A comprehensive review of Acoustic deployable machine learning model

by
Diya Khurdiya
Ashoka ID: 1020201649
BS Computer Science (Major)
Ashoka University
Haryana, India

Mentor
Rintu Kutum, Ph.D.
Computational Health Science Lab Faculty Fellow,
Department of Computer Science Faculty, Trivedi School of Biosciences
Data Scientist, Mphasis Lab for ML Computational Thinking
Ashoka University
Sonepat, Haryana - 131029
2023

Contents

1 Abstract 3

2 Introduction 4

3 The Data set 5

4 Audio Data Pre-processing 6

5.1 Audio visualization 6

5.2 Spectrogram 7

5.3 Feature Extraction 9

6 Noise Reduction 12

7 Data Augmentation 14

7.1 Introduction 14

7.2 Noise Addition and implementation 15

8 Logisitic Regression 16

8.1 Introduction 16

8.2 Implementation 16

8.3 Results 17

9 Appendix 19

10 References 20

Abstract:

This report presents a comprehensive analysis of acoustic deployable audio classification model.Audio classification and analysis play a vital role in numerous fields, including speech recognition, music genre identification, environmental sound monitoring, and more. The aim of this report is to explore the field of audio classification and develop a deeper understanding of how the logistic regression works.

Introduction:

Audio classification and analysis is a fairly new concept in machine learning which has not been completely exploited. Audio analysis is a process of transforming, exploring, and interpreting audio signals recorded by digital devices. It comes in varied forms like classifying music clips to identify the genre of the music, or classifying short utterances by a set of speakers to identify the speaker based on the voice.

Audio data represents analog sounds in a digital form, preserving the main properties of the original audio. Sound wave has charactertisitcs like frequency, time period and amplitude. Audio frequency between 20Hz - 20,000Hz is audible to a human ear.
We would be using **.WAV** format for storing audio data as it doesn't compress audio quality.

The process that follows audio classification is:

1. Audio data is loaded and pre-processed by plotting frequency-time graphs and also various feature extraction methods are done to get a better sense of our data.
2. Audio data needs to be converted to a spectrogram (a visual representation of) to be able to input them to a machine learning model.
3. A CNN or logistic regression model is trained and tested, for predictions and metrics like accuracy and precision score are noted.

We will also look at some noise reduction techniques like high pass filtering, spectral subtraction etc. as audio data in real life is often intermixed with random disturbances. As the signal transmits through the channel, it faces some random disturbances due to environment or during analog to digital conversion process which is normally distributed at the time.

At last, we will be understanding logistic regression and it's implementation on the dataset.

The Data set

The audio dataset used in the report comprises of two folders tb_0 and tb_1 taken from the CODA code challenge. The tb_0 folder consists of **4250** audio files that are **TB negative** whereas tb_1 consists of **2250 TB positive** files.
Each audio recording is less than a second and belons to either of the two categories.
The audio files is already pre-processed and there is negligible external noise.

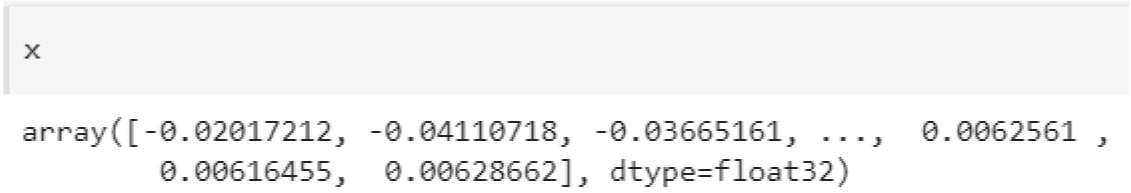
Audio Data Pre-processing:

An important library that supports audio and music analysis is Librosa and we would be using it extensively for feature extraction.
Before feature extraction, we will load and visualize audio in the form of a waveform for both tb_0 and tb_1.
A waveform is a basic visual representation of an audio signal that reflects how an amplitude changes over time. The graph displays the time on the horizontal (X) axis and the amplitude on the vertical (Y) axis.

1. Audio visualization

While loading the audio data, there are two important parameters that will be used extensively here.

1. **Sample Rate**: Samples are recorded per second. The default sampling rate and which is also used in this project is 22050 as librosa tries to normalize the audio and gives it a single sample rate.
2. **Numpy Array:** It is a floating point time series loaded as 1-D floating point array, which represents the amplitude of the waveform at a sample time t.



```
x
```

```
array([-0.02017212, -0.04110718, -0.03665161, ...,  0.0062561 ,
        0.00616455,  0.00628662], dtype=float32)
```

