rms_values[texture_window:]
    ple_frames = [le_filter(rms_adjusted[i],
    mean_values[i]) for i in range(0,len(
    mean_values))]
    nframes = list(ple_frames).count(1)/float(len(
    ple_frames))

    return nframes #feature

def rms_window(window):
    """return rms value of analysis window"""
    return np.sqrt(1/len(window)*np.sum(window*
    window))

def past_mean(mean_window):
    """return past mean for an rms value i"""
    return np.mean(mean_window)

def le_filter(rms_curr, past_mean):
    """filter low energy frames"""
    return 1 if(rms_curr < past_mean) else 0

```

The two plots below are the characteristic plots of showing how zero-crossing rate is evaluated and a histogram showing the relevant distribution for a typical speech signal.

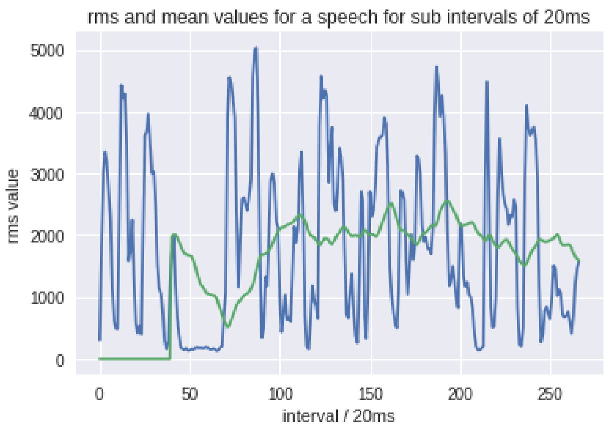

Figure 3: rms and trailing mean values for ple derivation

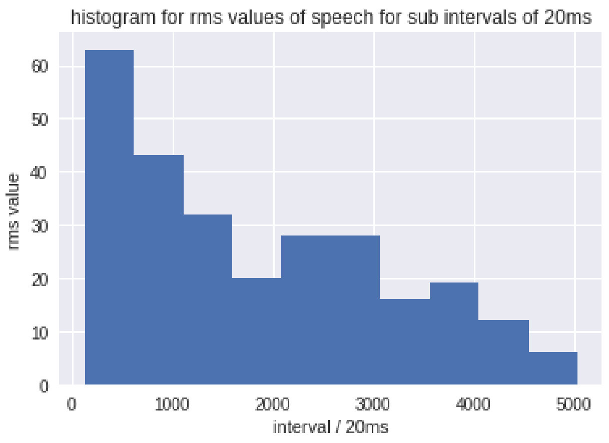

Figure 4: histogram and distribution rms values

The two plots below are the characteristic plots of showing how zero-crossing rate is evaluated and a histogram showing the relevant distribution for a typical music signal.

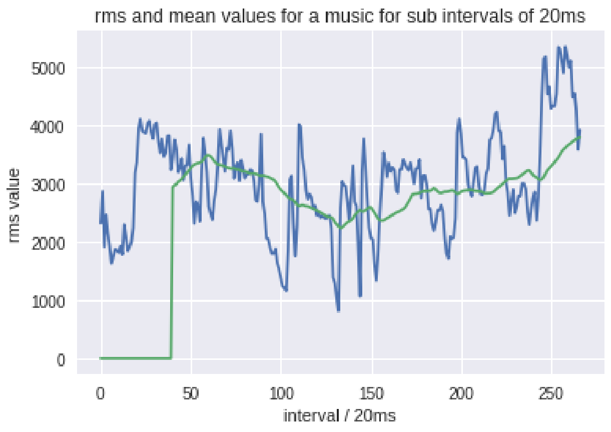

Figure 5: rms and trailing mean values for ple derivation

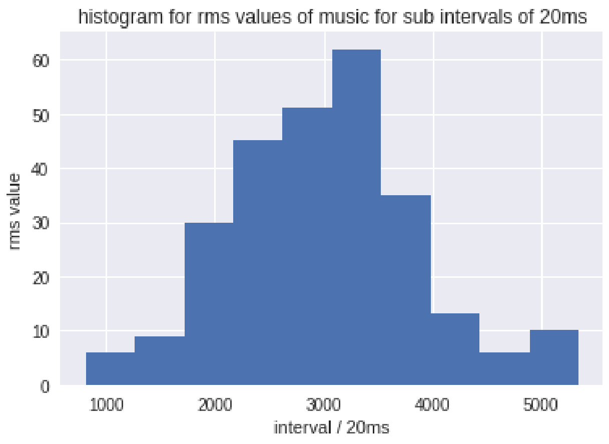

Figure 6: histogram and distribution rms values

The difference in distribution of the rms value for speech and music is a good indicator of the use of rms value as a classification feature.

