EE514/CS353: Machine Learning Project Report

Session: Spring 2022

- ☐ **Abdul Hannan Anjum Chaudhry** 2023-11-0058
- ☐ **Muhammad Hamza** 2023-11-0359
- ☐ **Zeeshan Ashraf** 2023-02-0083

Department of Electrical Engineering

Syed Babar Ali School of Science and Engineering **Lahore University of Management Sciences Pakistan**

Abstract

In the current geopolitical climate, deep fake media such as videos and audios become ever increasingly harder to detect and can pose a serious threat to personal, national and international security. The basic steps of Speech recognition comprise of pre-processing, feature selection, feature extraction, model building and classification. A lot of preliminary work has been published for improvement of these stages and to achieve excellent results. Our approach consists of a mixture of theoretical and experimental methods, where we, through brute force, compare different approaches to audio feature extraction and selection like different combinations of Mel-frequency cepstral coefficients, Chroma Feature (time-chroma representation), Spectral Centroid and afterwards we compare the efficiency of different machine learning algorithms like Naive Bayes, SVM, Logistic regression, Neural Network and K-Nearest Neighbor classifier in order to conclusively find the best model. After model training and testing based on a 80/20 split, we found that Support Vector Machines perform the best in terms of precision, even though K Nearest Neighbor algorithm performs slightly better in terms of accuracy. Furthermore, in the dataset presented to us, we find that classes 1 and 2 were the greatest inhibitors in reaching a near perfect or close to 99% accuracy, so we propose further analyzing in future of the algorithms used to generate them to find which audio feature could be used to make the machine learning model even more effective.

Synthetic Speech Algorithm Detection

1.1 Introduction and Literature Overview:

Over the years, different approaches and methods have been identified for synthetic speech recognition and audio detection. With the novelty of these machine learning methods in place, practical results are now achievable very easily. The usage of devices that have these speech recognition features installed has become more common than ever, where we can interact or control these devices and systems like the virtual assistants (Siri), Smart home control devices, to real-time speech translation. Similarly, noise robust speech recognition has also become an integral part of human-machine Interface interaction systems. These systems give rise to the techniques used in Machine Learning. The reality is that everyone can manipulate these devices given the advancement in technology and the accessibility of it. Research has revealed that these highly efficient and complex algorithms can achieve very high accuracy in distinguishing voices that even humans find difficult to distinguish.

To be able to communicate is an important aspect of human behaviour. The most common format of human interaction is verbal, where the vocalized form of verbal communication is also referred as Speech. This vocalized form of natural language speech makes it possible for humans to communicate effectively. It makes way for speech recognition model development whereby machines understand the meaning of human speech. This is predominantly an active research area which focuses on translation of words into written scripts through use of speech recognition. Automated speech recognition (ASR) systems make it feasible to convert respective natural language segments of input speech into respective units of text. Perhaps, various applications of man machine interaction and speed-based applications are demonstrated using these ASR models like communication interfaces for disabled people with hearing problems, translation devices, dictation systems and voice-mail

systems in telephony. Therefore, numerous researchers have conducted extensive work to contribute to this area.

The analysis of speech synthesis reveals that a variety of other things are also part of these audio speeches like silence intervals, ambient noise, and other non-vocal content. More importantly during the feature extraction process, it is important that they should be extracted from intervals consisting of actual speech data so that relevant information can be captured by the neural network. Analysis of these features needs to be done cautiously as these sensory features significantly affect our algorithm and recognition performances. Therefore, the first step is voice activity detection (VAD), within this speech-processing pipeline (accurate labelling of content in the audio within time intervals).

As per the research, features that are more prevalent in audio or speech signals includes energy, zero-crossing rate (ZCR), the harmonic distribution, Fast Fourier, noise ratio, and perceptual linear predictions or Mel-Frequency Cepstral Coefficients (MFCCs), Short-Term-Fourier-Transformation (STFT), Mel Spectrogram, and Constant-Q transform (CQT), along with other different spectral features and time derivatives. To use these features, different machine learning based classifiers have been documented, using frequency analysis for example, Gaussian mixture models (GMMs), Logistic Multi Linear Regression, Kalman filters, neural networks (multilayer perceptron (MLPs)), recurrent neural networks (RNNs) using long short-term memory (LSTM) cells. There are a variety of tasks that can be solved in this domain of audio signalling and speech recognition, the majority of which have novel machine learning procedures. As literature review suggest, nowadays, Support Vector Machine along with convolutional neural networks (CNNs) are being used for automatic feature Extraction using time and frequency domain signals and contextual time modelling, like discrete Fourier transform (DFT) or spectrograms (Borrelli et al.) can outperform other existing deep learning models (Zhang et al). Similarly, several hybrid CNN models have shown affirmative
results but they have significantly high cost.

Recent studies have shown that use of Deep Neural Networks for speech recognition allows us to incorporate large chunks of audio data to generate a model, which can produce utterances on unseen data very accurately (Awad). Our research suggests that new deep learning mechanisms are also being used for feature extraction like the use of unsupervised machine learning procedures to learn fused representations over different modalities, which helps to increase accuracy of speech recognition process. Deep denoising auto-encoder present in neural networks trains the network for precise prediction of original clean audio features like MFCCs from the noise signals, which are synthetic ones. The acquired features are processed using the existing hidden Markov model (HMM) with Gaussian process so that noise-robust audio features can easily be extracted using this mechanism. However, DNN (deep neural network) based systems can produce more natural utterances than HMM or GMM ones. These frequency analysis procedures i.e GMM and HMM provides reasonable classification results but they often ignore the temporal utterances that are pivotal in detecting synthetic speech. For that reason, DNN is the best way to go about, as it not only considers frequency features but also contextual time modelling. These DNN are good feature extractors as they use unsupervised learning to learn both frequency and temporal signals. It extracts all the features like (STFT, MFCC, Mel-Spectrogram) after each utterance and inserts them into DNN architecture to compare which of these structures performs better.

In this paper, our focus is on classification and identification of speech algorithms that have been used to generate synthetic audio given the audio signals in our augmented dataset and clean dataset. The classifiers used for the algorithm includes logistic regression, K-Nearest Neighbor, Support Vector Machines, Neural Networks and Logistic Regression. The rationale behind it is to detect fraud in speeches within our audio anti spoofing community.We look primarily at MFCCs, Spectral Centroid and Chromagram and contrast to understand why they complement with each other. For validation of our method, our analysis was initially performed in a data set that contains speech tracks from multiple different speech synthetics techniques ranging from traditional ones (wave concatenation etc) to novel based like CNN methods. It

starts of with the methodology used in proposed work and comparison results obtained by various researchers using different classifiers. Our research paper contents are then followed by illustrations of our proposed method (like speech processing techniques) and review on use of Naive Bayes, Neural Networks, SVM, KNN and Logistic Regression for speech recognition systems starting from feature extraction and engineering to classification followed by Mathematical models used in the algorithm. After that, our focus shifts towards the evaluation process, the accuracy of the method (achieved results) and the simulation results obtained using proposed work via breakdown of our experimental design process. Finally, our research paper would pave the way for future research questions that stem from our proposed method and end with a conclusion.

The three main key stages in this process are pre-processing, feature extraction and classification of basic Automatic Speech Recognition systems.

Pre-Processing:

Background noise filtering, optimizing audio signals and blocking are the main tasks. Initially the dataset is imported using a library and then these analog audio signals are converted to discrete value or time signal through the Digital conversion process. Quantization (Scaling) and sampling are two techniques used for this process. Using a realistic approach an 8 to 20 kHz sampling rate is used and about 10 kHz of frequency occur in speech as
indicated by previous studies; however, within this considerable limit, speech remains intelligible. The Quantization factor takes into consideration the type of scaling process used to determine intensity of signal as discrete number. Sufficient information can be captured by 11-bit numbers although only 8-bit numbers are achieved using log scale. Perhaps, 256 unique categories are used to classify each segment of signal. Therefore, a large chunk of data of 8-bit numbers, speech signals or 10,000 numbers per second is a difficult task. The scaling of data into some number representation is a challenge for speech recognition systems. Moreover, to maintain high SNR, background
noise is filtered after the signal conversion process.

Presence of background noise in the audio signals is a major concern as it can also be quantized along with speech data. This is an end point detection problem where the amount of processing is kept at minimum as to accurately begin and end point of audio signals. The next step of the process involves Pre-emphasis that emphasizes important frequency components of audio
track by spectrally flattening the signal.

1.2 Feature Extraction and Selection

1.2.1 Feature Extraction

Feature engineering involves determining the set of properties of the utterance with acoustic correlation properties. It helps us evaluate the feature parameters which can be computed via processing of signal waveform. Parameterization of high frequency waves means processing of frequency or energy response like important characteristics of a signal. Mel Frequency Cepstral Coefficients (MFCC), Chromagram, Spectral centroid and as feature extraction processes are some of the ways of extracting features.

Through literature review, we have identified the different feature extractions and the methods we will be using to select these features. Main idea behind these is to first observe such features, which are present in our audio-based data and then see which features are useful to us and are not redundant. This is an important aspect because it will improve the learning rate, minimize computational complexity, and optimize storage use. Here we will discuss some feature extraction methods.

1) Mel frequency cepstral coefficients (MFCCs): It is a widely used feature extraction method for speech recognition. It uses a different spectrum of voice signals and deals with human perception analysis, which considers different frequencies to recognize different speeches. Mel-frequency cepstrum (MFCs) is based on the short term power spectrum of an audio, based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency, all MFCCs are is essentially all the coefficients that together make up a MFC. We carry out feature extraction using 'Librosa'. First, we extract 13 MFCCs using Librosa, and then we also calculate their first order derivative and second order derivative. Using this, we end up with a vector of 39 values. The purpose of calculating the first and second order derivative is to know how the MFCCs correspond to time, which can be useful in classifying audio that may have reverb added to it.