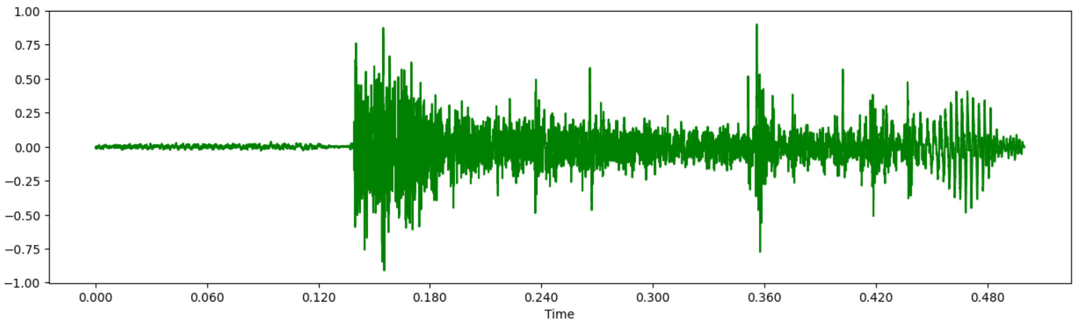


Fig. 1. Tuberculsosis positive

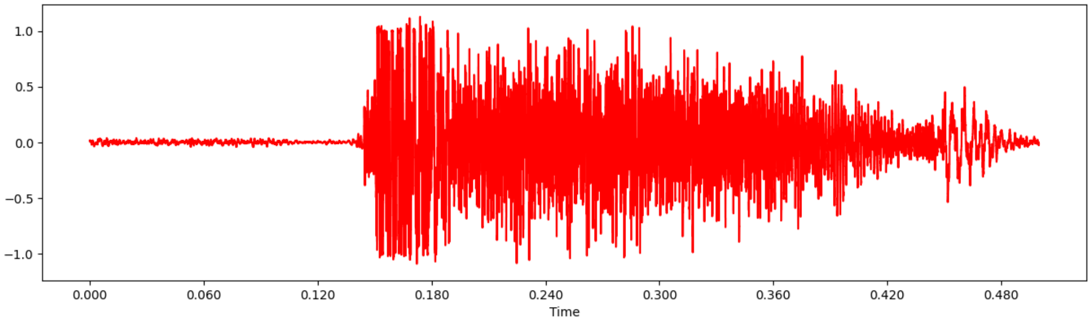


Fig. 2. Tuberculosis negative

2. Spectrogram

Audio files needs to be converted to spectrogram to be able to feed it as an input for machine learning model. A **spectrogram** is a visual representation of the spectrum of frequencies of a signal as it varies with time. They are time-frequency portraits of signals. This is possible because every signal can be decomposed into a set of sine and cosine waves that add up to the original signal. This is a remarkable theorem known as Fourier's theorem.

The **short-time Fourier transform (STFT)** is a sequence of Fourier transforms converting a waveform into a spectrogram.

We use **X = librosa.stft()** represents a signal in the time-frequency domain by computing discrete Fourier transforms (DFT) over short overlapping windows.

The function returns a complex valued matrix with **np.abs(X)** giving us the magnitude of frequence bin f at time t.

librosa.amplitude_to_db(abs(X)) converts fourier transform values to energy levels.

Then using librosa.display.specshow, we can get a spectrogram as shown in Fig. 3.

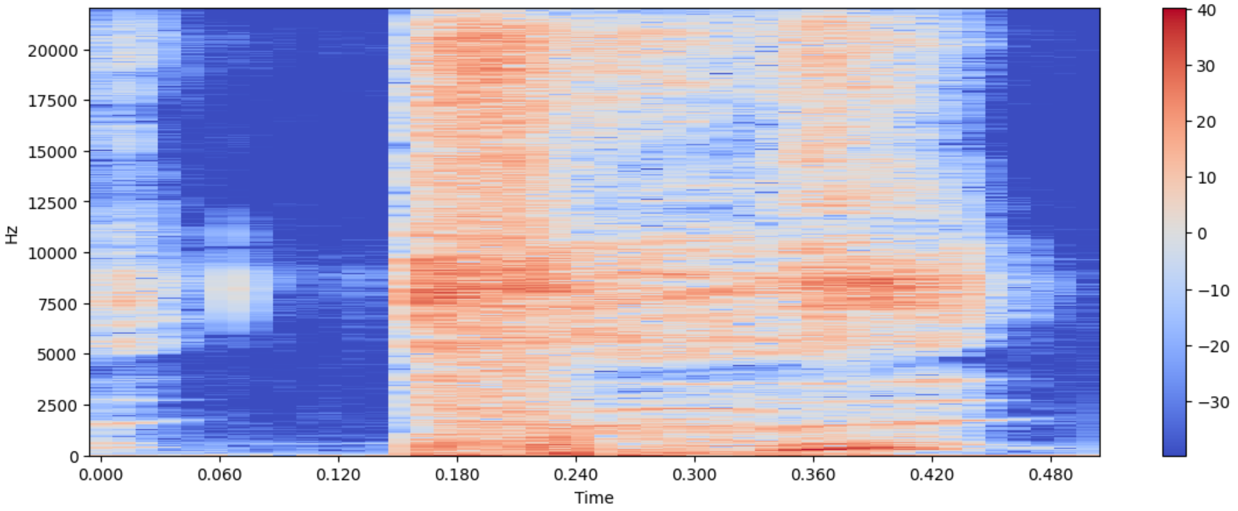


Fig. 3 Spectrogram of an audio file

A mel spectrogram is computed by converting linear frequency axis to a log frequency axis. The mel scale measures equal distances in pitch that sounds equally distant to the listener. This is the mel spectrogram of our audio input and the higher intensity color shows higher signal amplitude.