4.1.3 Zero-Crossing Rate

Zero-crossing rate is a measure of number of times in a given time interval/frame that the amplitude of the speech signals passes through a value of zero. The zero-crossing rate was calculated for each 20ms analysis window (441 frames/analysis window using the 22050Hz sampling rate) of the sample signal. local variance of zero crossing rate was then calculated for each texture window (50 analysis windows/texture window). The mean of the local variance over the various texture windows in the sample was then used as sample's zero-crossing rate feature.

$$zcr_n = \frac{1}{N-1}\sum_{i=0}^{N-1} |sgn[x[i]] - sgn[x[i-1]]| \qquad (2)$$

where:

$$sgn[x[i]] = \begin{cases} 1, & x[i] \geq 0 \\ -1, & x[i] < 0 \end{cases}$$

n = analysis window in signal
N = number of analysis windows
i = frame within analysis window

```python
def zcr(raw_audio, analysis_window=551, texture_window=40, overlap=0.25):
    """return zero crossing rate for sound"""
    # step1: calculate zcr
    zcr_values = [zcr_window(raw_audio[i:i+analysis_window]) for i in range(0,len(raw_audio),round((1-overlap)*analysis_window))]

    # step2: calculate local variance of zcr in say 1 second of data i.e 50 windows
    var_values = [zcr_variance(zcr_values[i:i+texture_window]) for i in range(0,len(zcr_values),texture_window)]

    # step3: return mean of local variances as feature
    return np.var(var_values)

def zcr_window(window):
    """calculates the zero crossing rate for a window"""
    length = len(window)
    window1 = window[1:length] # n
    window2 = window[0:length-1] # (n-1)
    return (1/(length-1))*np.sum(np.absolute(np.sign(window1)-np.sign(window2))) #zcr

def zcr_variance(l_window):
    """function to return local variance over a larger window"""
    return np.mean(l_window)

```

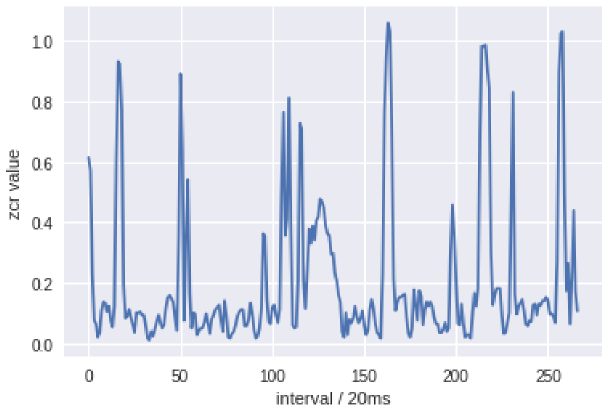

Figure 7: zcr values of a typical speech signal

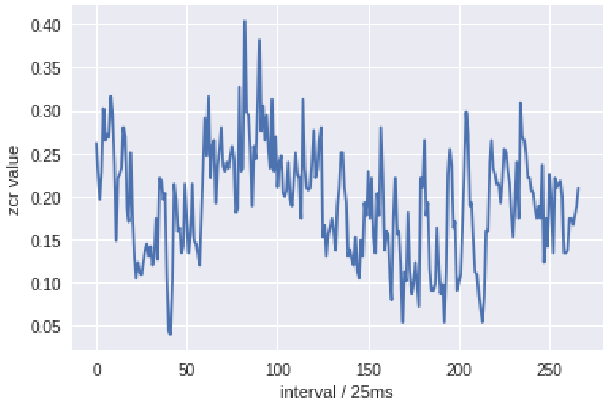

Figure 8: zcr values of a typical music signal

4.1.4 Spectral Roll-Off

This is a measure of the amount of the right-skewedness of the power spectrum. The spectral roll-off is the frequency below which 85 percent of the magnitude distribution is concentrated. The spectral roll-off was computed for the various 25ms analysis windows, variance of the average roll-off across different texture windows in the signal provided was used as the feature. [3]

$$\sum_{i=0}^{\omega-1} M_t[i] = 0.85 \sum_{i=0}^{N-1} M_t[i] \tag{3}$$

where:

ω = spectral roll-off point for a texture window
N = number of analysis windows in texture window
$M_t[i]$ = Power Spectral Density at frequency i

```python
def srf(raw_audio, fs=22050, analysis_window=441, texture_window=22050, overlap=0.25):
    """return var of calculate spectral roll-off frequency"""
    #step 1: calculate srf for texture windows
    omega = [texture_srf(raw_audio[i:i+texture_window],fs,analysis_window) for i in range(0,len(raw_audio),texture_window)]

    #step 2: return variance of srf values over signal
    return np.var(omega)

def texture_srf(window,fs,analysis_window):
    """return srf for window"""
    window = np.asarray(window)
    f, pxx = welch(window, fs=fs, window='hamming', nperseg=analysis_window, noverlap=None)
    total_area = np.sum(pxx)
    thresh = 0.85*total_area
    cum = 0.0
    for i in range(0,len(f)):
        if(cum < thresh):
            cum += pxx[i]
        else:
            break
    return f[i]

```

The figures below show plots for Power Spectral density of 1 second texture windows with overlap 0.25 over typical 5 second sample sounds for the categories of speech and unvoiced music. The dotted lines show the various values for spectral roll-off for the different texture windows observed for each sound. The variance between these values was used as a feature for the classifier.

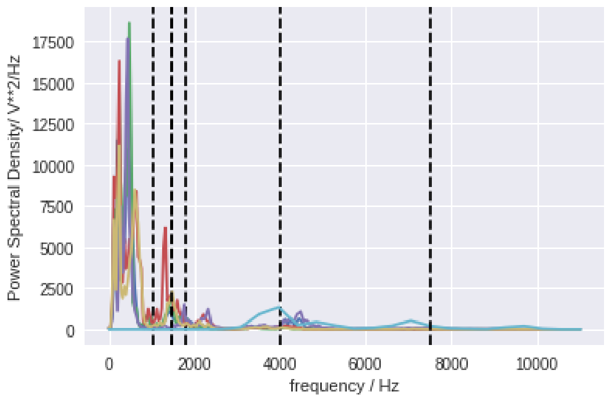

Figure 9: srf derivation for a typical speech signal

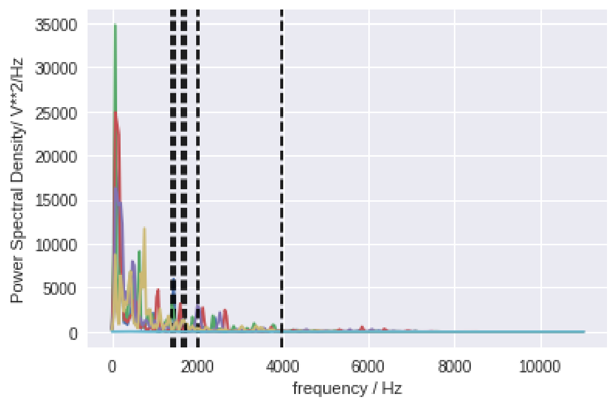

Figure 10: srf derivation for a typical music signal

4.2 Mel-frequency Cepstrum Features

4.2.1 Mel Scale

The human auditory system perceives sound on a different scale from the Hertz scale. Evenly spaced frequencies in Hertz do not generally correspond to what humans perceive as evenly spaced frequencies. The Mel scale attempts to map frequencies that humans perceive as evenly spaced to frequencies on the Hertz scale. The exact mapping and its inverse are shown in Equation 1 [7]:

$$m = 2595 log_{10}(1 + \frac{f}{700}) \tag{4}$$

$$f = 700(10^{\frac{m}{2595}} - 1) = 700(e^{\frac{m}{1127}} - 1) \tag{5}$$

where:

m = the frequency in mels
f = the frequency in Hertz

Figure 9 shows the mapping from Hertz to Mels for frequencies between 100 and 22000 Hz. Note that for larger values, frequencies appear closer together on the mel scale. This is because the human ear is less capable of distinguishing between higher frequencies, which is exploited in the design of the mel-frequency features.