Based on our research, we selected MFCCs instead of Mel-Spectrogram, because even though it is documented that Mel-Spectrogram performs slightly better than MFCCs, the additional time complexity involved in using Mel-Spectrogram is also to be considered. We can however conclude that Mel-

Spectrogram will perform better than MFCCs in case of future need, and based on our evaluation we will be able to conclude how Mel-based audio features perform.

2) Spectral Centroid: In research, Spectral Centroid is considered an extremely useful feature extraction method because it describes the spectrum of the sound signal, and the calculation describes the center of the spectrum. This feature is directly linked to the brightness of the sound. It uses the weighted mean of the frequencies method to identify the center of the spectrum. We hypothesize that if noise is added to an audio track, spectral centroid may be useful in case the noise is evenly distributed across the signals and even if it is not, it may give a general idea about the track that may help in classification. However, we are not too confident in this feature as it considers a more wholistic view of the audio signal, but in the audio classification problem we have to consider more details which are found in Chromagram and MFCCs.

3) Chromagram: This extraction focuses on the pitch characteristic of the different types of sounds. It classifies sounds according to the pitch level it has. It identifies 12 different types of pitch classes and classifies each to its level. Based on our literature review, we found that Chromagrams are especially important to the musical sound analysis, however, it will be important in our analysis too since we expect that different algorithms used to generate synthetic audio cause specific changes to pitch classes which can be used to predict what sort of specific effect an algorithm has had on an audio. Using that information, Machine Learning based algorithms can classify between different synthetic speech generation audio algorithms.

1.2.1 Features to Consider

1.2.1.1 Feature Selection

To carry out feature selection for the final model, we implemented a brute force approach by trying all possible combinations of a set of MFCCs, Chromagram, and spectral centroid. Obviously, this method does introduce a limitation of high Computational Intensity. However, since we are only focusing on three features, instead of implementing greedy or wrapper-based methods, we decided to use this naïve approach so we could perfectly account for feature dependence. It is to be noted that, for many audio features, it would essentially take a large amount of time which would not be feasible for the purpose of this project.

Based on our literature review and understanding of the problem at hand, we do expect the best features to be a combination of MFCCs and Chromagram.

1.2.1.2 Dimensionality Reduction

Initially, we ran all the classifiers without applying Dimensionality Reduction. We were satisfied with the results, however, as expected since we ran all the machine learning algorithms exhaustively on the set of all possible selected features, it took a lot of time. We reran all the algorithms after applying dimensionality reduction and we found that the results remained consistent with little to no change, but we witnessed a significant improvement in performance.

The purpose of PCA in our project is for dimensionality reduction, for example, MFCCs with 39 dimensions is a lot for Machine Learning algorithms and for certain algorithms like K-Nearest Neighbor, it would be too slow due to involved greater number of calculations. It could be noted that we could just run PCA on the entirety of the combined datasets, but we want to be thorough and find the best features suited for synthetic algorithm classification, so applied them on individual features. Using a target explained variance of 95%, MFCCs were reduced from (1,39) to (1,26), and Chromagram was reduced from (1,12) to (1,7).

1.2.3 Machine Learning Models

For the purpose of this project, we will be looking at the following Machine Learning Models.

1) Support Vector Machine (SVM): SVM classifier has numerous uses in pattern recognition and its benefits in speech recognition are explored by different literature. SVM's are based on structural risk mitigation techniques in which the model does not over fit to the given data set and generalizes it enough to avoid overfitting of the model. Given the large amount of feature dimensions in our data, this technique will be useful to us. However, it is to be noted that SVM's tend to work the best when the classification is done on exactly two classes.
2) K Nearest Neighbor (KNN): One of the simplest and sometimes the most efficient, KNN algorithm calculates the distance between a test point and all the points in the training set, and depending on the number of K neighbours' it classifies the test point based on the shortest distance.
3) Naive Bayes: Naive Bayes is a probabilistic classifier, based on the Bayes theorem, which is that P(A/B)=(P(B/A)P(A))/P(B). The term 'naive' comes from the fact that we assume that there is independence among the features or predictors.
4) Neural Networks: Neural networks are a set of classification algorithms that are used to figure out the various relationships in the data. The term 'neural' comes from the fact that the algorithm is based upon how neurons in brains signal to one another. In essence, the goal of neural networks is to emulate how an actual brain would process information for a given task.
5) Logistic regression: It is a widely used machine learning classifier, which uses discrete or continuous features. For pattern classification it is widely used like SVM is, however, its use in automatic speech detection is recent. This converts each speech signal to a fixed d-vector and then the logistic model is used to predict the audio in data.

To tune the hyperparameters of the classification algorithms, we used GridSearchCV, which does an exhaustive search over all the parameters provided to return the best possible parameters of the algorithm.

1.3 Mathematical Formulation

1. Naive Bayes:

In Naive Bayes classifier, we follow an assumption that the features of audio signals are independent of each other. We assume that no pair of features are dependent. like the MFCC has no effect on other features like chroma, melspectro etc. Hence, the features are assumed to be **independent**. It follows the rule of Bayes theorem that follows assumption of independence among predictors.

We have a cleaned Data set of audio signals represented by notation:

$$D= \{ (x1,y1), (x2,y2),....., (xn,yn)\} \subseteq X^d * Y$$

In order to determine the best hypothesis $h \in \mathbf{H}$ we calculate posterior probability using the observed audio signal and given the hypothesis function. So we find h that maximizes

$$P(h|D) = P(D|h) * P(h) / P(D);$$

$P(h|D)$= posterior

$P(D|h) * P(h)$= Likelihood Function

$P(D)$= Prior

Given our data set with different classes {0,1,2,3,4} we find the hypothesis function that yields maximum posterior probability hence;

Y= {class 0, class 1, class 2, class 3, class 4)

$$hmap = \max h \in \mathbf{H}$$

$$P(h|\text{ class } 0) = \max \sum P(\text{class Yi } |h)*P(h)/P(\text{class Yi })$$

$$hMap = MAX \sum P(\text{Class Yi } |h) *P(h) \text{--------} > hML = MAX \sum P(\text{Class Yi } | h)$$

2. K-Nearest Neighbor:

The K Nearest Neighbor algorithm falls under the Supervised Learning category which is **used for classification (most commonly)**. It is a versatile algorithm also used for imputing missing values and resampling datasets. We determine the K nearest neighbor and assign the most frequent label. It is an extended version of a binary tree in higher dimensions.

The distance from all neighbors is computing using Manhattan distance. Considering the cleaned dataset,

Manhattan Distance: dist (x,x') = abs|xD-x'D'| = Σ |xid-xid'|

Euclidean Distance: dist (x,x') = |xD-x'D'|^2 = Σ sqrt [(xid-xid')^2]

We have a cleaned dataset of audio signals;

D= { (x1,y1), (x2,y2),....., (xn,yn)} ⊆ X^d * Y

Where; Y= {Class 0, Class 1, Class 2, Class 3, Class 4}

dist (x, x') >= max (x'',y'') ∈ **Sx** dist(x,x'') Using the

Sx, our classifier can defined as follows:

h(x) = mode ({y'': (x'',y'') ∈ **Sx**}

After computing the distances for both Manhattan and Euclidean at Different K values we fine tuned the parameters using train-test split and Validation data to learn about K value which gives us minimal validation loss on our cleaned and augmented dataset.

The Error rate can be found using: y*= h(x) = max P(y|x)

∈bayes classifier = 1- p (h(x)|x)= 1- P(y^*|x)

Probability of misclassification:
P (y|xNN) (1-P(y|x) + P (y|x) (1-P(y|xNN)

3. Neural Network:

Neural networks are a set of neurons arranged in layers where output is a nonlinear function of a linear combination with 3 different structures one at Input Layer, Other at Hidden Layer, and Output Layer. Each unit in network is referred to as Neuron and the connected Neurons are called Neural Network. In our given dataset of audio signals we have 5 different classes of algorithms hence 5 nodes network with different layers (x^1, x^2, x^3,x^4,x^5).

The forward pass function is used to transfer our input data points belonging from different classes into hidden layers. This function is as follows:

Z = wTx + B —-> G(Z) so

Zi[nx1] = wi[nx3]x[3x1] + [nx1]

Z[1]= w[1]X + B , ….. Z[5]= w[5]X + B

{Z[1]= w[1]a[0] + B,....Z[5]=w[5]a[4] + B} In

generaL;

a[L] = g(z[L]) and z[L]= w[L]a[L-1] + b[L]

We use an activation function (Softmax) to convert our nonlinear inputs to Linear combinations. As our dataset of audio signals consist of Multi-classes so softmax sigmoid function is used so our features (MFCC), CHROMAM, MEL SPECTRO transferred to neurons are non-linear hence we need activation function for that.

g(z)= σ(z)= 1/(1+e^-z)

To find the Total minimum loss using Gradient Descent for each output node back to the neural network we take partial derivatives. This helps us navigate the contribution of each and every node in the loss of system.