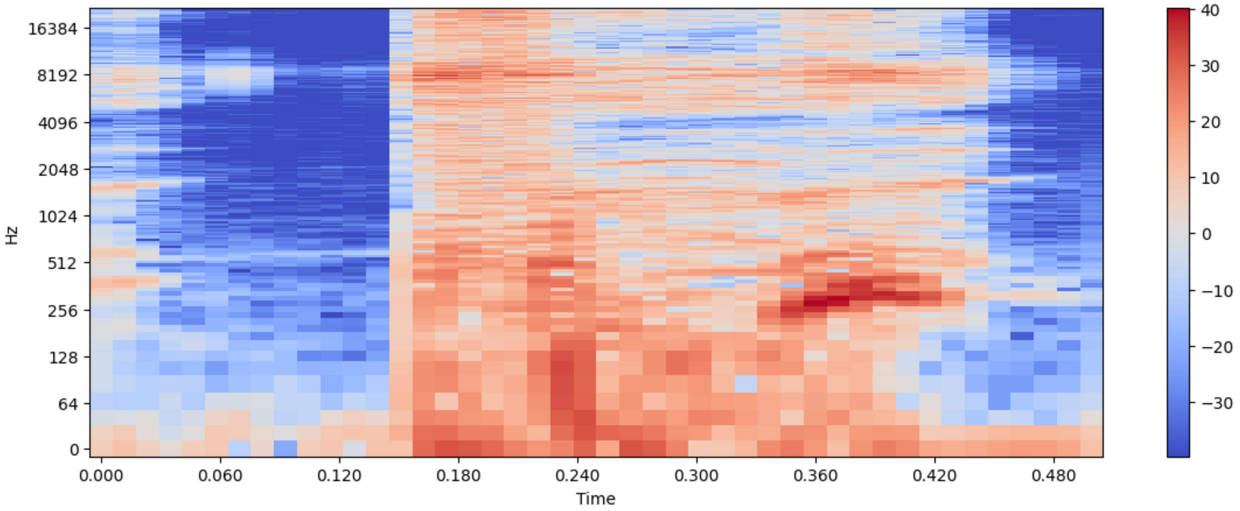


Fig. 4. Mel spectrogram of the previous spectrogram in Fig. 3.

3. Feature Extraction

The **Fourier transform (FT)** is a mathematical function that breaks a signal into spikes of different amplitudes and frequencies. We use it to convert waveforms into corresponding spectrum plots to look at the same signal from a different angle and perform frequency analysis. The **Fast Fourier Transform (FFT)** is the algorithm computing the Fourier transform.

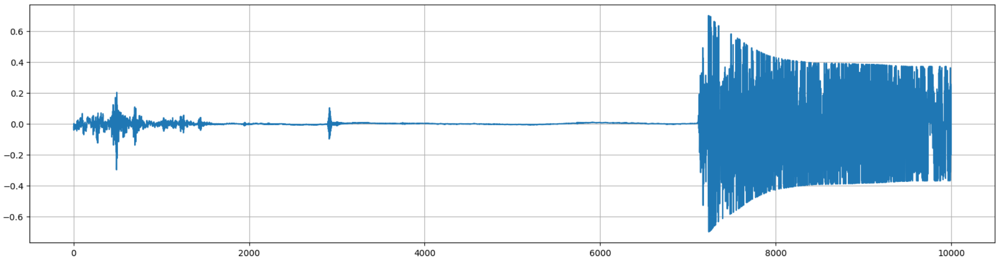
Audio features or descriptors are properties of signals, computed from visualizations of preprocessed audio data. They can belong to one of three domains

- time domain represented by waveforms,
- frequency domain represented by spectrum plots, and
- time and frequency domain represented by spectrograms.

