Mini Project – II
VIth Semester
Report

INFANT CRY ANALYSIS

Submitted in partial fulfilment of
the requirements of the term work for subject Mini Project-I

Submitted by

Roll No.	Name of Student
2020200018	Dil Gupta
2020200035	Ruchi Kowarkar
2020200020	Vijay Prasanna Gurubaran

Under the Guidance of
Prof. Sujata Kulkarni

&

Dr. Sukanya A. Kulkarni

Department of Electronics and Telecommunications Engineering
Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Munshi Nagar, Andheri(W), Mumbai-400058
University of Mumbai

Academic Year 2020-21

CERTIFICATE

This is to certify that this is a bona de record of the project presented by the students whose names are given below during semester V in partial fulfilment of the requirements of the degree of Bachelor of Engineering in Electronics Engineering.

Roll No.	Name of Student
2020200018	Dil Gupta
2020200035	Ruchi Kowarkar
2020200020	Vijay Prasanna Gurubaran

External Examiner

(signature)

Name: Deepak Karia

Date: 29th May, 2023

Internal Examiner

(signature)

Name: Sukanya Kulkarni

Date: 29th May, 2023

Seal of Institute

Abstract

It has always been difficult to figure out the exact reason why an infant is crying. However, modern machine learning techniques have made it possible to predict the reason how this could be possible. This report summarizes a machine learning project that predicts the reason for why an infant is crying. The dataset has been taken and pre-processed with techniques such as silence removal, normalization, pre-emphasis, hamming window. Feature extraction is an important step in any machine learning project and the feature extraction techniques that have been used in this project are Zero Crossing Rate, Short Time Energy and MFCC's. After the pre-processing and feature extraction step, we finally get to the classification part. The machine learning models that were used were Naïve-Bayes, Random Forest and KNN models.

Keywords: ZCR, STE, MFCC, Random Forest

Acknowledgements

We extend our gratitude to our mentor, Prof. Sujata Kulkarni, Department of Electronics and Telecommunications Engineering, along with our internal examiner, Dr. Sukanya A. Kulkarni, Associate Professor, Department of Electronics and Telecommunications Engineering, S.P.I.T, who have helped us in our project. Without their support, guidance and technical knowledge, our project would have not been successful.

Students Names:

Dil Gupta
Ruchi Kowarkar
Vijay Prasanna Gurubaran

Contents:

Chapter 1

Introduction

Infant crying occurs as a result of a certain problem, and many parents become really stressed trying to calm the baby down. The cries do not really have a discernible pattern that parents can inspect. Due to this, parents might spend a lot of time trying to calm the baby down, time which can be used to put bread on the table. This has given rise to the need to answer the reason why a baby might be crying.

One of the easiest ways to do that might be to analyze the cries of the baby. Using audio processing techniques, it might be able to see a difference in the cries of a baby for different reasons. Using machine learning techniques, we can analyze these audio processing techniques to create a model that can accurately predict the reason why a baby is crying.

1.1 Problem Statement

The problem statement of this report is:

To create a machine learning model that can predict the reason for an infant's cry by analyzing the features that are extracted from audio processing techniques.

Chapter 2

Literature Survey

Authors	**Name of Literature**	**Insight**
Lina Abou-Abbas, Chakib Tadj, Hesam Alaie Fersaie	A fully automated approach for baby cry signal segmentation and boundary detection of expiratory and inspiratory episodes	A framework for automatic cry sound segmentation for use in diagnostic systems, monitoring systems, and robotics for baby caregivers.
Amany Mounes Mahmoud, Sarah Mohamed Swilem, Abrar Saeed Alqarni, Fazilah Haron	Infant Cry Classification Using Semi-supervised KNearest Neighbor Approach	An infant cry classification model has been used with normal KNN, semi-supervised KNN with no selection and semi-supervised KNN with selection criteria.
Pradeep G C M, Tejaswini S and Natarajan Sriraam	Recognition of Infant Cries Using Wavelet Derived Mel Frequency Feature with SVM Classification	Wavelet transform and Mel frequency estimation were used to analyse infant cries for clinical investigation. SVM classifier achieved promising results.
L. Harshita, K.G.S. Sai Veni, P. Rohit Varma, A. Hari Narayana, P. Nikhil	Baby Cry Classification Using Machine Learning Algorithms	Voice Activity Detection and how it's not the best way to go about in infant cry analysis. Detection of inhalation and exhalation efficiently is more useful.

2.1-EXP and INSV:

The most important components of the infant cry signals are expiration and inspiration segments (EXP and INSV, respectively). The most common technique used to detect these segments is the Voice Activity Detection (VAD). VAD is not really reliable on a dataset where the cries have been recorded in a noisy hospital environment (Abou-Abbas et al., 2017). Following are the reasons why VAD techniques are not as useful:

- Threshold settings were not easy to select in a variable and noisy environment.
- Traditional VAD could not distinguish between important cry segments (EXP and INSV) and speech segments recorded during the data acquisition.
- Traditional VAD failed in distinguishing expiration phases from inspiration phases of cry signals.

2.2-Feature Extraction Techniques:

2.2.1-LFCC's:

Linear Frequency Cepstral Coefficient method is a method that is used for extracting speech recognition features. In an existing system, a pre-emphasis filter was used followed by frame blocking and windowing and finally an FFT was taken before passing it to the filter bank for the LFCC (Sita Purnama Dewi et al., 2019).

2.2.2-MFCC's:

The best feature extraction technique that has been used in almost all of the literature that has been surveyed is Mel-Frequency Cepstral Coefficients (MFCC) (Tejaswini S et al., 2016). MFCCs are a widely used feature extraction technique for speech and audio signal processing. MFCCs are a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear Mel frequency scale.

MFCC uses a filter bank that is based on the human auditory system, where filters are spaced linearly at lower frequencies and logarithmically at higher frequencies. LFCC uses a filter bank that is linearly spaced across the entire frequency range.

2.3-Machine Learning Techniques:

Support vector machine (SVM) is the most common technique used in most of the literature that has been surveyed. However, K-Nearest Neighbours (KNN) and deep learning techniques have also been used quite often. In the paper, Recognition of Infant Cries Using Wavelet Derived Mel Frequency Feature with SVM Classification, by Tejaswini S. et al., the classification accuracy using SVM was 93.095% for hunger vs. discomfort, 90.27% for pain vs. hunger and 71.29% for discomfort vs pain. Mahmoud, Swilem, Alqarni and Haron in their paper, Infant Cry Classification Using Semi-supervised KNearest Neighbor Approach, used KNN and semi-supervised KNN techniques and witnessed an 87% and 94% accuracy respectively.