H(x) = wT⌀(x) + b; H(x) = wTmax((vTx + c) ,0) + b

L(H(x), y) = 1/2m Σ (m) (h(xi) - yi)^2

Squared error: L= ½ (a[l] - y)^2 = ½ || a[L] - Y||^2 Log-Loss (cross entropy) = L= -y log a[l] + (1-y) log (1-a[l])

Derivative of pre-activation output of the layer with respect to the input of the current layer or output of the previous layer:

$$\mathbf{z}^{[\ell+1]} = \mathbf{W}^{[\ell+1]}\mathbf{a}^{[\ell]} + \mathbf{b}^{[\ell+1]} \qquad \frac{\partial \mathbf{z}^{[\ell+1]}}{\partial \mathbf{a}^{[\ell]}} = \mathbf{W}^{[\ell+1]}$$

Gradient descent: $$w_{i,j}^{[\ell]} = w_{i,j}^{[\ell]} - \alpha \frac{\partial \mathcal{L}}{\partial w_{i,j}^{[\ell]}}$$

4. Support Vector Machines:

In SVM, we construct a model that constructs a Hyper-plane (used to separate classes) as a maximum margin classifier. The margin that pushes the data points are called the support vectors. We first look at Hard SVM, which does not allow misclassification of data points. We have n data-points, d number of real valued features x = [x1, …,xd] where the features could be any we have extracted using the feature extraction methods such

as MFCC. Hard SVM generates a Boolean output, y belongs to {1,-1} Where 1 represents category 1 and -1
category 2.

The way we will use SVM to implement a multiclass problem is by breaking down the multiclass problem that we have into multiple binary classification problems.

For the (s, t)- th classifier:

– Classifier 0 Samples: all the points in class 0 ({ xi : s ∈ yi })

– Classifier 1 samples: all the points in class 1 ({ xi : t ∈ yi })

– Classifier 2 Samples: all the points in class 2 ({ xi : s ∈ yi })

– Classifier 3 samples: all the points in class 3 ({ xi : t ∈ yi })

– Classifier 4 Samples: all the points in class 4 ({ xi : s ∈ yi })

– fs, t(x): the decision value of this classifierPrediction: f(x)= argmax s

(Σ t fs, t(x))

0 if Σ wi x(i) - θ >= 0 or Σ wi x(i) >=0

1 if Σ wi x(i) - θ <= 0 or Σ wi x(i) <=0

OR 1 1 if Σ wi x(i) - θ <= 0 or Σ wi x(i) <=0

2 if Σ wi x(i) - θ <= 0 or Σ wi x(i) <=0

And So on.

Support Vector is the data point for each class closest to the hyper-plane, and margin denoted by p is the distance between the support vectors. We define the problem in a way that our aim is to maximize the margin, and this depends on the support vectors.

Since the support vectors are at equal distance from the decision boundary, we define the plus plane as (w^T) x – theta = a and the minus plane as (w^T)x – theta = -b where w is perpendicular to both the decision boundary and plus/minus planes.

Take x^- as any points on the minus plane, and take a point x^+ that is closest to x^- and thus, we can relate x^-, x^+ and w as:

$$X+ = X- + \beta w$$

Since the margin p is the distance between plus and minus planes;

$\rho = ||X+ - X-|| = ||\beta w|| = \beta||w||$

$w^Tx+ - \theta = w^Tx- + \beta w^Tw - \theta = 1$

$\beta w^Tw = 2; \quad \beta = 2/w^Tw$ or $\beta = 2 / ||w||^2$

Since **$\rho = ||X+ - X-|| = ||\beta w|| = \beta||w||$, we have $\rho = 2/||w||$**

We have an express margin in terms of w, so we can now formulate the optimization problem. We want to minimize ||w||^2 but we should ensure that the training data points are classified correctly, thus:

Minimize w,θ: $||W||^2 = WT\ W$ subject

to $yi\ (W^Txi - \theta) > 1$

In case we are not able to find the Hard SVM, we use the Soft SVM, which is similarly formulated, however the difference is that it introduces Slack Variables, which allows the misclassification of difficult or noisy data points (which hard SVM does not allow).

Alternatively we can approach this using **One vs All Classifier** whereby SVM number 1 learns “class_output = 1” vs “class_output ≠ 1″ , SVM number 2 learns “class_output = 2” vs “class_output ≠2″ SVM number N learns “class_output = N” vs “class_output ≠N .Then to predict the output for new input, just predict with each of the build SVMs and then find which one puts the prediction the farthest into the particular class (behaves as a confidence criterion).

5. Logistic Regression:

It fits a linear model using the feature space provided by the data. It is a discriminative classifier as it learns which features are useful in our data to make different classes (Multi Dimensional feature space).Given the dataset we follow One vs All Classifier to predict labels. So mathematical formulation is:

$h\theta(x) = P(y|x)$ —> Posterior Probability

P (Classifier 0|MFCC's , Melspectrogram, Chromagram, Spectral centroid) = (P(MFCC's and Melspectrogram, Chromagram, Spectral centroid| Classifier 0) * P(Classifier 0)) / P(MFCC's and Melspectrogram, Chromagram, Spectral centroid)

P (Classifier 1|MFCC's , Melspectrogram, Chromagram, Spectral centroid) = (P(MFCC's and Melspectrogram, Chromagram, Spectral centroid| Classifier 1) * P(Classifier 1)) / P(MFCC's and Melspectrogram, Chromagram, Spectral centroid)

P (Classifier 2| MFCC's and Melspectrogram, Chromagram, Spectral centroid)= (P(MFCC's and Melspectrogram, Chromagram, Spectral centroid| Classifier 2) * P(Classifier 2)) / P(MFCC's and Melspectrogram, Chromagram, Spectral centroid)

P (Classifier 3| MFCC's and Melspectrogram, Chromagram, Spectral centroid)= (P(MFCC's and Melspectrogram, Chromagram, Spectral centroid| Classifier 3) * P(Classifier 3)) / P(MFCC's and Melspectrogram, Chromagram, Spectral centroid)

P (Classifier 4| MFCC's and Melspectrogram, Chromagram, Spectral centroid)= (P(MFCC's and Melspectrogram, Chromagram, Spectral centroid| Classifier 4) * P(Classifier 4)) / P(MFCC's and Melspectrogram, Chromagram, Spectral centroid)

We will use multi-class classification as we are dealing with speech processing which contains different labels to be classified. In our speech synthesis model X are the features that are extracted and we have to determine whether the speech signals belong to class 0,1 ,2 ,3 or 4. For simplification of the mathematical model, we will look at the multi-class classification.

Yà {Classifier 1, Classifier 2, Classifier 3, Classifier 4) Y{0,1,2,3,4}

or Y {1,2,3,4,5}

y^ max = max Si where i= 0-4
P (class 1| X) = hθ= 1/(1+ e^-θx) and

P (Other classes|X) = 1-hθ= 1/ (e^θx +1) so our general model of m classes is ; **P(y=m|x)= hθm(x)= e^θmTx / Σ(m-1) e^θkTx**

Theta here represents the model weights to the different feature dimensions. However, the output probability of this function should be between 0 and 1 so we will use activation function sigmoid to find probability distribution. **Logistic/Sigmoid**

Function: σx= 1/ (1+e^-z) =

1/(1+e^-hθ(x))

Now this activation function will provide us with correct probabilities and we can then expand this for further different features. The parameter of the decision boundaries of different features can be illustrated using logistic regression.

d=1 ; θ_T x= θ_0 + θ_1 $x_{(1)}$ = 0 d=2; θ_T

x= θ_0 + θ_1 $x_{(1)}$ + θ_2 $x_{(2)}$ = 0

We can use squared error loss function using cross entropy to see the efficiency of our classifier and we will use the following mathematical model for it.

Minimize:

$L(\theta) = -\sum_{(n)} \sum_{(m-1)}$ (yi-m) log log(h_θ(xi))

Or

$L(\theta) = -\sum_{(n)} \sum_{(m-1)}$ (yi-m) log(e^θmTx / Σ(m-1) e^θkTx))

Implementation:

Train-Test split is essential in developing machine learning models as we can efficiently make our models produce predictions on the test data that is not used in training the model itself. So the split needs to be rational, and in our models, we have used an 80-20 separation, which has been good enough. The feature extraction in our models is done through the librosa library, which is very efficient. We have chosen these functions from the librosa library to extract features from the audio files: MFCC, Spectral Centroid, Chromagram. After that, another central part of the implementation was the data preprocessing. This is useful because it makes it easier to interpret data and easy to use. It cleans the data so that there are no duplicates and irregularities. It also removes the problems that are in the data due to data entry.

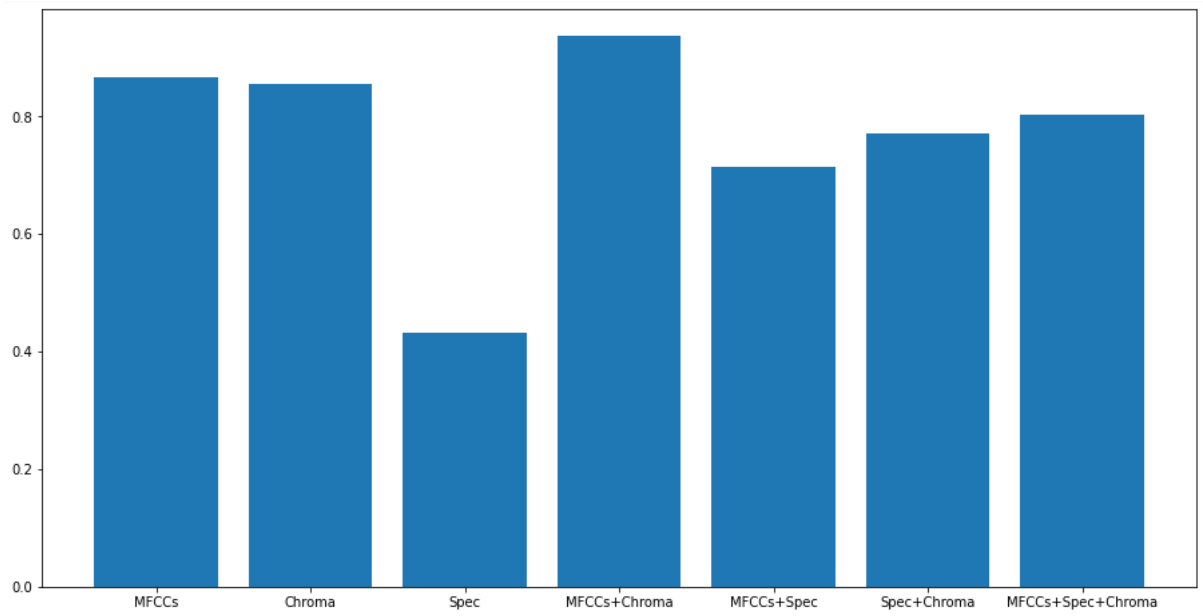
Data preprocessing gives accurate and consistent data, and that's why it is essential. So here, we removed background noise from our data by preprocessing. After that, we used Principle Component Analysis(PCA) to reduce the dimensions because it is difficult to handle large dimensions in the ML models. After that, we implemented SVM, KNN, Naïve Bayes, Neural Network, and Logistic regression model using the Sklearn library. After that, hyperparameter tuning was done through the GridsearchCv library. Hyperparameter tuning aims to find the best possible parameters that maximize the model's performance. However, we could not work on many hyperparameters due to time constraints. This is a general overview of our implementation, and the details of each are given ahead.