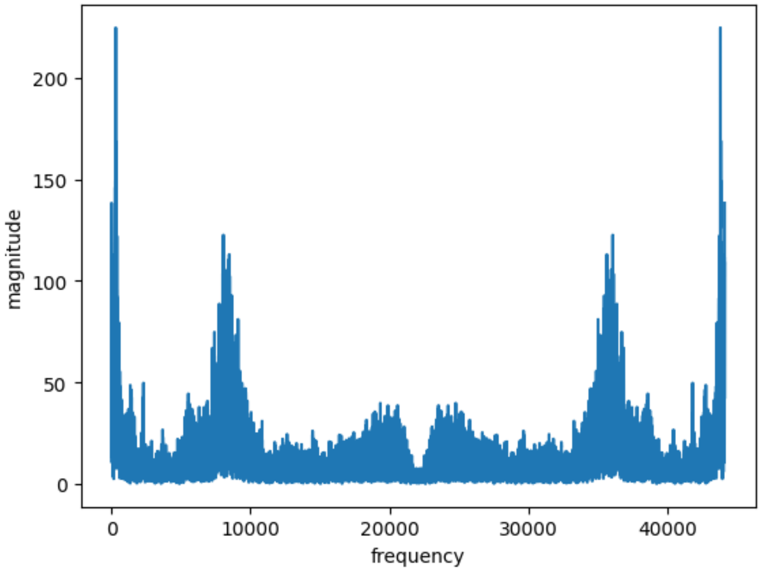
Time domain or temporal features are extracted directly from original waveforms, they indicate only how the amplitude changes with time.

Some features that we have experimented with here are:

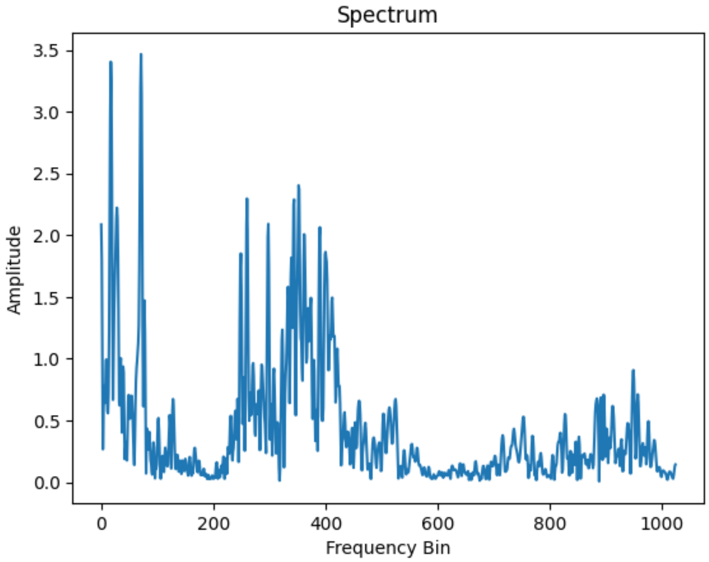
1. **Zero-crossing Rate (ZCR)** counts how many times the signal wave crosses the horizontal axis within a frame. It's one of the most important acoustic features, widely used to detect the presence or absence of speech, and differentiate noise from silence and music from speech.



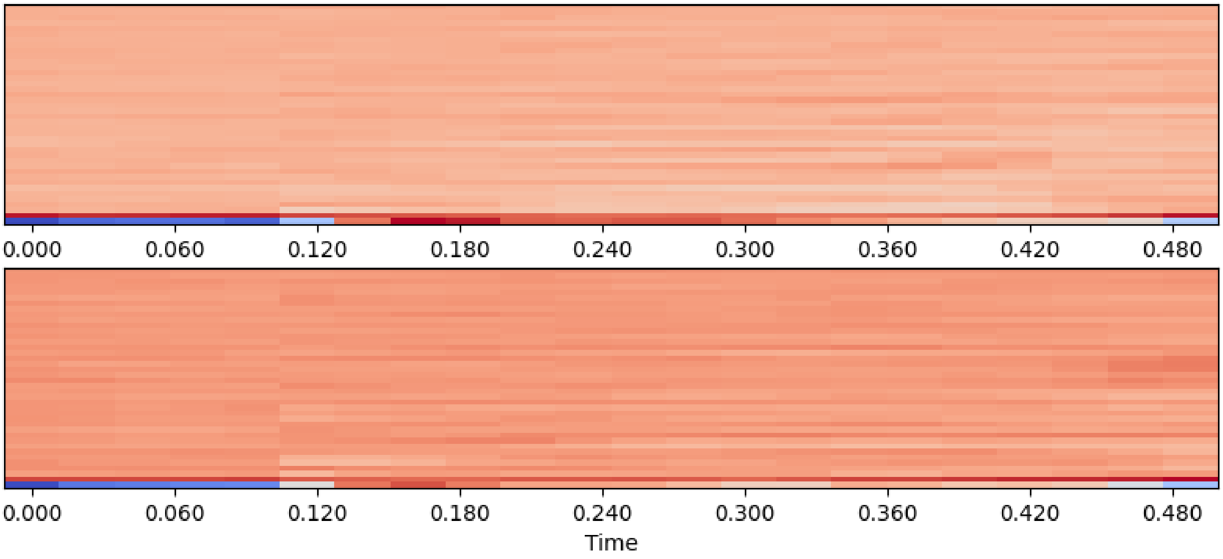
2. **Power Spectrum** applies a **fast Fourier transform (FFT)** to the variation of a particular signal to compute its frequency spectrum. The result is presented as a plot of signal power against frequency and is referred to as its power spectrum. The power spectrum of a signal indicates the relative magnitudes of the frequency components that combine to make up the signal.