Chapter 3

Theory

3.1 What is pre-processing of a signal?

Pre-processing of a signal refers to the various techniques and operations that are performed on the signal data prior to analysis or further processing. Pre-processing is an essential step in signal processing applications as it helps to remove noise, artifacts and other unwanted components from the signal and prepares it for subsequent analysis.

Pre-processing is an essential step in audio signal processing applications as it helps to remove noise, artifacts and other unwanted components from the audio signal and prepares it for subsequent analysis.

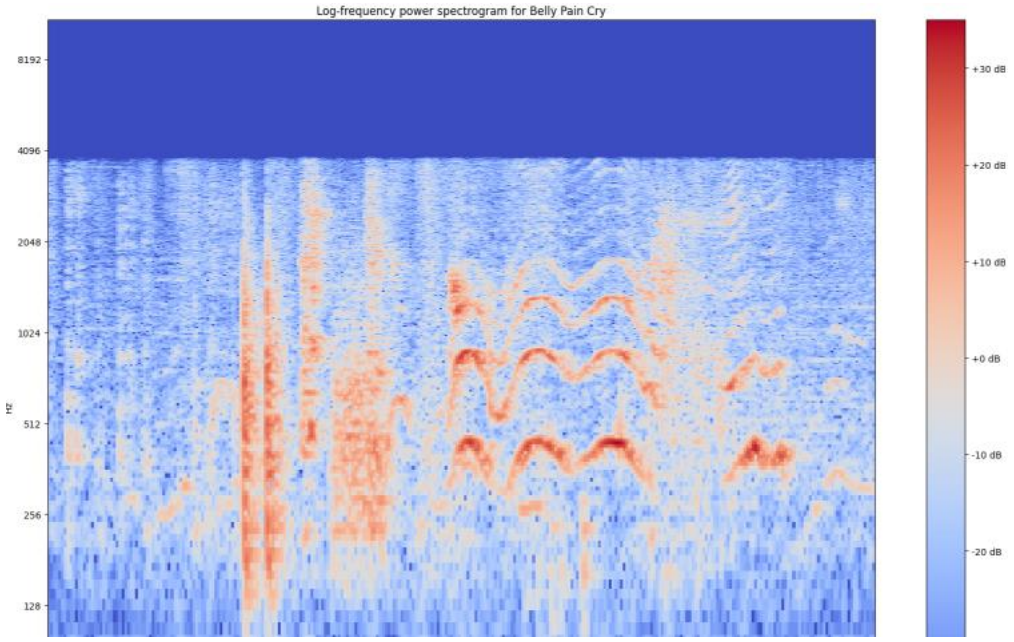
Pre-processing techniques needed for the project:

1. Sampling and Quantization: Sampling is the process of taking points from the signal periodically. In quantization, we plot each value and approximate it to the closest value (the 4 bits column on the right-side). The higher the quantization resolution (that is, more numbers on the right), lower will be the quantization error. In our projects, we have omitted this process as the dataset that we have used have already included these processes.
2. Silence removal: Silent parts of the audio signal were removed.
3. Normalization: Normalization of audio signals is the process of adjusting the overall volume level of an audio recording or a portion of it to a desired level.
4. Pre-emphasis: A high-pass filtering technique that amplifies the high-frequency components of an audio signal.
5. Windowing: A mathematical function that is zero-valued outside of a finite interval and is used to taper the edges of a signal section at the interval endpoints.

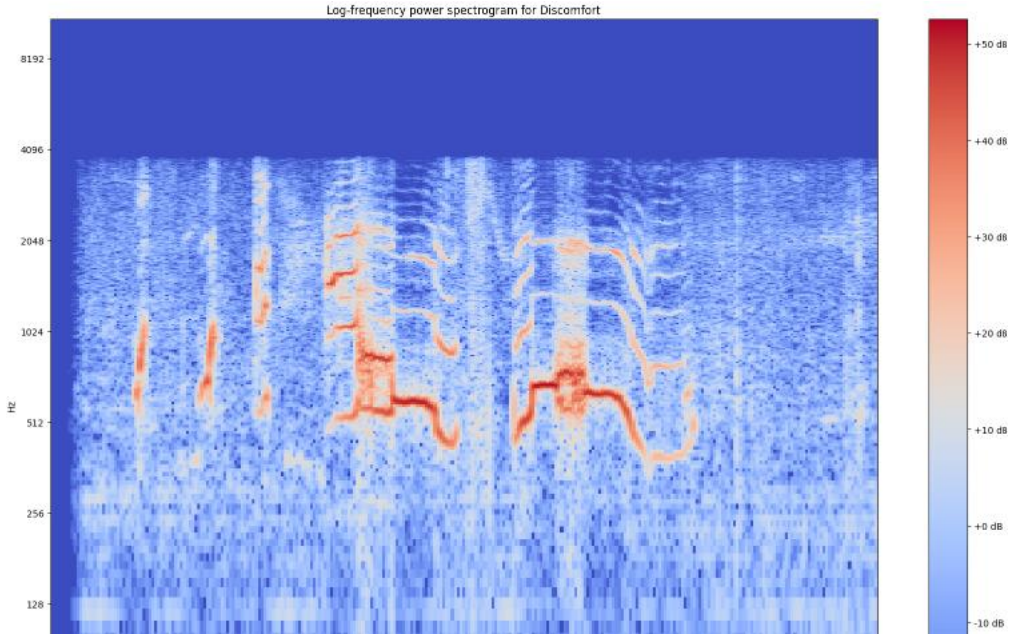
3.2 Feature Extraction:

The audio signals that have been used can be differentiated as below:

For belly pain, the spectrogram (frequency vs. time) is:



For discomfort, the spectrogram is:



As we can see, the important parts of discomfort audio signal have higher frequencies than the important parts of belly pain. These are the kinds of differences in the audio file that can be used to differentiate in the machine learning model. This is known as feature extraction.