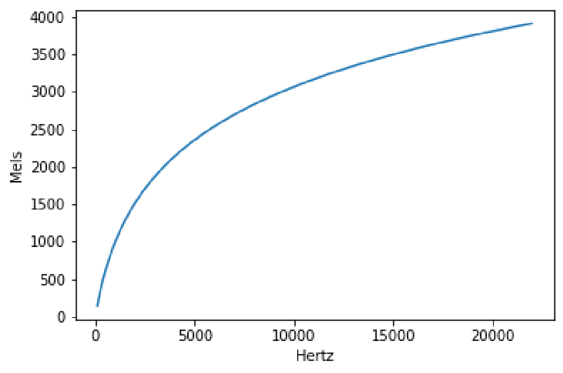

Figure 11: Mapping from Hertz to Mels for frequencies between 100 to 22000Hz

4.2.2 Mel-frequency Cepstral Coefficients

The Mel-frequency Cepstrum (MFC) is a representation of the short-term power spectrum of an audio signal based on the mel scale. Audio signals are converted to MFC representations, analogous to Fourier representations, in order to derive Mel-frequency Cepstral Coefficients (MFCC). [5] first introduced MFCCs to characterise speech signals in a manner close to human perception. They are a particularly attractive feature set, due to their ability to compactly represent the frequency characteristics of speech. As shown in this work, they provide a powerful, yet simple means of discriminating between speech and music. This is shown in Section 6.

For each 5 second audio track, the MFCC features were calculated as follows [4]:

1. Compute the periodogram estimate, $\mathbf{P}$ of the track using 20 millisecond frames with 50% overlap. The periodogram was calculated using Welch's method, configured to use a Hamming window.
2. Compute M mel-spaced filter banks. Following [4], the process took the following steps:
   (a) Map the maximum and minimum frequencies considered in the signal to the mel-scale using Equation (2) to get m_{max}, m_{min}
   (b) Construct a M+2 dimensional vector of evenly spaced mel frequencies from m_{min} to m_{max}
   (c) Map the vector back to the Hertz scale to obtain a vector $\mathbf{f}$ using Equation 3 [FIX]
   (d) Each filter, $H(k, m)$, in the filter bank is defined as

$$H(k, m) =$$

3. Perform the dot product of each filter H_m with the periodogram $\mathbf{P}$ to obtain M scalars. These are the filter bank energies. In this work, 26 filter banks were chosen, yielding a vector of length 26.
4. Take the logarithm of all the filter bank energies
5. Take the discrete cosine transform of the above result, in order to decorrelate the different frequencies.
6. Keep N coefficients. These coefficients are the Mel-frequency cepstral coefficients, or MFCC features.

The mel-frequency filterbank consists of a set of triangular filters that reach their peaks at frequencies that are evenly spaced on the mel scale. A filterbank of 26 filters, calculated for a sampling rate of 22050Hz and a minimum frequency of 300Hz, is shown in Figure 12.

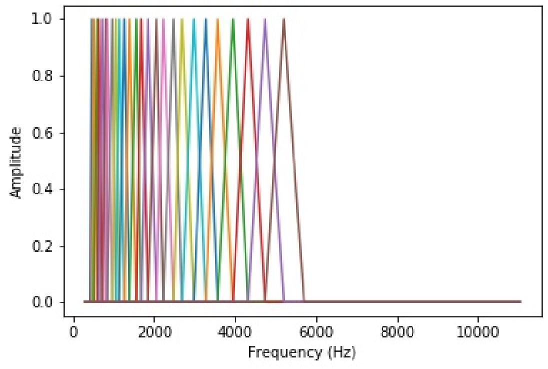

Figure 12: Plot of MFC filterbank for sampling rate 22050Hz

5 Classification

This work looks at two primary categories of classification algorithms: unsupervised and supervised classification. Because of the low data quantity, complex supervised approaches are not considered, as they would likely overfit the data. The inherent structure of the data in the feature space is examined by fitting simple clustering algorithms. For accurate performance it is required that the data is organized into 2 distinct clusters.