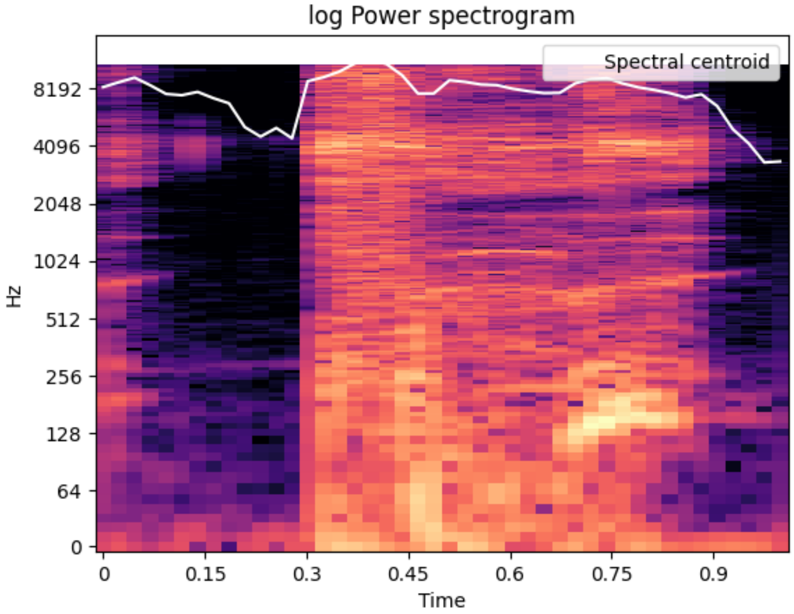
3. **Amplitude Spectrum** simply gives amplitude at each frequency. In other words, it is the vector that contains the absolute values (or moduli) of the coefficients of the frequency-domain representation.



4. **MFCC (mel frequency cepstral coefficients)** is a compact representation of the spectrum and MFCC coefficients contain information about the rate changes in the different spectrum bands.



5. **Spectral centroid** a measure of the amplitude at the center of the spectrum of the signal distribution over a window calculated from the Fourier transform frequency and amplitude information. Each frame of a magnitude spectrogram is normalized and treated as a distribution over frequency bins, from which the mean (centroid) is extracted per frame.



Noise Reduction

It is important to reduce the attenuation produces by random noise and improve the performance of the signal.
When the signal transmits through the channel, it faces some random disturbances due to environment or during analog to digital conversion process which is normally distributed at the time termed as AWGN (additive white Gaussian noise). This noise added to the original information signal and produces the errors in the information. The noise distorts the information signal and lowers its quality.

There are various types of noise that are often introduced into the input data.

- Continuous noise
- Intermittent noise
- Impulsive noise
- Low-frequency noise

Some ML learning algorithms that are used for noise removal:

1. **Recurrent neural networks** are models that can recognize and understand sequential data. Sequential data includes things like audio, text, or the position of an object over time. RNNs are particularly effective for background noise removal because they can learn patterns across time which is essential for understanding audio.
2. **Active noise control (ANC)** methods are based on adaptive signal processing with the least mean square algorithm as the foundation. The deep ANC method can be trained to achieve active noise cancellation no matter whether the reference signal is noise or noisy speech. Effective for wideband noise reduction and generalizes well to untrained noises
3. **Wavelet Transform (WT)** is a powerful tool for removal of noise from various signals. It decomposes the input signal into multiple scales, which represent different space-frequency components of the original signal. At each scale, some operations, such as thresholding and statistical modeling, can be performed to suppress noise. Denoising is accomplished by transforming back the processed wavelet coefficients into spatial domain. These methods known as wavelet-based denoising techniques can be viewed also as fixed basis dictionaries to whole images.

4. **Auto-encoders** are useful for the de-noising purpose. As it is possible to make them learn the distinctive noise from the signal or data, thus they can be served as de-noisers by providing the noisy data as input and getting clean data as output. Auto-encoders contains two parts: one is an encoder that encodes input data into encoded form, second is a decoder, which decodes the encoded state.
5. **Filters** are proposed to reduce the attenuation level of signal and get the desired signal. The filter is a device which shapes the signal waveform in a desired manner. The main purpose of filters in digital signal processing is to reduce the noise which improves the performance of the signal and to extract the desired information from the signal. There are many different types of filters that are used to reduce the effects of noise. Like a low pass filter which takes the lower frequencies and rejects the higher frequencies. A high pass filter passes the signal above a cut-off frequency.

High pass filtering method

An audio high pass filter (HPF) is an equalization tool that attenuates all frequencies below a set point. In other words, high pass filters remove low frequencies while allowing high frequencies to pass through. They are used in almost every application of audio technology and can help in removing extraneous low frequencies, removing environmental noises and shaping tone for mix clarity.