1.4.1 Results and Evaluation:

Before we consider looking at the performance metrics, we first take a minute to consider the problem at hand. We are classifying synthetic audios based on which algorithm was used to generate them. We are not currently concerned about having a lower number of false negatives, since there will not be severe consequences as there would be in for example a case where a security system is implemented using these results as a basis. So, our focus is on precision, how many algorithms have been correctly classified.

First, we implemented the K-Nearest Neighbor algorithm. For the clean dataset, we find that MFCCs and Chromagram work the best to achieve a high degree of accuracy of 93.6%. We find that Manhattan distance worked the best with the value of k neighbors

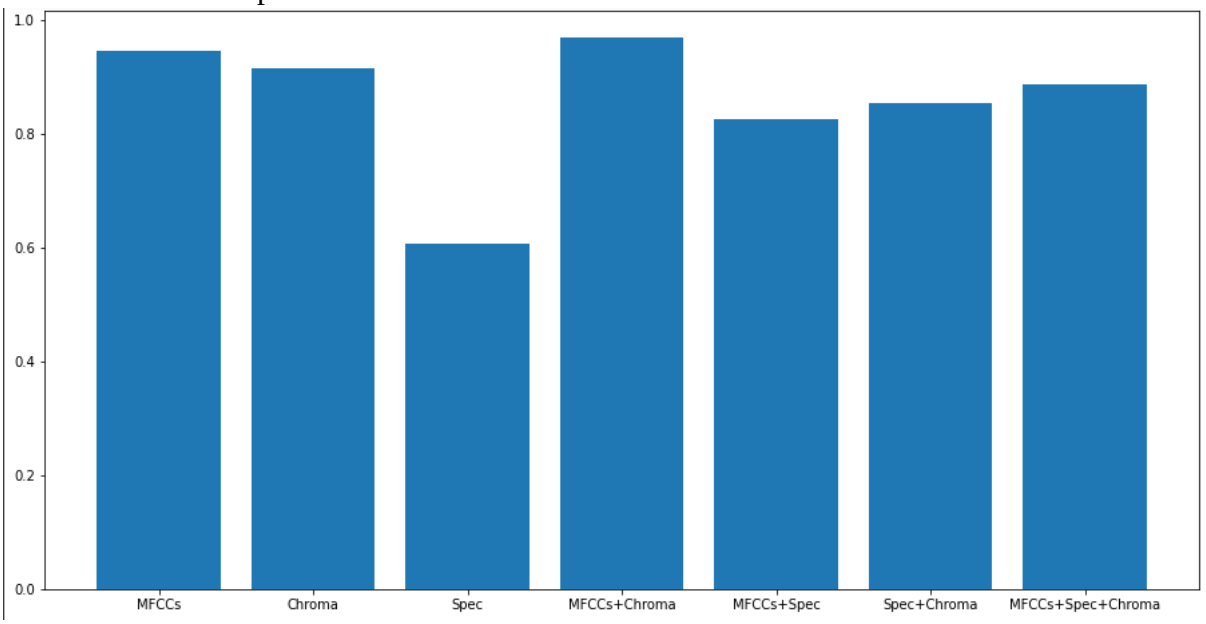
equal to 5. In terms of precision, we see that class 0, class 3 and class 4 has a near perfect score.



	precision	recall	f1-score	support
0	0.99	1.00	1.00	208
1	0.84	0.89	0.86	202
2	0.87	0.78	0.82	170
3	0.97	0.99	0.98	217
4	1.00	1.00	1.00	203
accuracy			0.94	1000
macro avg	0.93	0.93	0.93	1000
weighted avg	0.94	0.94	0.94	1000

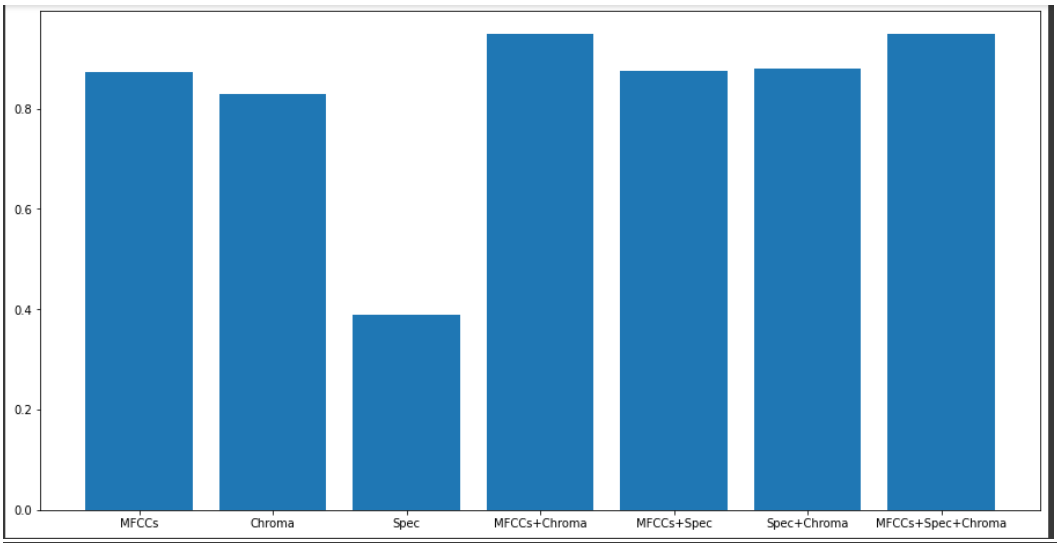
For the clean and augmented dataset combined, using Manhattan distance with k neighbors equal to 1, we achieved an accuracy of 96.775%. This time around too, class 0, class 4 and class 3 were nearly perfectly predicted, but we see a significant improvement in terms of the

clean dataset of the precision scores of classes 1 and 2.



	precision	recall	f1-score	support
0	0.99	1.00	0.99	816
1	0.93	0.94	0.93	842
2	0.94	0.91	0.93	795
3	0.98	1.00	0.99	772
4	1.00	0.99	1.00	775
accuracy			0.97	4000
macro avg	0.97	0.97	0.97	4000
weighted avg	0.97	0.97	0.97	4000

For Logistic Regression using l1 penalty, first looking at the clean dataset, we saw a significant improvement on the performance in terms of accuracy of MFCCs, Spectral Centroid and Chromagram features together, reaching 93%, which is comparable to the MFCCs and Chromagram performance of 94%. However, when we look at the evaluation metrics, we see that MFFCs Chromagram and Spectral Centroid together suffers from relatively poorer precision, recall and F-1 Scores.



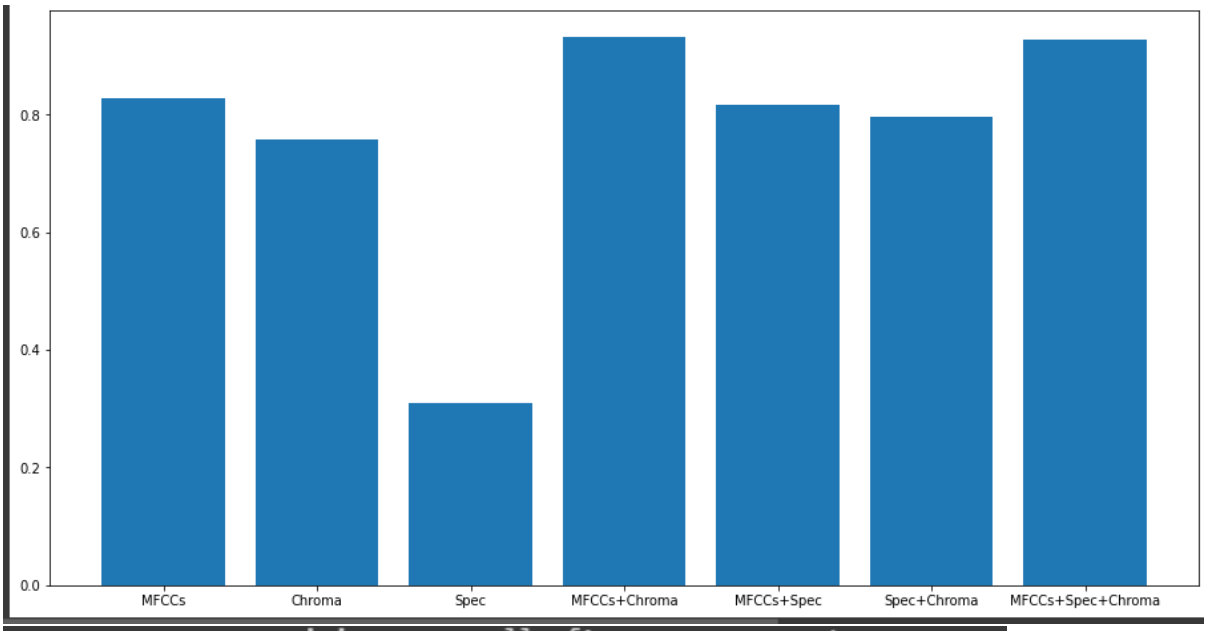
For MFCCs+Chroma
The classifier used is Logistic Regression with the l1 penalty

	precision	recall	f1-score	support
0	1.00	1.00	1.00	208
1	0.90	0.85	0.87	202
2	0.83	0.89	0.86	170
3	0.99	0.99	0.99	217
4	1.00	1.00	1.00	203
accuracy			0.95	1000
macro avg	0.94	0.94	0.94	1000
weighted avg	0.95	0.95	0.95	1000