Feature extraction of audio signals is the process of extracting meaningful and relevant information from the raw audio signal, which can then be used as input to various machine learning or signal processing algorithms for further analysis, classification, or recognition purposes. The following are the points that the extracted features must satisfy:

- Display of characteristics of speech such as vocal tract characteristics of the speakers and sense of hearing characteristics
- The features must be easy to compute
- The features must be impervious to imitation
- Perfect in describing environment variation

Speaker recognition is another aspect of audio processing applications. The features for speaker recognition are:

- Short-term spectral features
- Voice source features
- Spectral-temporal and prosodic features
- High-level features

The difficulty of extracting increases from short-term spectral features to high-level features. However, susceptibility to noise decreases.

- Because they are the most straightforward and discriminative, short-term spectral features are frequently used in speech and speaker recognition.
- Modern recognition systems frequently combine these aspects in an effort to produce more accurate recognition outcomes.
- The transient spectral features reveal details about the spectral envelope.
- The spectral envelope is frequently used to identify speakers because it contains information on the speaker's vocal tract features, such as the location and size of the peaks (formants) in the spectrum.

Because of the above reasons, this project will be using short-term spectral features only.

Nowadays, the most common short-term features used in speech and speaker recognition are:

- Short-time Energy (STE)
- Short-time Zero Crossing Rate (ZCR)
- Short-time autocorrelation function
- Linear Predictive Coding (LPC)
- Linear Predictive Cepstrum Coefficients (LPCCs)

- Log Frequency Power Coefficients (LFPCs)
- Perceptual Linear Prediction (PLP)
- Subband Energy
- Wavelet Transform (WT)
- Formant
- Pitch
- Mel-Frequency Cepstral Coefficients (MFCCs)

The main techniques that this project will be using are zero crossing rate (ZCR), short term energy (STE) and mel frequency cepstral coefficients (MFCC). Each of the techniques has been described as below:

[1] Zero Crossing Rate:

The rate at which a signal changes from positive to zero to negative or from negative to zero to positive is called the zero-crossing rate (ZCR). Zero-Crossing Rate (ZCR) is the rate at which a signal changes from positive to zero to negative or from negative to zero to positive. It is a commonly used feature in audio signal processing for detecting the frequency and rhythmic properties of a signal.

A higher ZCR indicates a higher frequency or more variation in the signal, while a lower ZCR indicates a lower frequency or less variation.

[2] Short Time Energy:

Short-time energy (STE) is a measure of the energy of a signal in a short window of time. It is calculated by squaring the values of the signal in a short-time frame, and then summing them up. The function used for STE in this project is librosa.feature.rms(y=emphasized_signal, frame_length=2048, hop_length=512).

[3] MFCC's:

The use of MFCCs allows for the extraction of significant features from audio signals, such as speech or music, which can then be applied to tasks like speaker identification, speech recognition, and music genre classification.

The goal of MFCCs is to provide machine learning algorithms with a compact representation of the most important aspects of the frequency content of the audio signal.

To accomplish this, MFCCs first use the Fast Fourier Transform (FFT), which separates the audio signal into its component frequencies, to transform the audio signal from the time domain to the frequency domain. The frequencies are then

organised into a set of Mel-scaled frequency bands, which differ from linear frequency bands in their spacing because they mimic how the human ear perceives sound.

What is Mel-scale?

To create the Mel-scale, we use triangular filters that overlap with each other on the frequency spectrum. The Mel-scale is a sound measurement system that is based on how pitch differences are perceived by humans. The range of sounds that humans can hear is represented by the scale, which runs from 20 Hz to 20 kHz. Since the Mel-scale is logarithmic, the pitch difference between two sounds is determined by the ratio of their frequencies rather than by the actual hertz difference.

These filters are made to mimic the human ear's frequency selectivity and are evenly spaced along the Mel-scale. Every filter counts the energy within a specific frequency band. The frequency spectrum of an audio signal is frequently converted into a set of Mel-scaled frequency bands during audio processing. These bands can be used as input features for models that use machine learning or other types of analysis. The Mel-scale and the MFCC feature extraction method can be used together to extract crucial spectral data from the audio signal.

Feature Extraction using MFCC:

In this project, we extract MFCC’s from the audio file. The MFCC’s are a 13x302 matrix. We also concatenated the delta coefficients and created a 39x302 matrix.

Why we chose MFCC?

Following are the advantages of MFCC over other techniques:

- Relatively insensitive to background noise as compared to these other methods.
- Allows for effective speaker normalization.
- Takes into account the human auditory system’s sensitivity to frequency ranges. The Mel scale that is used in the MFCC method maps the frequency spectrum to a non-linear spectrum, which can be observed and comprehended much easily.
- Less computational complexity as compared to other techniques.

3.3 Classification process:

The final step in this project is the classification process. This process uses machine learning techniques to classify the infant cry reasons into 5 categories:

1. Discomfort
2. Hunger
3. Belly pain
4. Tired
5. Burping

The main techniques used were Naïve Bayes, Random Forest and KNN. Each of these techniques are explained below:

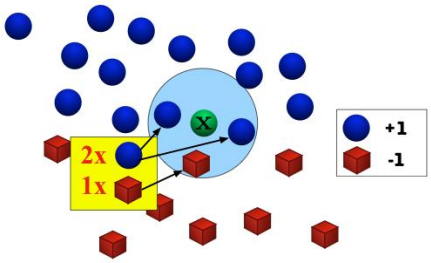
[1] KNN:

KNN stands for k-nearest neighbors, which is a supervised machine learning algorithm used for both classification and regression tasks. In the KNN algorithm, the training data consists of a set of labeled points in a multidimensional feature space. When a new input point needs to be classified or predicted, the algorithm finds the k closest labeled points (i.e., nearest neighbors) to the input point in the feature space. The output label or value of the input point is then determined by taking a majority vote or weighted average of the labels or values of its k nearest neighbors.

The value of k is a hyperparameter of the algorithm that needs to be specified by the user. A larger value of k typically results in a smoother decision boundary and lower variance, but higher bias. A smaller value of k can lead to more complex decision boundaries and higher variance, but lower bias.

KNN is well suited to handle high-dimensional feature spaces like MFCCs, as it can compare the distance between the feature vectors of the training and test data directly. KNN is a non-parametric algorithm, meaning it does not make any assumptions about the underlying distribution of the data. This is particularly useful for audio data, as the distribution of audio features can be complex and difficult to model parametrically.