Data Augmentation

1. Introduction

Data augmentation is a technique to diversify your audio dataset by modifying the existing samples or adding small perturbations. The objective is to make our model invariant to those perturbations and enhace its ability to generalize. But pertubations conserve the same label as original sample.

There are two types of data augmentation for audio data:

1. **Augmentations for waveform:** Applied to 1-D signal and we can add noise, shift time of the audio signal, shift the pitch of the signal, time stretch randomly slow down or speed up etc.
2. **Augmentations for spectrogram/melspectrogram**: Using a method known as specAugment,
    a. Frequency mask — randomly mask out a range of consecutive frequencies by adding horizontal bars on the spectrogram.
    b. Time mask — similar to frequency masks, except that we randomly block out ranges of time from the spectrogram by using vertical bars.

We would be implementing noise injection using code for gaussian noise (white noise).

Gaussian noise addition
np.random.normal() draws out random normal samples from the normal distribution of the signal and taking mean = 0 and standard deviation as signal.std() i.e. signal's standard deviation and signal.size() which is the number of samples drawn.
Then we simply add the original signal in the form of a numpy array to a noise factor which is calculated by multiplying the noise factor as specified by the user and random sample drawn. We then pass it on as a function to see the variations as the noise factor increases.

2. Noise Addition and implementation

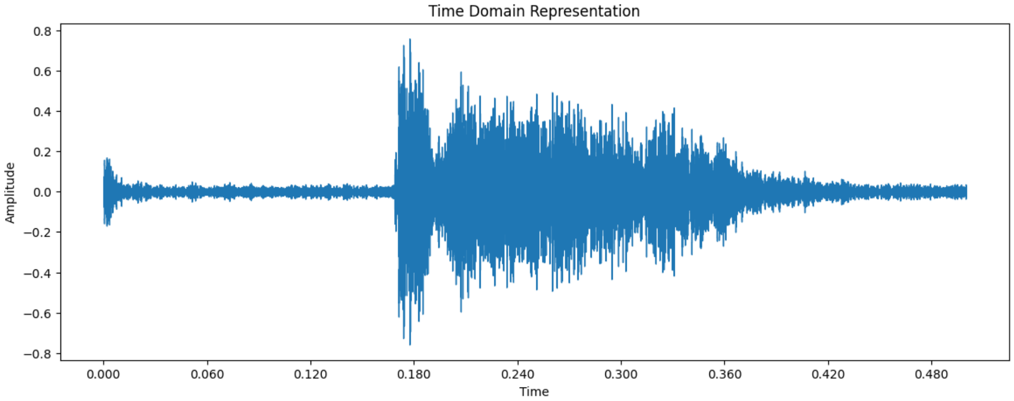


Fig. 8 Original audio sample

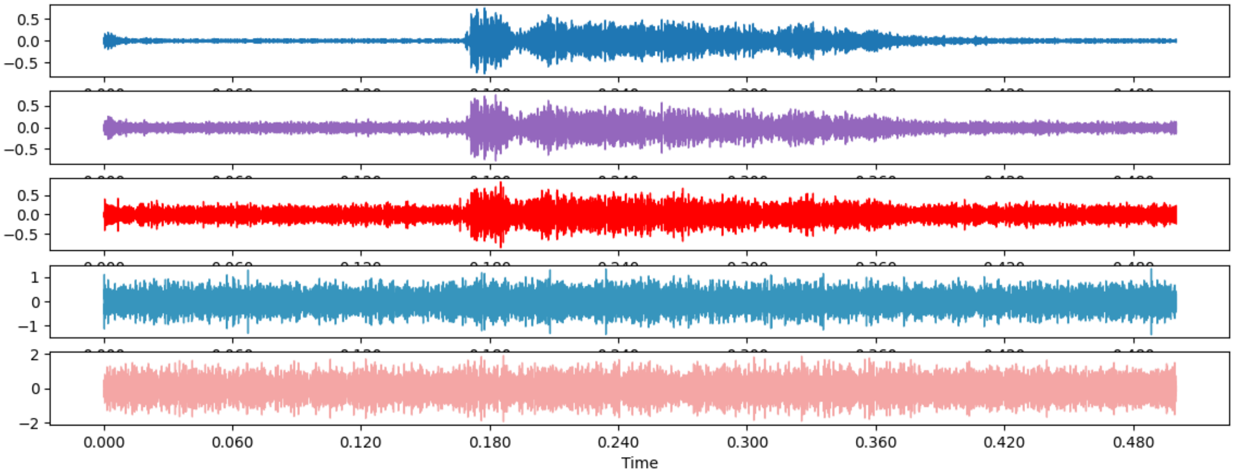


Fig. 9. Audio samples after adding noise of factor 0.1, 0.5, 1, 3, 5 respectively

Logisitic Regression

1. Introduction

Logistic regression is a regression analysis that predicts the probability of an outcome that can only have two values (i.e. a dichotomy). A logistic regression produces a logistic curve, which is limited to values between 0 and 1. Logistic regression models the probability that each input belongs to a particular category.