For MFCCs+Spec+Chroma
The classifier used is Logistic Regression with the l1 penalty

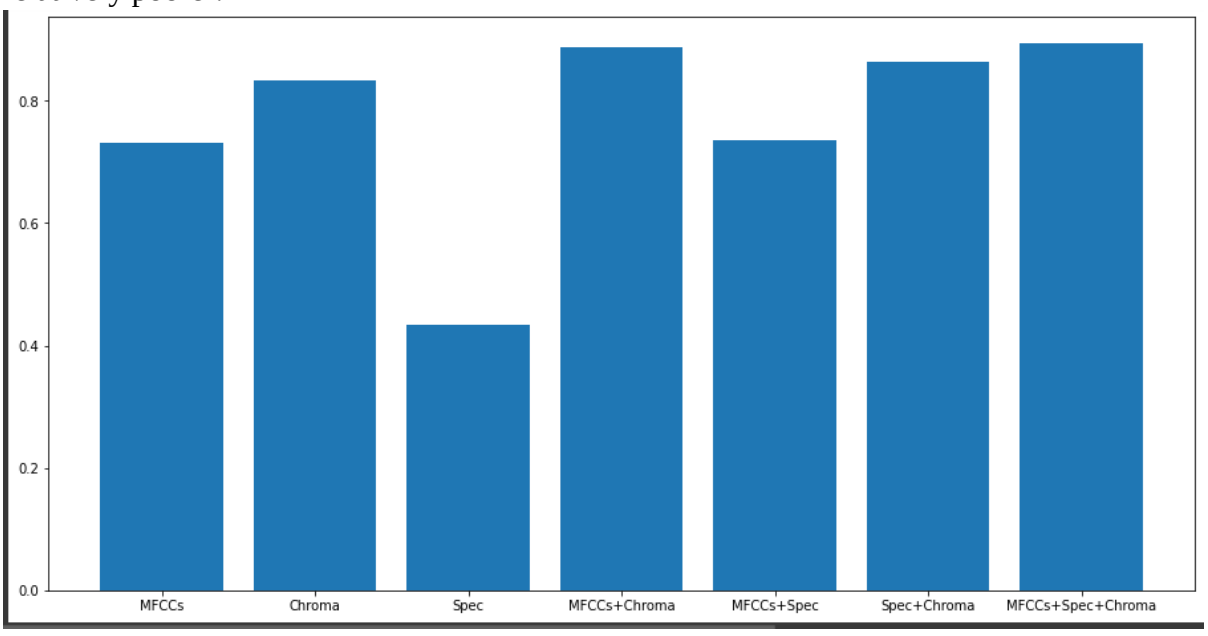
	precision	recall	f1-score	support
0	0.99	1.00	0.99	208
1	0.85	0.80	0.83	202
2	0.80	0.83	0.81	170
3	0.97	0.99	0.98	217
4	1.00	0.99	1.00	203
accuracy			0.93	1000
macro avg	0.92	0.92	0.92	1000
weighted avg	0.93	0.93	0.93	1000

For the augmented and clean dataset combined, we again saw that MFCCs and Chromagram work the best together in terms of accuracy reaching 93.075%. This time around, we found that elastic net regression worked the best, with l1 ratio of 0.6666. But we do see poor class 1 and class 2 precision.



	precision	recall	f1-score	support
0	1.00	1.00	1.00	816
1	0.83	0.86	0.85	842
2	0.85	0.82	0.83	795
3	0.99	0.98	0.98	772
4	1.00	1.00	1.00	775
accuracy			0.93	4000
macro avg	0.93	0.93	0.93	4000
weighted avg	0.93	0.93	0.93	4000

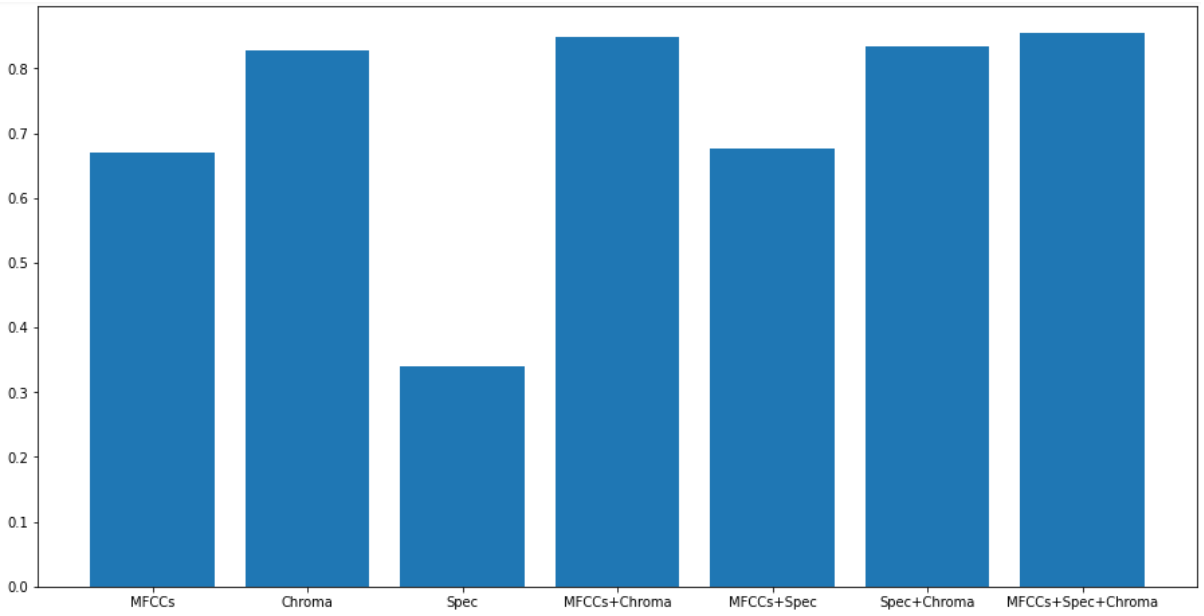
For Naïve Bayes, similarly, MFCCs and Chromagram work best together with an accuracy of 88.8%, however, this time around MFCCs, Spectral Centroid and Chromagram achieved a higher accuracy of 89.3 %. However, compared to KNN and logistic regression, the precision scores of class 1 and class 2 are relatively poorer.



```
For MFCCs+Chroma
The classifier used is  Naive Bayes  with accuracy  0.888
              precision    recall  f1-score   support

           0       1.00      0.91      0.96       199
           1       0.77      0.86      0.81       219
           2       0.76      0.75      0.75       187
           3       0.96      0.91      0.93       191
           4       0.99      1.00      0.99       204

    accuracy                           0.89      1000
   macro avg       0.89      0.89      0.89      1000
weighted avg       0.89      0.89      0.89      1000
```

```
For MFCCs+Spec+Chroma
The classifier used is  Naive Bayes  with accuracy  0.893
              precision    recall  f1-score   support

           0       1.00      0.92      0.96       199
           1       0.78      0.86      0.82       219
           2       0.78      0.76      0.77       187
           3       0.96      0.91      0.93       191
           4       0.98      1.00      0.99       204

    accuracy                           0.89      1000
   macro avg       0.90      0.89      0.89      1000
weighted avg       0.90      0.89      0.89      1000
```