5.1 Unsupervised Classification

5.1.1 K-means

The K-means classification algorithm attempts to find a predefined number of clusters, k, such that the Euclidean distance between samples is maximized between clusters and minimized within them. This is equivalent to maximizing the *inter-cluster variance* and minimizing the *intra-cluster variance*. Each cluster is represented by a *cluster centroid*, whose value is the mean of the values of all the members of the cluster. For this work, features are extracted for each 5 second audio snippet, and then used as data points to be clustered. Two clusters are selected, representing speech, music. The cluster assignments are then treated as classifications. Given a new sample, its class is determined to be that of the cluster centroid to which it is closest. To determine the impact of the number of Mel filter banks and the number of features retained on the algorithm's performance, the algorithm is run for numerous configurations of the two variables. The v-measure (see Section 6.1) is then taken for each run and the results are plotted. The data is pre-processed by subtracting the mean and dividing it by the standard deviation along the feature columns.

5.2 Supervised Classification

5.2.1 Support Vector Machine

A Support Vector Machine (SVM) is fitted to the data to perform speech-music discrimination. An SVM attempts to find the hyper-plane in the feature space that maximally separates the input data points. In a non-linear variant of the SVM, data points are projected via a nonlinear mapping into a new feature space, allowing the algorithm to find non-linear separations of data. In this work, the non-linear variant is chosen, with a radial basis function kernel [9]. To form a comprehensive picture of performance, the model is trained and validated using 3-fold cross-validation. Here, the data is divided into 3 partitions, and for 3 permutations, trained on 2 of those partitions and validated on the 3rd. This training scheme is shown in Figure 13. The data is pre-processed by first calculating the mean and standard deviation of the training folds along the feature columns. The data is then scaled by subtracting the mean and dividing it by the standard deviation. The scaling is also applied to the test set using the values calculated on the training set.

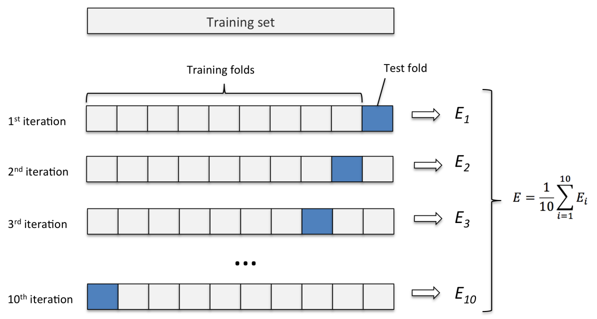

Figure 13: Visualisation of 10-fold cross validation training scheme. Data is divided into 10 evenly partitions. For 10 distinct permutations, the model is trained on 9 of the partitions and validated on the 10^{th}.

6 Evaluation Criteria

6.1 Clustering Performance

The performance of the clustering algorithm is determined by combining the *homogeneity* and *completeness* scores to form the *v-measure* [6]. For any given set of clustering assignments, the homogeneity of the assignments is a measure of the tendency for each cluster to contain only members of the same class. Note, however that it does not measure the spread of classes across clusters. Homogeneity ranges from 0 to 1, with 1 indicating perfect homogeneity. In this case, every cluster contains only one class. Completeness measures the tendency of a set of assignments to assign members of the same class to the same cluster. Note that an assignment may achieve a high completeness score even if multiple classes are assigned to the same cluster, which is undesirable. The v-measure combines the two scores according to the following formula:

$$V = 2\frac{Completeness \times Homogeneity}{Completeness + Homogeneity} \tag{6}$$

A high v-measure is achieved when a set of assignments is *homogeneous* and *complete*, meaning members of the same class tend to be assigned to the same cluster and each cluster tends to contain mostly members of the same class.

6.2 Supervised Classification Performance

As described in Section 5.2.1, an SVM is validated using the 3-fold cross validation scheme. The accuracy, precision, and recall of the model is taken after each run, and then the average of the accuracy is taken. Given that the training data is perfectly balanced, accuracy is an adequate measure of performance, as there is no opportunity for the model to score highly by only outputting the label of the majority class. These quantities are formally defined below.

$$Precision = \frac{TP}{TP + FP} \tag{7}$$

$$Recall = \frac{TP}{TP + FN} \tag{8}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{9}$$

where TP, TN, FN, FP are the number of True Positives, True Negatives, False Negatives, and False Positives in the binary classification class.