Math behind logistic regression:
The logsiitic function most commonly called as sigmoid function depicted by the formula shown below,

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}}$$

Where e is the Euler’s constant i.e. 2.718, x_0 is the value of sigmoid’s midpoint on the x - axis, L is the maximum value and k is the steepness of the curve.
In general, logistic function is depicted by,

$$z = \Theta^T x$$

$$sigmoid(z) = \frac{1}{1 + e^{-(z)}}$$

Cross-entropy loss function
Loss functions are implemented to model accuracy and for better optimization of the machine learning model. A loss function operates on the error to quantify how bad it is to get an error of a particular size/direction, which is affected by the negative consequences that result in an incorrect prediction. the entropy of a random variable is the average level of “information,” “surprise,” or “uncertainty” inherent in the variable’s possible outcomes.

Cross entrtopy is the measure of the difference between two probability distributions for a given random variable or set of events

$$\boxed{-\sum_{n=i}^{c} t_i \ \log(f(s_i)) + \ (1 \ - t_i)\log(1 - f(s_i))}$$

where Si - inputs/weights, f - the activation function in this case, t - the target predictions, i - the class to predict.

Gradient descent:

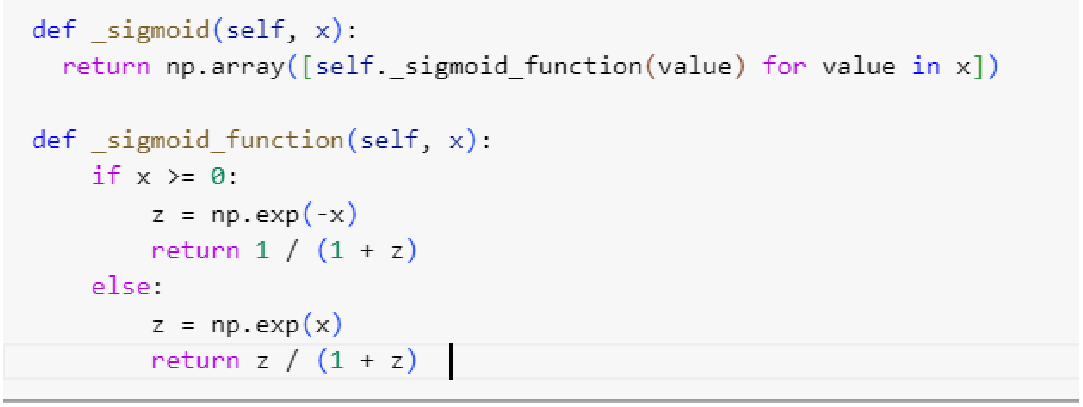
1. It is an optimization algorithm that helps models find the optimal set of parameters by iteratively adjusting them in the opposite direction of the gradient.
2. Here we would need cost function, that measures the performance of a model for any given data and basically quantifies the error between predicted values and expected values and presents it in the form of a single real number.
3. The goal is to reduce the cost function by modifying the parameters using the gradient descent algorithm.
4. The gradient is the slope of the cost function, and by moving in the direction of the negative gradient, the algorithm can converge to the optimal set of parameters that best fits the training data.

2. Implementation

The steps that we would be following are:

1. Initialize the model's weights to zero or small random values.
2. Feed the training data into the model to get the predicted outputs (class probabilities) using the logistic function:

   y_hat = sigmoid(w * x + b)

   ```
   def _sigmoid(self, x):
     return np.array([self._sigmoid_function(value) for value in x])

   def _sigmoid_function(self, x):
       if x >= 0:
           z = np.exp(-x)
           return 1 / (1 + z)
       else:
           z = np.exp(x)
           return z / (1 + z)
   ```

   where x is the input data, w is the weight vector, b is the bias, and sigmoid is the logistic function.
3. Compute the loss between the predicted outputs and the actual outputs using a loss function, such as the binary cross-entropy loss:

   L = -1/m * sum(y * log(y_hat) + (1 - y) * log(1 - y_hat))

   where m is the number of training examples, y is the actual output (0 or 1), and log is the natural logarithm.