Below is an example from a lecture from Cornell university. It has k (n_neighbours) = 3. The blue circle is drawn to keep 3 neighbours to the green value (x) within it.



For the above example, set of the k nearest neighbours of x as S_x:

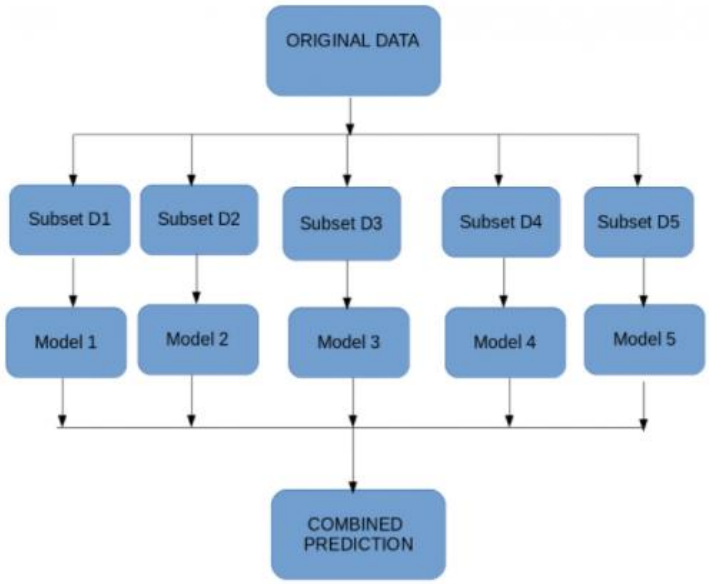
$$dist(x, x') \geq \max_{(x'', y'') \in S_x} (x, x'')$$

Here x is the point we are focusing on and x′ is a neighbouring point. D is the set of all points and Sx is the points within the blue circle. The above inequation simply means, all the points within D that are not in Sx must be at least.

[2] Random Forest Classifier:

Random forest classification is a type of machine learning algorithm that is used for supervised learning tasks, particularly for classification problems. It is a type of ensemble learning technique that involves building multiple decision trees at training time and aggregating their outputs to make predictions. In a random forest classifier, each decision tree is constructed using a random subset of the features in the training set, which helps to reduce overfitting and increase the robustness of the model. The final prediction is then made by aggregating the predictions of all the individual decision trees in the forest.

Random forest classification technique is also extremely helpful in audio processing. In this project, the classification technique analyzes the MFCC features that were extracted in the feature extraction step. The following diagram was taken from analyticsvidhya.com:



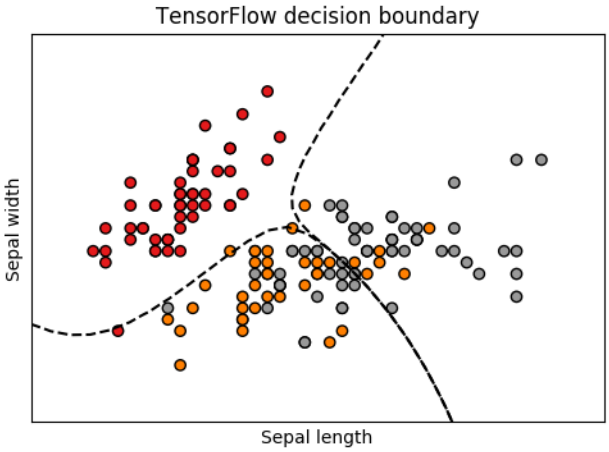
[3] Naïve Bayes:

Naive Bayes is a type of probabilistic machine learning algorithm that is commonly used for classification tasks, such as text classification or spam filtering. It is based on the Bayes' theorem that calculates the probability of a hypothesis based on prior knowledge or evidence. In Naive Bayes, the probability of each class is estimated independently from the other classes, assuming that the features (or variables) are conditionally independent given the class label. This assumption of independence is often relaxed in practice, but the algorithm can still perform well even with dependencies. Naive Bayes is called "naive" because it simplifies the problem by assuming that all features are equally important and have the same impact on the prediction, regardless of their correlations. Despite its simplicity, Naive Bayes can be very effective in many real-world applications, especially when the number of features is large or when there is limited training data available.

Naive Bayes is useful for audio processing classification because it is a simple and efficient algorithm that can handle high-dimensional feature vectors with a relatively small amount of training data. Here are some specific ways that Naive Bayes can be beneficial for audio classification:

1. Fast and Efficient: Naive Bayes is a simple algorithm that can classify audio signals quickly, making it well-suited for real-time applications such as speech recognition or audio event detection.
2. Small Training Data Requirements: Naive Bayes requires relatively little training data compared to other classification algorithms, making it useful when training data is limited or expensive to obtain.
3. Handles High-Dimensional Feature Vectors: Audio signals typically have high-dimensional feature vectors, such as MFCCs, which can be difficult for some algorithms to handle. Naive Bayes is well-suited to handle these high-dimensional feature vectors because it assumes that each feature is independent of the others.
4. Handles Categorical and Continuous Features: Audio features can be either categorical (e.g., speech vs. music) or continuous (e.g., MFCCs). Naive Bayes is a probabilistic algorithm that can handle both types of features by modeling the likelihood of each feature given the class label.

The following diagram shows the application of Naïve Bayes classifier on an iris flower sepal dataset by nicolovaligi on GitHub:

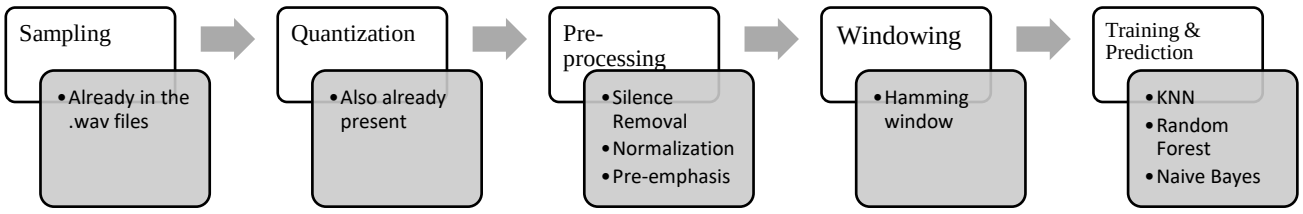


Each colour of dot is a species of iris flower and this was plotted as sepal width vs. sepal length. The red dot flowers had lower sepal widths and lengths, the orange and grey ones were very similar except the orange ones were more spread out and the grey ones had longer sepals. It should also be noted that despite there being a decision boundary, there are overlaps and some dots go into the other colour's areas and vice versa. This indicates the decision boundary is not absolutely perfect.