7 Results

7.1 Unsupervised Performance

7.1.1 Performance of K-means Algorithm on Low-Level Features

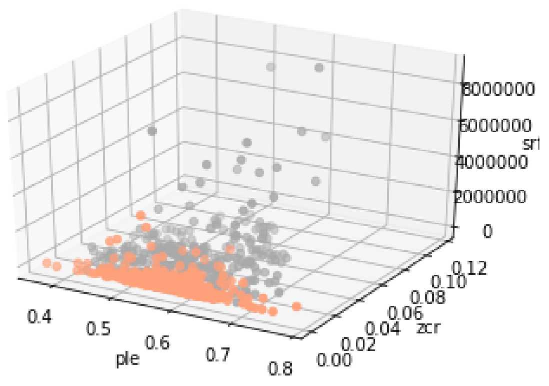

Figure 14: Visualisation of true values for ple, zcr and srf features

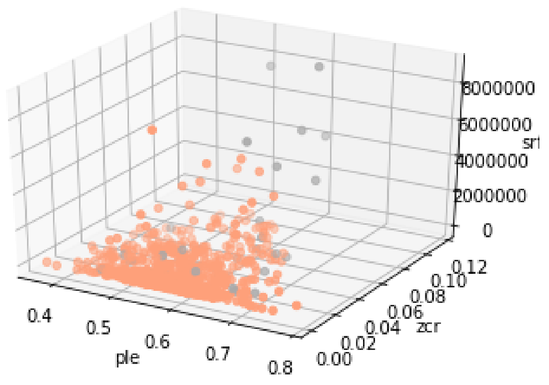

Figure 15: Visualisation of K-means Clustering Results for ple, zcr and srf features

While it should be noted that the visualisation above constitutes a reduction of the Low-Level feature vectors' dimensionality from 4 to 3, the visualisation of the spread of data points and their two values show that it is hard to distinguish sound from music based on the selected features using the K-means Algorithm as outlined in section 5.1.1

7.1.2 Performance of K-means Algorithm on MFCC Features

The K-means algorithm was applied to the MFCC features. The performance of the algorithm was measured for varying numbers of Mel-spaced filters and retained MFCC features. The effect of the number of filters is shown in Figure 16.

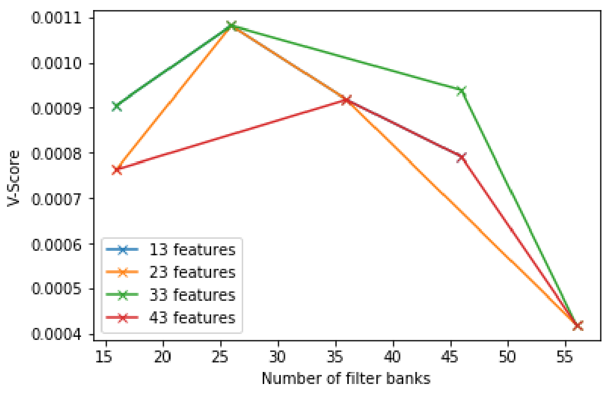

Figure 16: Impact of the number of filters on the v-score. Note that performance peaks at 26 filters.

Figure 17 shows the effect of the number of features on the clustering performance.

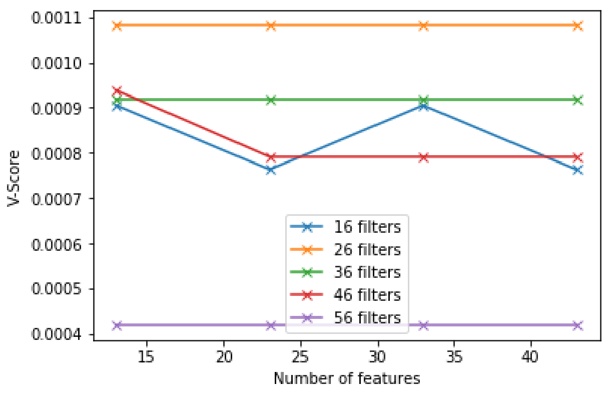

Figure 17: Impact of number of features on clustering performance. Note that, in general, the number of features has no impact on clustering performance.

7.1.3 Combined Features

The two separate Low-Level and MFCC feature vectors were not combined for K-means clustering as the method already proved ineffective for the individual features.

7.2 Supervised Performance

7.2.1 Performance of Support Vector Machine on Low-Level Features

An SVM was fitted to the low-level features vector, and performance is outlined in the table below:

Metric	Average	Standard Dev.