4. Compute the gradients of the loss with respect to the weights and bias using backpropagation:

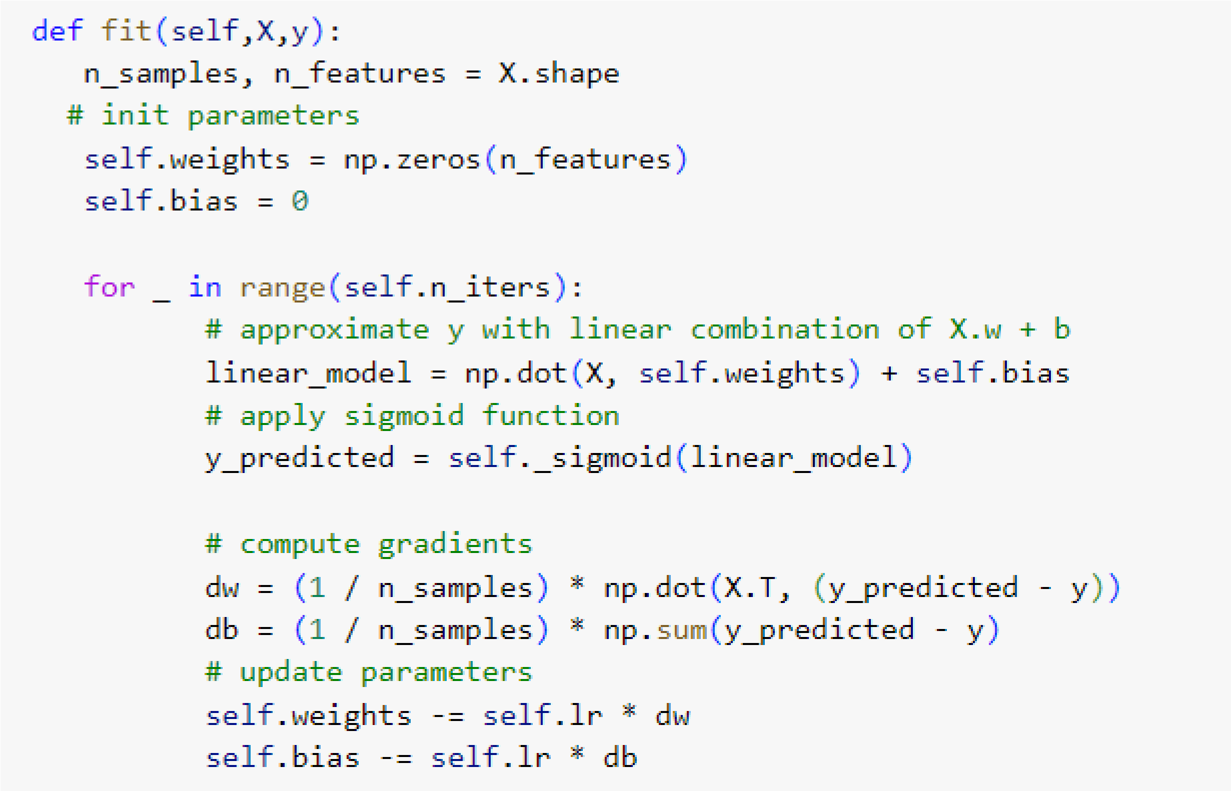
   dw = 1/m * dot(x.T, (y_hat - y)) db = 1/m * sum(y_hat - y)

   where dot is the dot product, T is the transpose, and sum is the sum.

5. Update the weights and bias using gradient descent:

   w = w - learning_rate * dw b = b - learning_rate * db

   where learning_rate is a hyperparameter that controls the step size of the update.



```
def fit(self,X,y):
   n_samples, n_features = X.shape
  # init parameters
   self.weights = np.zeros(n_features)
   self.bias = 0

   for _ in range(self.n_iters):
          # approximate y with linear combination of X.w + b
          linear_model = np.dot(X, self.weights) + self.bias
          # apply sigmoid function
          y_predicted = self._sigmoid(linear_model)

          # compute gradients
          dw = (1 / n_samples) * np.dot(X.T, (y_predicted - y))
          db = (1 / n_samples) * np.sum(y_predicted - y)
          # update parameters
          self.weights -= self.lr * dw
          self.bias -= self.lr * db
```

6. Use the trained model to make predictions on new input data by applying the logistic function to the input data using the learned weights and bias:

   y_hat = sigmoid(w * x + b)

   where x is the input data and w and b are the learned weights and bias. The predicted output is the class with the highest probability (usually rounded to 0 or 1 for binary classification).

   The accuracy score is approximately 83% and is very less than what we would obtain using sklearn library.

   I hope to further deep-dive into this algorithm and attempt to fix the various shortcomings it has, or develop this into a more robust algorithm.

Appendix

1. Report1
2. Report2
3. Presentation (Transformers)

References:

1. https://towardsdatascience.com/audio-deep-learning-made-simple-sound-classification-step-by-step-cebc936bbe5
2. https://www.sciencedirect.com/topics/engineering/amplitude-spectrum
3. https://www.kaggle.com/code/jagannathrk/logistic-regression-from-scratch-python
4. https://arxiv.org/pdf/2107.05463.pdf
5. https://www.linkedin.com/pulse/logistic-regression-from-scratch-python-shahiryar-saleem/
6. https://developer.ibm.com/articles/implementing-logistic-regression-from-scratch-in-python/
7. https://www.altexsoft.com/blog/audio-analysis/