Chapter 4

Methodology

The below is the flow-chart for the methodology used in this project:



1. Sampling & Quantization:

This step has been omitted from our project due to the fact that the dataset that is used contains .wav files that have already gone through both sampling and quantization processes.

2. Silence Removal:

Silent parts of the audio signal has been removed using librosa.effects.trim() function, default parameters were taken.

- The threshold was taken as 60 dB. Below this value, the part would be considered as silent.
- The frame length, that is, the number of samples per analysis frame was taken to be 2048.
- The number of samples between analysis frames, that is the hop length, was taken as 512.

3. Normalization:

Normalization has been done using the librosa.util.normalize() function. The default parameters were taken, which are norm = infinity, no threshold and no fill.

4. Pre-emphasis:

A pre-emphasis coefficient of 0.97 was taken. 0.97 means stronger amplification of high frequencies.

5. Hamming Window:

Hamming window was used. It has a raised cosine shape and is defined as $w(n) = 0.54 - 0.46\cos(2\pi n/N)$. Hamming windows are usually used for speech processing.

6. Zero-Crossing Rate:

The frame length was taken as 2048 and hop length as 512, same as for silence removal.

7. STE:

The same parameters were taken for short-time energy as well.

8. MFCC:

n_mfcc is set to 13, which means that the function will extract 13 MFCCs from the audio signal emphasized_signal with a sampling rate of sr. The resulting MFCC variable will be a 2D array with shape (n_mfcc, n_frames), where n_frames is the number of frames in the audio signal.

9. Model Training:

Three classification techniques were used and evaluated and compared with each other. These three were k-Nearest neighbours, random forest classifier and Naïve Bayes classifier.

9.1 KNN:

The KNN algorithm was used with the value of k (ie, the number of neighbours) taken to be 3.

9.2 Random Forest Classifier:

The random forest algorithm was used with the number of estimators equal to 100, ie, the number of decision trees created is taken as 100.

9.3 Naïve Bayes Classifier:

Naïve Bayes classifier was used. In our project, Gaussian naïve Bayes was used over multinomial. This is because the nature of the MFCC's is almost continuous.

10. Prediction:

The best model out of the above three was taken and then used for prediction of a random audio file. The model used was random forest classifier which has an accuracy of 78.26%.

Chapter 5

Experimental Analysis and Result

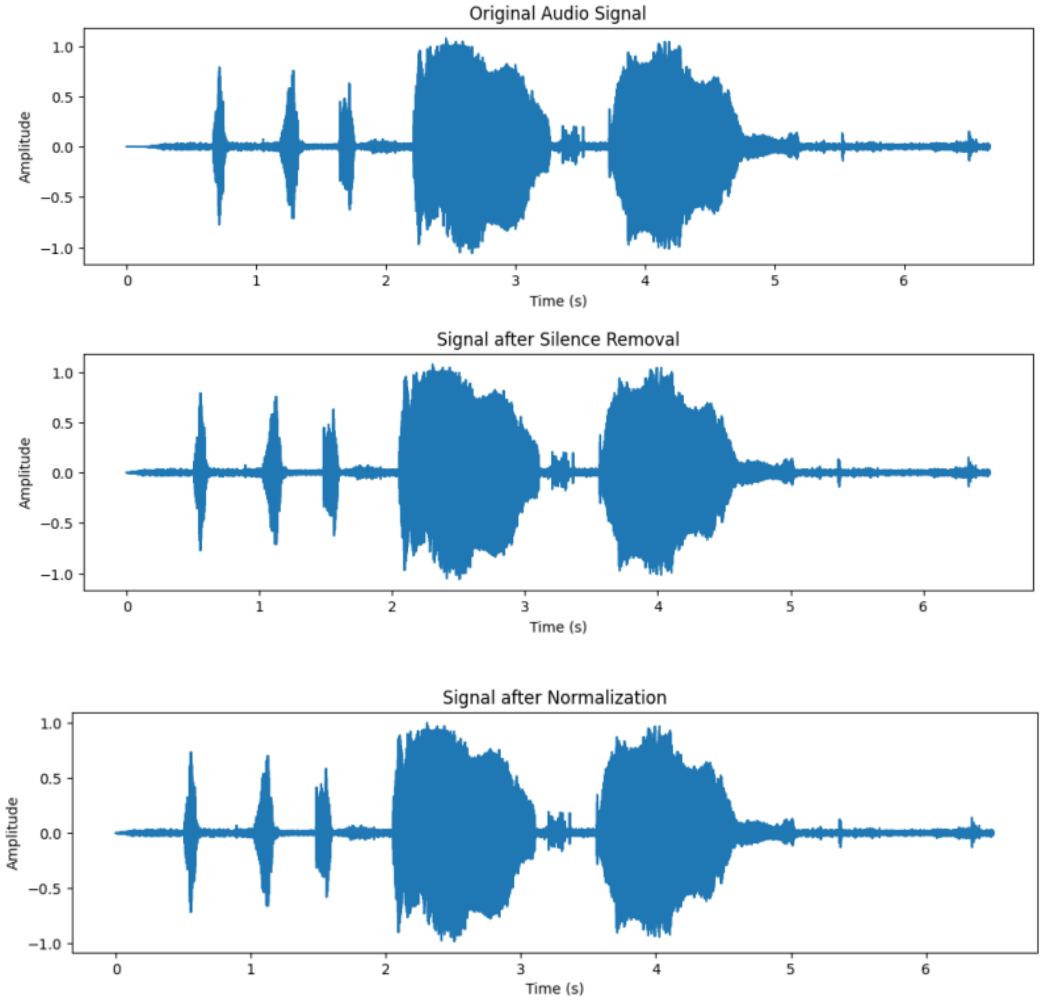
The code and the output for feature extraction and demonstration of it is given below:

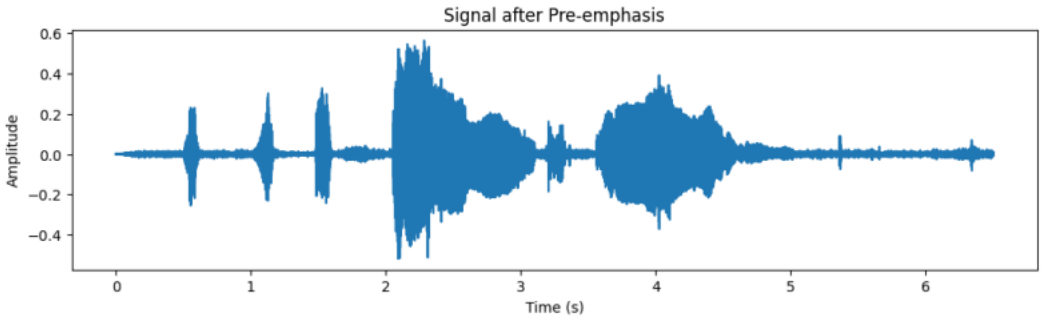
```
import librosa
import numpy as np
import matplotlib.pyplot as plt
# Load audio file
audio_file = "D:\Dil\S.P.I.T Semester 6\Mini project\dataset\discomfort
\discomfort (3).wav"
signal, sr = librosa.load(audio_file, sr=16000)
# Plot original audio signal
plt.figure(figsize=(12, 3))
plt.title("Original Audio Signal")
plt.plot(np.arange(len(signal)) / sr, signal)
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.show()
# Sampling
# In this example, we are already using a sample rate of 16000 Hz so we
 do not need to resample
the audio signal.
# Quantization
# Quantization is done implicitly during the audio file loading process
 with librosa.
# Silence Removal
# Remove leading and trailing silence
signal, _ = librosa.effects.trim(signal)
# Plot signal after silence removal
plt.figure(figsize=(12, 3))
plt.title("Signal after Silence Removal")
plt.plot(np.arange(len(signal)) / sr, signal)
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.show()
# Normalization
signal = librosa.util.normalize(signal)
# Plot signal after normalization
plt.figure(figsize=(12, 3))
plt.title("Signal after Normalization")
plt.plot(np.arange(len(signal)) / sr, signal)
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.show()
# Pre-emphasis
pre_emphasis = 0.97
```

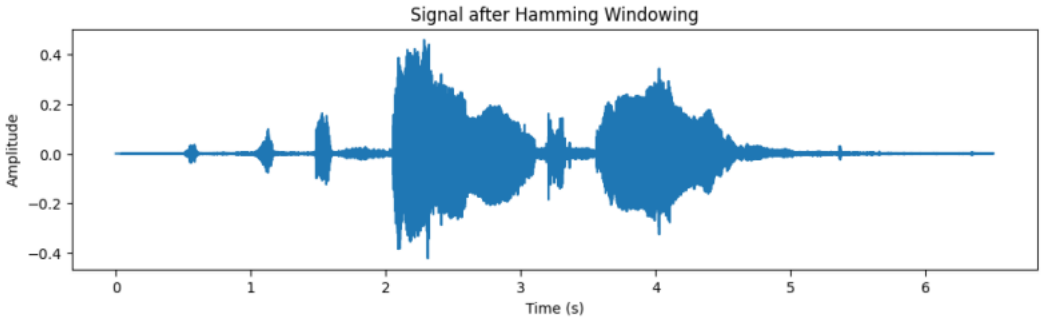
```
emphasized_signal = np.append(signal[0], signal[1:] - pre_emphasis * si
gnal[:-1])
# Plot signal after pre-emphasis
plt.figure(figsize=(12, 3))
plt.title("Signal after Pre-emphasis")
plt.plot(np.arange(len(emphasized_signal)) / sr, emphasized_signal)
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.show()
# Hamming Windowing
hamming_window = np.hamming(len(emphasized_signal))
emphasized_signal = emphasized_signal * hamming_window
# Plot signal after Hamming windowing
plt.figure(figsize=(12, 3))
plt.title("Signal after Hamming Windowing")
plt.plot(np.arange(len(emphasized_signal)) / sr, emphasized_signal)
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.show()
# Zero-crossing rate (ZCR)
zcr = librosa.feature.zero_crossing_rate(y=emphasized_signal, frame_len
gth=2048,
hop_length=512)
# Get the mean of the ZCR features
zcr_mean = np.mean(zcr, axis=1)
# Plot ZCR
plt.figure(figsize=(10, 4))
plt.title("ZCR")
plt.imshow(zcr, cmap="coolwarm", aspect="auto")
plt.colorbar()
plt.xlabel("Frame")
plt.ylabel("ZCR")
plt.show()
# Short-time energy (STE)
ste = librosa.feature.rms(y=emphasized_signal, frame_length=2048, hop_l
ength=512)
# Get the mean of the STE features
ste_mean = np.mean(ste, axis=1)
# Plot STE
plt.figure(figsize=(10, 4))
plt.title("STE")
plt.imshow(ste, cmap="coolwarm", aspect="auto")
plt.colorbar()
plt.xlabel("Frame")
plt.ylabel("STE")
plt.show()
# MFCC
n_mfcc = 13
```