Train accuracy	0.863	0.011
Train precision	0.826	0.013
Train recall	0.919	0.010
Test accuracy	0.814	0.058
Test precision	0.776	0.083
Test recall	0.906	0.010

Figure 18: Table showing SVM Performance for Low-Level features. Performance was validated using 3-fold cross validation and the average performance over all runs was taken, along with the standard deviation.

7.2.2 Performance of Support Vector Machine on MFCC Features

An SVM was fitted to the MFCC features. The performance of the algorithm was measured for varying numbers of Mel-spaced filters and retained MFCC features. The effect of the number of filters is shown in Figure 19.

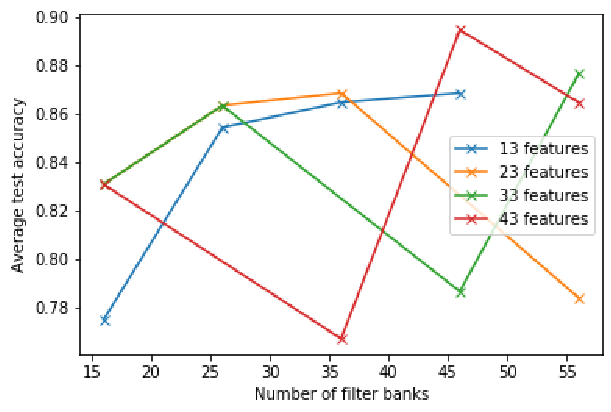

Figure 19: Impact of the number of filters on SVM test set accuracy. Note, the effect does not follow any general pattern.

Figure 20 shows the effect of the number of features on classification performance.

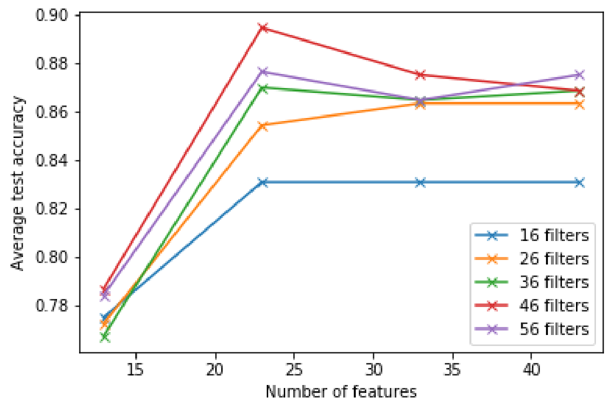

Figure 20: Impact of the number of features on SVM test set accuracy.

Metric	Average	Standard Dev.
Train accuracy	0.956	0.002
Train precision	0.961	0.000
Train recall	0.952	0.005
Test accuracy	0.803	0.029
Test precision	0.828	0.046
Test recall	0.711	0.005

Figure 21: Table showing SVM performance for MFCC features. Performance was validated using 3-fold cross validation and the average performance over all runs was taken, along with the standard deviation.

7.2.3 Combined Features

Metric	Average	Standard Dev.
Train accuracy	0.990	0.003
Train precision	0.983	0.005
Train recall	0.996	0.003
Test accuracy	0.903	0.021
Test precision	0.893	0.031
Test recall	0.927	0.022

Figure 22: Table showing SVM performance for combined features. Performance was validated using 3-fold cross validation and the average performance over all runs was taken, along with the standard deviation.

8 Analysis

8.1 Low-Level Features

8.1.1 Evaluation of K-means Performance

The K-means algorithm was found to perform poorly on the speech-music discrimination task. Looking at Figure 15, it is clear that the speech and music classes cannot be easily separated by the K-means criterion. when visualising three, not being able to accurately distinguish the music and speech classes over the four dimensional low-level feature space as evidenced in Table **??**. Yielding results with this approach would require the development of features that project the classes into two distinct clusters in the feature space.

8.1.2 Evaluation of SVM Performance

The SVM achieved a performance of 80.3% accuracy on the test set.

8.2 MFCC Features

8.2.1 Evaluation of K-means Performance

The K-means algorithm was found to perform extremely poorly on the speech-music discrimination task. The maximum v-measure was 0.001 across all configurations tested. This indicates that the two classes were not easily separable in the MFCC feature space. As shown in Figure 17, the number of filter banks had the largest effect on the clustering performance, but was inadequate to produce acceptable performance. The number of features used had no effect on clustering performance.