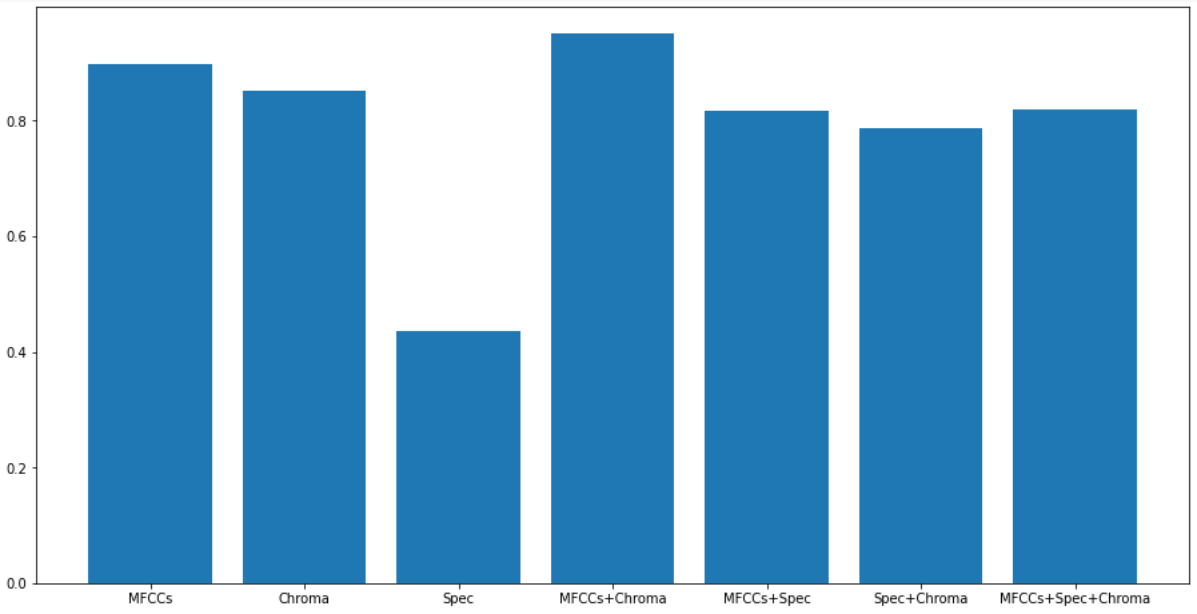
Looking at the clean and augmented dataset combined, MFCCs spectral centroid and chromagram perform the best with an accuracy of 85.4 %, however the performance of MFCCs and Chromagram is quite comparable with an accuracy of 84.8%. However, the same problem of poor precision and recall scores for classes 1 and 2 still exists for the classifier.



```
For MFCCs+Chroma
The classifier used is  Naive Bayes  with accuracy  0.848
              precision    recall  f1-score   support

           0       0.95      0.92      0.94       771
           1       0.68      0.79      0.73       791
           2       0.72      0.63      0.67       857
           3       0.95      0.91      0.93       785
           4       0.96      1.00      0.98       796

    accuracy                           0.85      4000
   macro avg       0.85      0.85      0.85      4000
weighted avg       0.85      0.85      0.85      4000
```

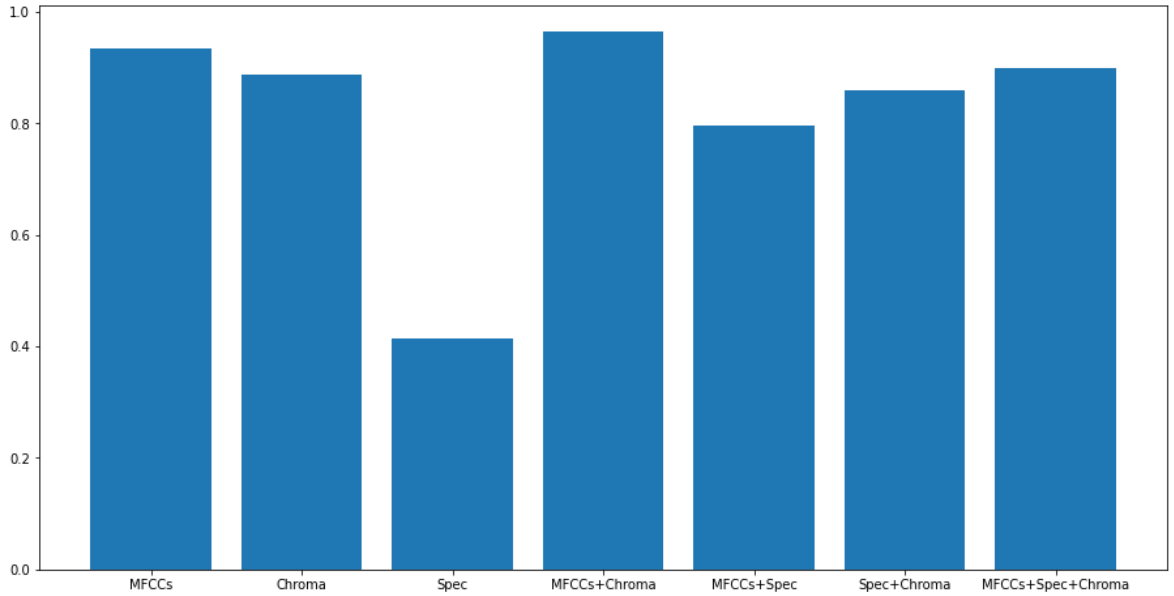
```
For MFCCs+Spec+Chroma
The classifier used is  Naive Bayes  with accuracy  0.854
              precision    recall  f1-score   support

           0       0.97      0.93      0.95       771
           1       0.68      0.80      0.74       791
           2       0.73      0.65      0.68       857
           3       0.96      0.92      0.94       785
           4       0.97      1.00      0.98       796

    accuracy                           0.85      4000
   macro avg       0.86      0.86      0.86      4000
weighted avg       0.86      0.85      0.85      4000
```

For support Vector machines, we see that on the clean dataset, MFCCs and Chromagram combined perform the best together, achieving an accuracy of 95%, the highest we have achieved on the clean data set until now, using radial basis function as the activation function, 0.01 as the regularization parameter. We also see good class 1 and 2 precision scores but nowhere as good as classes 0,3 and 4.



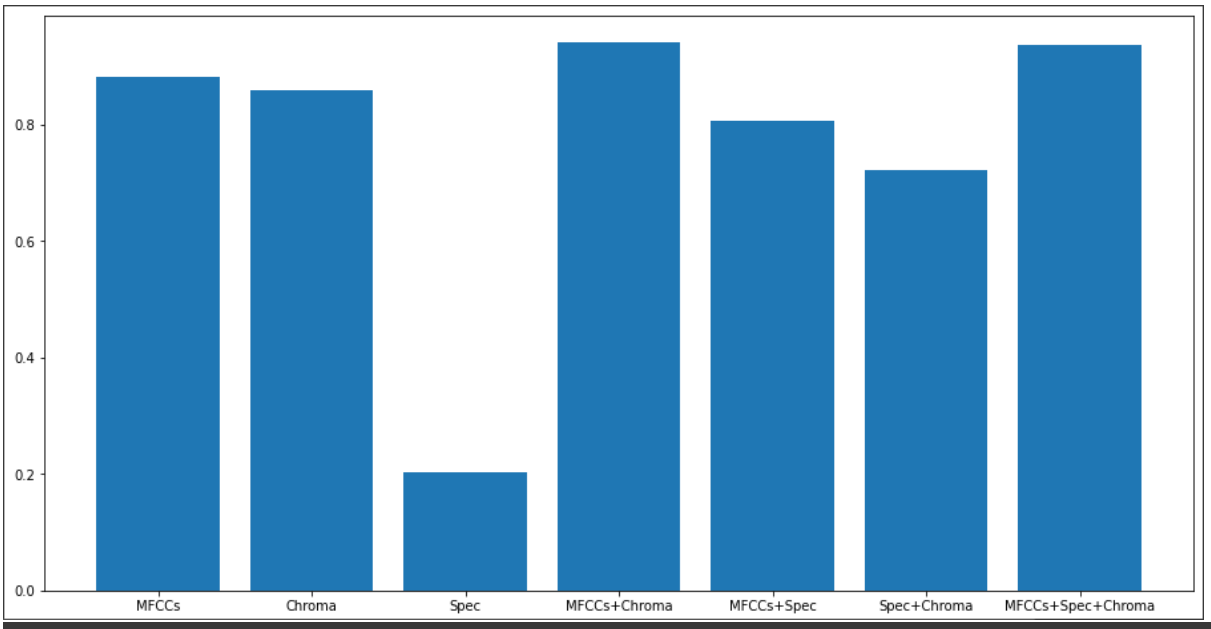
	precision	recall	f1-score	support
0	1.00	1.00	1.00	204
1	0.89	0.87	0.88	203
2	0.87	0.90	0.88	194
3	1.00	0.99	0.99	207
4	1.00	1.00	1.00	192
accuracy			0.95	1000
macro avg	0.95	0.95	0.95	1000
weighted avg	0.95	0.95	0.95	1000

On the augmented and clean dataset combined, we see MFCCs and Chromagram perform the best with an accuracy of 96.4 %. The hyperparameters used were the radial basis function as the activation function and 0.1 as the regularization parameter. We see a significant improvement in classes 1 and 2, with a high degree of precision of 0.95 and 0.93 respectively, but class 4 suffers from lower precision this time around.



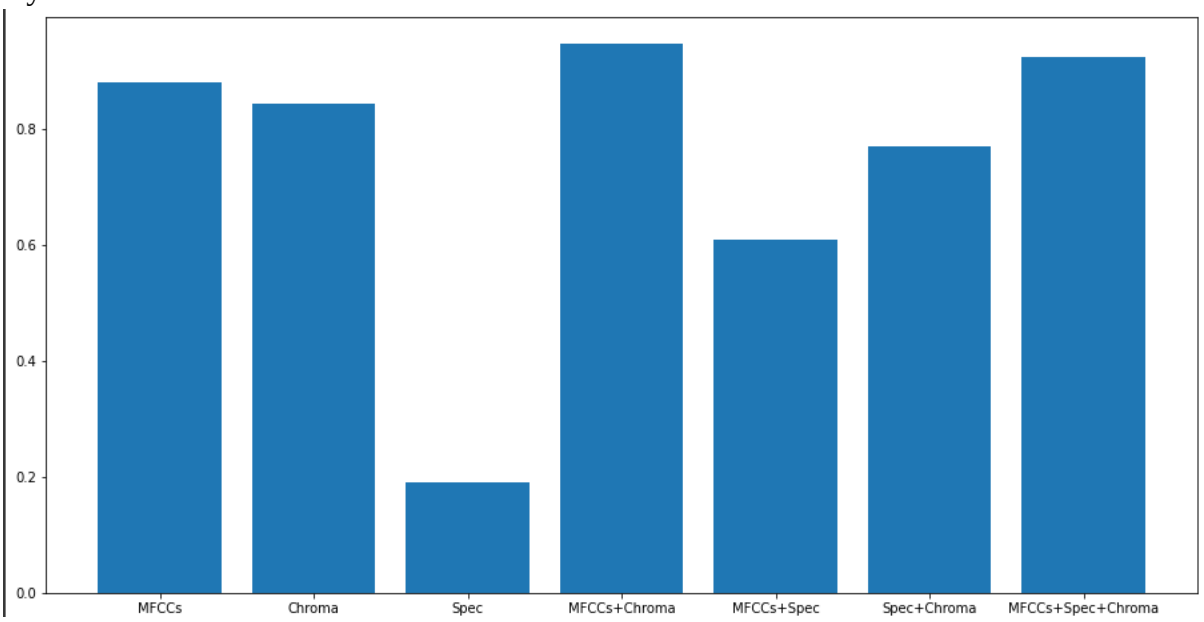
0	1.00	0.99	0.99	792
1	0.95	0.93	0.94	797
2	0.93	0.93	0.93	816
3	1.00	0.97	0.99	811
4	0.94	1.00	0.97	784
accuracy			0.96	4000
macro avg	0.96	0.96	0.96	4000
weighted avg	0.96	0.96	0.96	4000