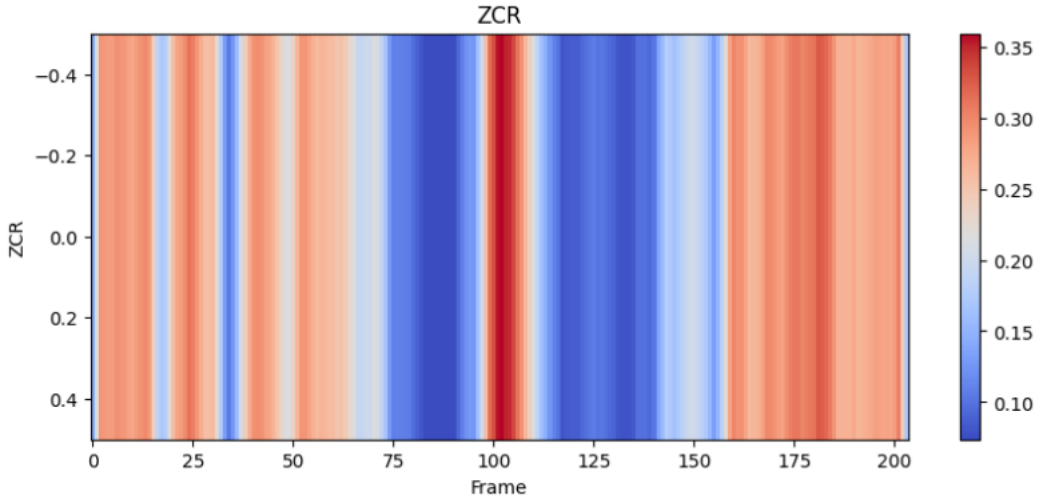
```
mfcc = librosa.feature.mfcc(y=emphasized_signal, sr=sr, n_mfcc=n_mfcc)
# Get the mean of the MFCC features
mfcc_mean = np.mean(mfcc, axis=1)
# Plot MFCC
plt.figure(figsize=(10, 4))
plt.title("MFCC")
plt.imshow(mfcc, cmap="coolwarm", aspect="auto")
plt.colorbar()
plt.xlabel("Frame")
plt.ylabel("MFCC Coefficients")
plt.show()
# Print MFCC coefficients and their mean
print("MFCC Coefficients:\n", mfcc)
 print("MFCC Mean:\n", mfcc_mean)
```

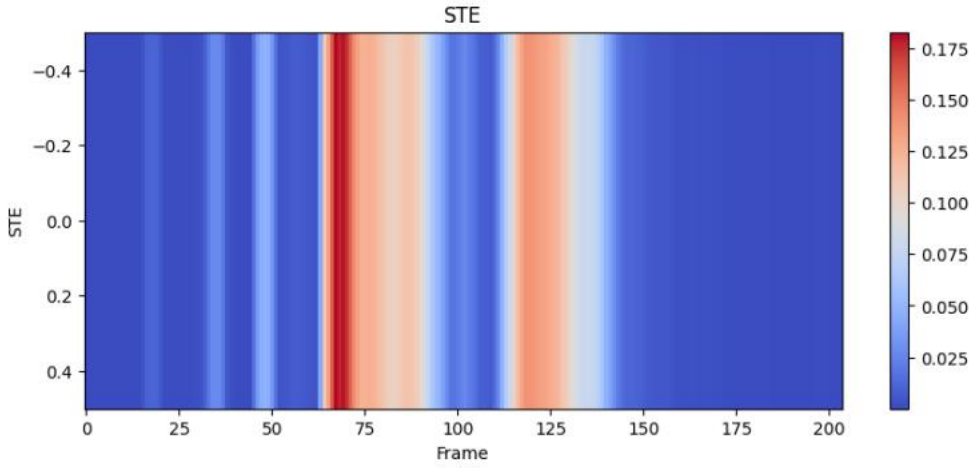
Output:

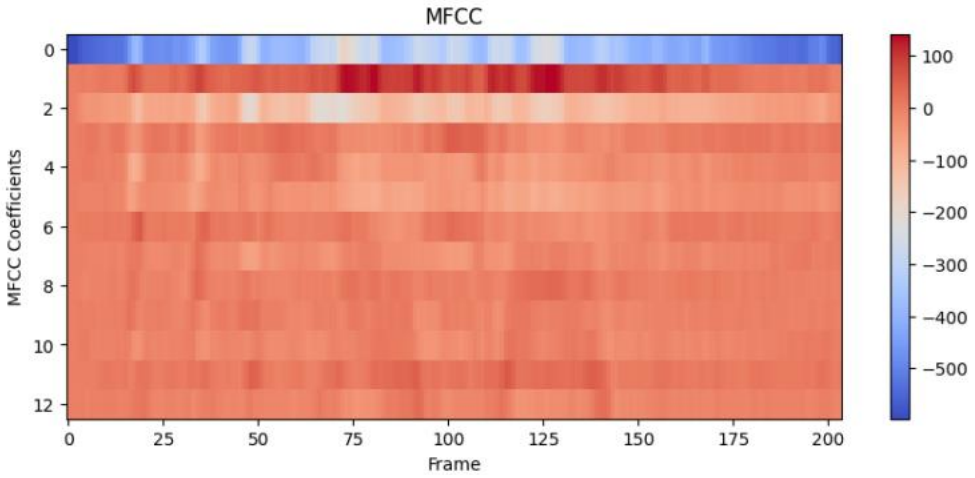



Signal after Pre-emphasis
Amplitude
Time (s)


Signal after Hamming Windowing
Amplitude
Time (s)


ZCR
ZCR
Frame






```
MFCC Coefficients:
 [[-5.97369436e+02 -5.97300174e+02 -5.91679004e+02 ... -5.53093696e+02
  -5.50870629e+02 -5.67637747e+02]
 [ 0.00000000e+00  7.64734300e-02 -1.25120889e+00 ... -4.08808317e+00
  -4.11240908e+00 -6.35989925e+00]
 [ 0.00000000e+00  2.14610278e-02 -5.43039280e+00 ... -3.51766184e+01
  -3.55628756e+01 -2.52274711e+01]
 ...
 [ 0.00000000e+00  8.74915703e-02 -3.45157262e-02 ...  5.37210718e+00
    1.82345216e+00  1.71412126e+00]
 [ 0.00000000e+00  4.07893920e-02  1.01860264e+00 ...  5.47269553e+00
    3.27509652e+00  4.03900027e+00]
 [ 0.00000000e+00 -2.37999766e-02 -1.07047320e+00 ... -3.46773037e+00

  -8.87093837e+00 -8.57448096e+00]]
MFCC Mean:
 [-3.97350301e+02  5.81444640e+01 -1.01609645e+02  2.02241357e+00
 -2.42364971e+01 -4.01145482e+01  1.72999466e-01 -1.46085487e+01
  2.99432878e+00 -1.90013566e+00 -8.47574591e+00  1.46588430e+01
 -7.06115901e+00]
```

The audio signal was first outputted with amplitude vs. time. Then it was plotted after silence removal, normalization,