8.2.2 Evaluation of SVM Performance

The performance of the SVM achieved a maximum accuracy of 89% on the test set, with 46 filters and the first 23 MFCC features retained. As shown in Figure 19, the number of filters used had an inconsistent impact on the classification performance. Increasing the number of features used improved performance, with a peak at 23 features for most filter bank configurations.

8.3 Combined Features

8.3.1 Evaluation of K-means

Following the conclusions of the Low-Level and MFCC features sets as not being able to distinguish the classes of speech and music, no combined K-means classifier was pursued.

8.3.2 Evaluation of SVM Performance

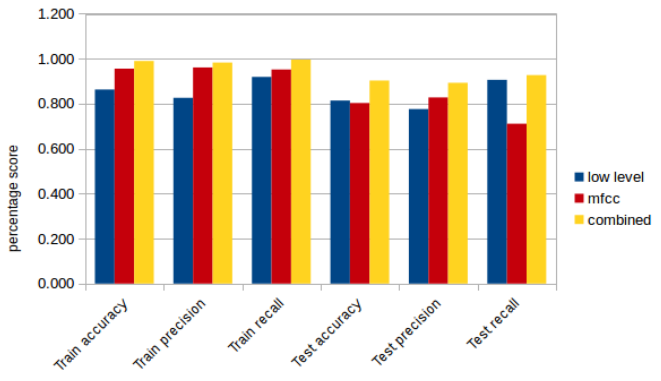

Figure 23: Comparison of SVM performance metrics for Low-Level, MFCC and combined feature vectors

9 Conclusion

The design of an audio signal classification system is heavily dependent on the features used. This work focused on the task of speech-music classification, and investigated the quality of the classifier produced using two different feature sets. Low-level signal features characterise an audio signal in terms of basic time-domain and frequency-domain features. Breebaart [3] proposed a split of features into 9 ”Low-Level” features, of which this work chose to inspect four, namely; rms value, percentage low-energy frames, zero-crossing rate and spectral roll-off resulting in a four dimensional low-level feature vector as detailed in section 4.1. The feature calculation was treated as per [2], calculating measures over short analysis windows, and incorporating long term characteristics by averaging measures in larger texture windows, and finally considering the variance of the averaged measure over these texture windows as the feature for a particular sample. In [7], O'Shaughnessy introduces the use of Mel-frequency Cepstral Coefficients (MFCC) to characterise human speech. The method applies a number of basic digital signal processing techniques to derive a fixed set of features for each audio sample. It was found in this work that, while MFCC features are difficult to classify using linear methods, non-linear methods manage to achieve reasonable accuracy. Finally, both feature sets were combined to determine the impact on classification performance. As reported in Section 8.3, the overall performance of the classifier improved substantially, compared to using either one of the feature sets alone. This work has shown that both low-level signal parameters and MFCC features are sufficient to build a classifier with a minimum accuracy of 80% on unseen data.

References

[1] Gerhard, D., 2003. Audio signal classification: History and current techniques. Department of Computer Science, University of Regina.

[2] Scheirer, E. and Slaney, M., 1997, April. Construction and evaluation of a robust multifeature speech/music discriminator. In Acoustics, Speech, and Signal Processing, 1997. ICASSP-97., 1997 IEEE International Conference on (Vol. 2, pp. 1331-1334). IEEE.

[3] McKinney, M. and Breebaart, J., 2003. Features for audio and music classification.

[4] Sigurdsson, S., Petersen, K.B. and Lehn-Schiøler, T., 2006, October. Mel Frequency Cepstral Coefficients: An Evaluation of Robustness of MP3 Encoded Music. In ISMIR (pp. 286-289).

[5] J. Foote. Content-based retrieval of music and audio. In Multimedia Storage and Archiving Systems II, Proc. of SPIE, volume 3229, pages 138–147, 1997.

[6] Rosenberg, A. and Hirschberg, J., 2007. V-measure: A conditional entropy-based external cluster evaluation measure. In Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL).

[7] O'Shaughnessy, D., 1987. Speech communication: human and machine. Universities press.

[8] Marsyas.info. (2018). [online] Available at: http://marsyas.info/downloads/datasets.html [Accessed 22 Apr. 2018].

[9] Mohri, M., Rostamizadeh, A. and Talwalkar, A., 2012. Foundations of machine learning. MIT press.