On Neural Networks, we were able to get an accuracy of 94% on the clean dataset combined, using MFCCs and Chromagram together. We used relu as the activation function, with 0.001 as the initial learning rate and the number of hidden layers was 12. Class 1 does have a low precision of 79%, but we see a significant improvement in class 2 precision of 86% relative to other classifiers.




```
For MFCCs+Chroma
The classifier used is  Neural Network  with accuracy  0.94 , the initial learning rate is  0.001  and the number of hidden lay
with the activation function:  relu
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       807
           1       0.79      0.87      0.83       787
           2       0.86      0.79      0.83       799
           3       0.98      0.97      0.98       845
           4       1.00      1.00      1.00       762

    accuracy                           0.92      4000
   macro avg       0.93      0.92      0.92      4000
weighted avg       0.93      0.92      0.92      4000
```

Similarly, we achieved an accuracy of 94.625 % on the augmented and clean dataset combined, using an initial learning rate of 0.005, relu and 12 hidden layers.

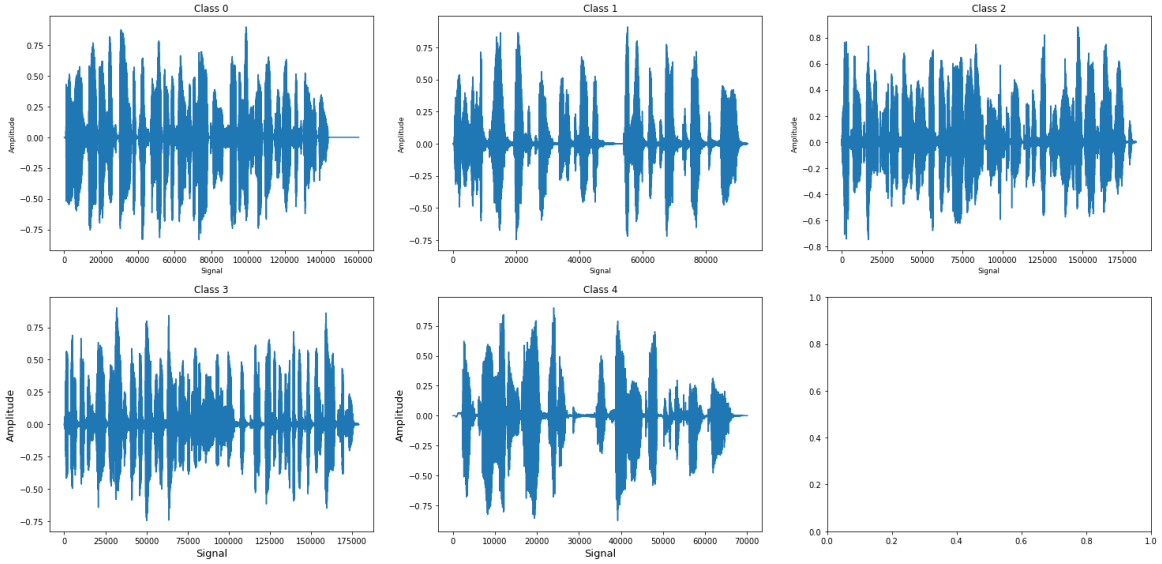


With the activation function: relu

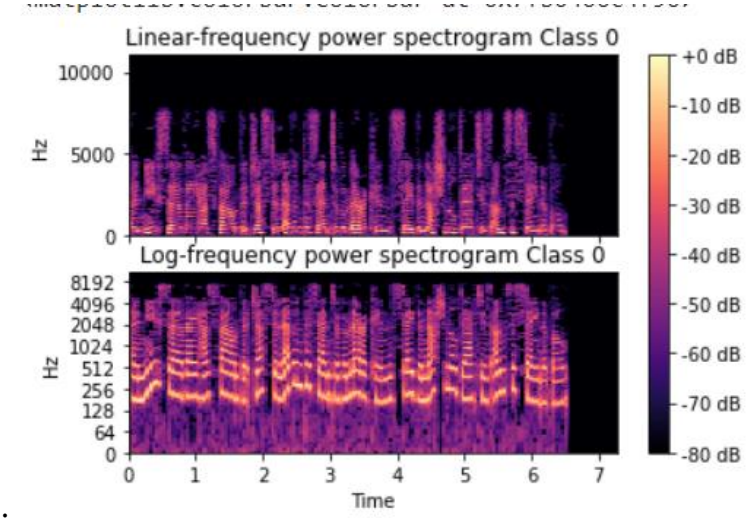
	precision	recall	f1-score	support
0	1.00	1.00	1.00	807
1	0.86	0.88	0.87	787
2	0.89	0.86	0.87	799
3	0.99	0.99	0.99	845
4	1.00	1.00	1.00	762
accuracy			0.95	4000
macro avg	0.95	0.95	0.95	4000
weighted avg	0.95	0.95	0.95	4000

1.4.2 Insights based on results and evaluation:

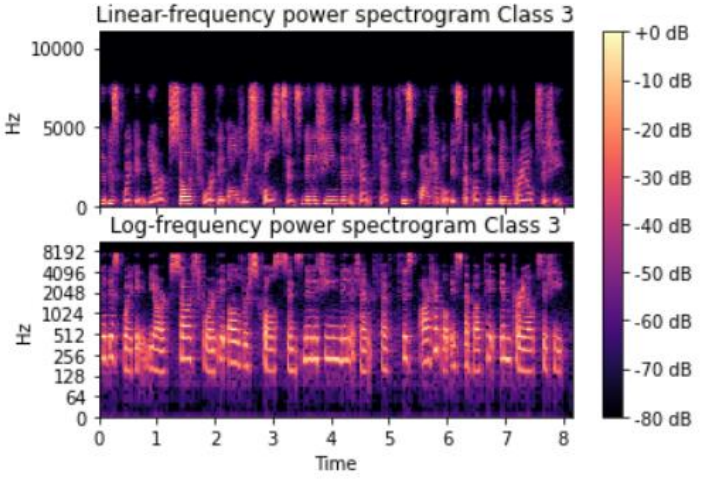
We see that based on our evaluation, classes 1 and 2 suffer from the worst precision scores, so we want to analyze what feature could possibly be used that would help in the classification of these classes. We first analyzed a random sample of each class based on amplitude and signal, however we found no useful information as we saw no significant visual difference in the variance of amplitude between different classes.

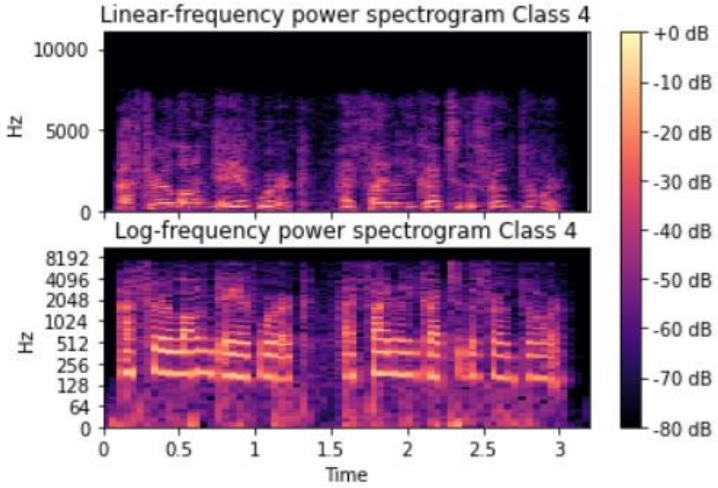


Next, one possibly important feature that we did not include was Fourier Transform. We visualize the Linear-Frequency and log frequency spectrogram to visualize fast Fourier transforms (it is to be noted that short term Fourier transforms tend to be more accurate, however for our purposes it should suffice to see whether Fourier transform introduces a visible difference in the different audios). First, looking at classes 0, 3 and 4, we do not notice a visible difference.