The code and output for the entire dataset is given below:

```
import os
import librosa
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
# Define the path to your dataset folder
dataset_path = "D:\Dil\S.P.I.T Semester 6\Mini project\dataset"
# Define the categories and the corresponding number of audio files
categories = ["belly_pain", "burping", "discomfort", "hungry", "tired"]
num_files = [16, 8, 27, 382, 24]
# Define the parameters for MFCC extraction
n_mfcc = 13
pre_emphasis = 0.97
# Define the classifiers to evaluate
classifiers = [GaussianNB(), RandomForestClassifier(n_estimators=100),
KNeighborsClassifier(n_neighbors=3)]
# Define the lists to store the mean MFCC features and labels for each
category
mfcc_features = []
labels = []
# Loop through each category folder and extract the mean MFCC features
for each audio file
for i, category in enumerate(categories):
 for j in range(num_files[i]):
  # Load audio file
  audio_file = os.path.join(dataset_path, category, f"{category} ({j+1}
).wav")
  signal, sr = librosa.load(audio_file, sr=16000)
  # Remove leading and trailing silence
  signal, _ = librosa.effects.trim(signal)
  # Normalize the signal
  signal = librosa.util.normalize(signal)
  # Apply pre-emphasis
  emphasized_signal = np.append(signal[0], signal[1:] - pre_emphasis *
signal[:-1])
  # Apply Hamming window
  hamming_window = np.hamming(len(emphasized_signal))
  emphasized_signal = emphasized_signal * hamming_window
  # Compute short-time zero crossing rate and short-time energy
  frame_length = 0.025 # seconds
  frame_stride = 0.01 # seconds
  hop_length = int(frame_stride * sr)
  win_length = int(frame_length * sr)
```

```
  zero_crossings = librosa.feature.zero_crossing_rate(emphasized_signal
, pad=False, frame_length=win_length, hop_length=hop_length).sum()
  energy = np.sum(np.power(emphasized_signal, 2))
  # Compute MFCC features
  mfcc = librosa.feature.mfcc(y=emphasized_signal, sr=sr, n_mfcc=n_mfcc
)
  # Compute the mean of the MFCC features, zero crossing rate and energ
y
  mfcc_mean = np.mean(mfcc, axis=1)
  features = np.concatenate([mfcc_mean, [zero_crossings, energy]])
  # Append the features and the label to the corresponding lists
  mfcc_features.append(features)
  labels.append(category)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(mfcc_features, labe
ls, test_size=0.2,
random_state=42)
# Train and evaluate each classifier
best_acc = 0
best_classifier = None
for clf in classifiers:
 clf.fit(X_train, y_train)
 y_pred = clf.predict(X_test)
 acc = accuracy_score(y_test, y_pred)
 print(f"Accuracy for {type(clf).__name__}: {acc}")
 if acc > best_acc:
 best_acc = acc
 best_classifier = clf
# Use the best classifier to predict the category of a new audio file
new_audio_file = "D:\Dil\S.P.I.T Semester 6\Mini project\dataset\hungry
\hungry (12).wav"
signal, sr = librosa.load(new_audio_file, sr=16000)
# Remove leading and trailing silence
signal, _ = librosa.effects.trim(signal)
# Normalize the signal
signal = librosa.util.normalize(signal)
# Apply pre-emphasis
emphasized_signal = np.append(signal[0], signal[1:] - pre_emphasis * si
gnal[:-1])
# Apply Hamming window
hamming_window = np.hamming(len(emphasized_signal))
emphasized_signal = emphasized_signal * hamming_window
# Compute short-time zero crossing rate and short-time energy
frame_length = 0.025 # seconds
frame_stride = 0.01 # seconds
hop_length = int(frame_stride * sr)
win_length = int(frame_length * sr)
```

```
zero_crossings = librosa.feature.zero_crossing_rate(emphasized_signal,
pad=False,
frame_length=win_length, hop_length=hop_length).sum()
energy = np.sum(np.power(emphasized_signal, 2))
# Compute MFCC features
mfcc = librosa.feature.mfcc(y=emphasized_signal, sr=sr, n_mfcc=n_mfcc)
# Compute the mean of the MFCC features, zero crossing rate and energy
mfcc_mean = np.mean(mfcc, axis=1)
features = np.concatenate([mfcc_mean, [zero_crossings, energy]])
# Use the best classifier to predict the category of the new audio file
predicted_category = best_classifier.predict([features])[0]
print(f"Predicted category: {predicted_category}")
```

Output:

```
Accuracy for GaussianNB: 0.7391304347826086
Accuracy for RandomForestClassifier: 0.782608695652174
Accuracy for KNeighborsClassifier: 0.7391304347826086
Predicted category: hungry
```

Chapter 6
Experimental Analysis and Result

The steps that were done for a single audio file have been reported in the nested for loop. After that, the machine learning part begins:

• X_train, X_test, y_train, y_test = train_test_split(mfcc_features , labels, test_size=0.2, random_state=42): In this part, the dataset after the feature extraction has been split into training and testing part. T 80% of the data is taken as the training part and 20% of the data is taken as testing part.

• Next each algorithm has been used for training ML models. We also put the best classifier into the variable best_classifier. In this case it was the random forest classifier. In order of accuracy, the models are: Random forest classifier with an accuracy of 78.26%, Gaussian Naive Bayes and K-Nearest Neighbours both with an accuracy of 73.913%.

• We applied feature extraction and pre-processing on a sample audio file (hungry (12).wav) and we applied the best classifier on it. It predicted the correct output.

Chapter 7
Conclusion

In this project, we applied machine learning algorithms to predict the reason for infant cry after applying preprocessing and feature extraction techniques. Each technique was selected after careful consideration and has been used to get the most accurate output. The pre-processing steps included:

1. Normalization
2. Pre-emphasis
3. Windowing

The feature extraction techniques included:

1. ZCR
2. STE
3. MFCC

The machine learning models applied were:

1. Random forest regression
2. KNN
3. Naive Bayes

Overall, the project focused on figuring out the best techniques that can be used on a dataset with a limited amount of data. In the future, we wish to expand our dataset further in hopes of improving the accuracy of the models used. We also wish to extract more features instead of just the techniques that we have used including techniques such as Linear Prediction Coefficients (LPC), Perceptual Linear Prediction (PLP), and Harmonic Pitch Class Profile (HPCP). We also wish to implement more ML models, especially models such as SVM. We also wish to implement modern deep learning techniques that have been evolving every year. We wish to attempt all of this in hopes of increasing the accuracy of our model in hopes of creating a proper infant cry analyzer that can actually be implemented in real life.