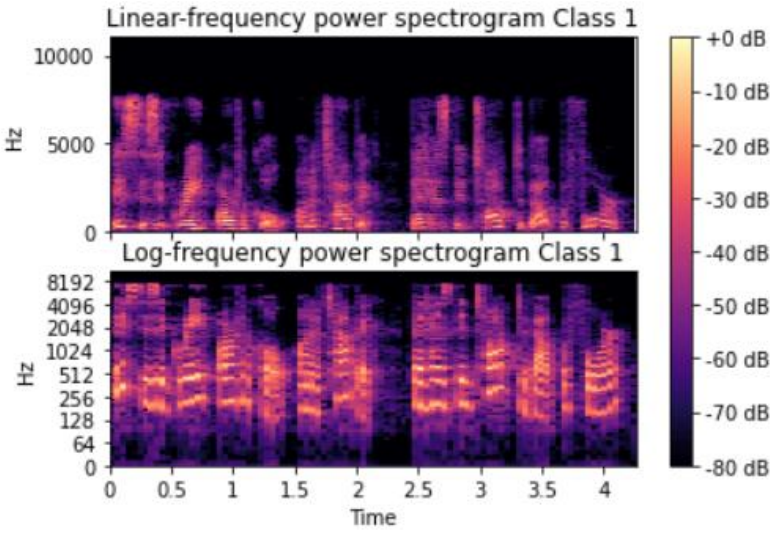


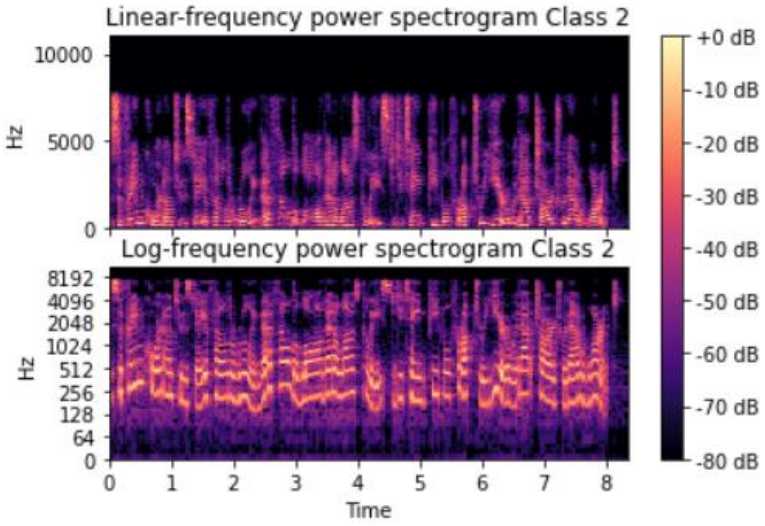
.





For classes 1 and 2, we again do not see a visible difference unfortunately. Perhaps it is our limitation of interpreting audio data, but from our naïve understanding of audio features, introducing Fourier Transform based features should not affect the performance of the model dramatically.


Linear-frequency power spectrogram Class 1
10000
5000
0
Hz
Log-frequency power spectrogram Class 1
8192
4096
2048
1024
512
256
128
64
0
Hz
0
0.5
1
1.5
2
2.5
3
3.5
4
Time
+0 dB
-10 dB
-20 dB
-30 dB
-40 dB
-50 dB
-60 dB
-70 dB
-80 dB


Linear-frequency power spectrogram Class 2
10000
5000
0
Hz
Log-frequency power spectrogram Class 2
8192
4096
2048
1024
512
256
128
64
0
Hz
0
1
2
3
4
5
6
7
8
Time
+0 dB
-10 dB
-20 dB
-30 dB
-40 dB
-50 dB
-60 dB
-70 dB
-80 dB

1.5 Conclusion:

First, we come back to our brute force feature selection method, we realized that across the board with various different classifiers, Spectral Centroid was not a good feature in terms of synthetic speech algorithm classification, so for future purposes we propose spectral centroid in its original form not to be considered for the purpose of any model building for a similar cause, and MFCCs and Chromagram performed the best (in our case). However, since we only analyzed these three features, it is to be noted there may be better features suited for speech algorithm classification.

Now, coming back to our model, albeit the fact that in hindsight, for the clean dataset, Support Vector Machines tend to perform the best with an achieved accuracy of 95% while the K-Nearest Neighbor performs the best with an achieved accuracy of 96.775, we also consider the precision scores in our conclusion. Classes 1 and classes 2 suffer from poorer precision scores in K-Nearest Neighbor algorithm, so we propose that Support Vector Machines (which achieved an accuracy of 96.4 on our test data) be used as the final model since it balances the scores between classes 1 2 and 4 while classes 0 and 3 achieve a near perfect score.

For future reference, if we look across the board, we see that the classes 1 and 2 suffer from poorer precision scores compared to other classes. If the model is able to distinguish between these two classes and other classes better, a near perfect 99%+ accuracy score could possibly be achieved. should be further analyzed, if we are able to determine which algorithm was used to generate them, perhaps a better suited feature could be employed by the model.

Works Cited

Borrelli, Clara, et al. “Synthetic Speech Detection through Short-Term and Long-Term Prediction Traces - EURASIP Journal on Information Security.” *SpringerOpen*, Springer International Publishing, 6 Apr. 2021, https://jis-eurasipjournals.springeropen.com/articles/10.1186/s13635-021-00116-3.

Li, Haizhou, and Zhizheng Wu. “Synthetic Speech Detection Using Temporal Modulation Feature.” *IEEE Xplore*, https://ieeexplore.ieee.org/abstract/document/6639067?casa_token=zJDqdWErstAAAAAA%3AU3RXD8V1NMT41R_-Dcj_N1f546I5TIDwemjtsrkQCVURU_qmBj4zKnvsITva9Bz1ooxOyKhy9Q.

Awad, Mariette. “(PDF) Support Vector Machines for Classification.” *ResearchGate*, 1 Jan. 2015, https://www.researchgate.net/publication/300723807_Support_Vector_Machines_for_Classification.

Doshi, Ketan. “Audio Deep Learning Made Simple: Sound Classification, Step-by-Step.” *Medium*, Towards Data Science, 21 May 2021, https://towardsdatascience.com/audio-deep-learning-made-simple-sound-classification-step-by-step-cebc936bbe5.

Mendiratta, Sunanda, and Dipali Bansal. *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, 30 Apr. 2022, https://www.ijitee.org/.

Rong, Feng. *Audio Classification Method Based on Machine Learning*. 1 Dec. 2016, https://www.researchgate.net/publication/319971531_Audio_Classification_Method_Based_on_Machine_Learning.

Katyal, Anchal, and Jasmeen Gill. *International Journal of Engineering and Advanced Technology (IJEAT)*, 6 Apr. 2022, https://www.ijeat.org/.

Noda, Kuniaki, et al. “Audio-Visual Speech Recognition Using Deep Learning - Applied Intelligence.” *SpringerLink*, Springer US, 20 Dec. 2014, https://link.springer.com/article/10.1007/s10489-014-0629-7.

Appendix:

- Cleaned Dataset Pre-processing:

```
#Preparing the Dataset

#clean dataset
data_path = "drive/MyDrive/Dataset Project 1/speech algorithm classification dataset/algorithm classification dataset"
data_folder = os.walk(data_path)
aud_list = []
classes_list = []
file = open(data_path + '/'+'labels.csv')
csvreader = csv.reader(file)
csv_list=[]
for row in csvreader:
  csv_list.append(row)
file.close()

for (root,dirs,files) in data_folder:
  for name in files:
    if name != "labels.csv":
      aud = (data_path+'/'+name)
      for x in csv_list:
        if x[0]==name:
          classes_list.append(x[1])
          aud_list.append(aud)
          break
```

- Augmented Dataset Pre-processing:

```
#augmented dataset
def nameParser(filename):
  if (filename[-10:-4])=="_noise":
    file2 = filename.replace("_noise","")
    return file2
  if (filename[-11:-4])=="_reverb":
    file2 = filename.replace("_reverb","")
    return file2
  if(filename[-15:-4])=="_compressed":
    file2 = filename.replace("_compressed","")
    return file2

data_path = "drive/MyDrive/Dataset Project 1/augmented_training_data/augmented_training_data"
data_folder = os.walk(data_path)

file = open(data_path + '/'+'labels.csv')
csvreader = csv.reader(file)
csv_list=[]
for row in csvreader:
  csv_list.append(row)
file.close()
```

- Principal Component Analysis

```
# First, we generate 1000 random numbers for train test split
import random

random_ints = random.sample(range(4999), 1000)
total_ints = list(range(0,5000))
split_ints =[]
for inte in total_ints:
  if inte not in random_ints:
    split_ints.append(inte)


def splitter(list_numbers,data,split_int):
  test_data = []
  train_data=[]
  for x in list_numbers:
    test_data.append(data[x])
  for x in split_int:
    train_data.append(data[x])
  return (test_data,train_data)

#Train Test Split for Clean Dataset
mfcc_clean = mfcc_list[0:5000]
mfcc_clean_test, mfcc_clean_train = splitter(random_ints,mfcc_clean,split_ints)
spectral_centroid_clean = spectral_centroid[0:5000]
spec_test_clean, spec_train_clean = splitter(random_ints,spectral_centroid_clean,split_ints)
chroma_list_clean = chroma_list[0:5000]
chroma_test_clean, chroma_train_clean = splitter(random_ints,chroma_list_clean,split_ints)
classes_list_clean = classes_list[0:5000]
classes_test_clean,classes_train_clean = splitter(random_ints,classes_list_clean,split_ints)
classes_test1 = classes_test_clean
classes_train1= classes_train_clean
```

```
print("original MFCC feature")
print((mfcc_clean_test[0].shape))
from sklearn import preprocessing
scaler = preprocessing.StandardScaler().fit(mfcc_clean_train)
mfcc_clean_train_scaled = scaler.transform(mfcc_clean_train)
print(mfcc_clean_train_scaled[0])
mfcc_clean_test_scaled = scaler.transform(mfcc_clean_test)
from sklearn.decomposition import PCA
pca = PCA(n_components = 0.95)
pca.fit(mfcc_clean_train_scaled)
mfcc_train1 = pca.transform(mfcc_clean_train_scaled)
mfcc_test1 = pca.transform(mfcc_clean_test_scaled)
print("After mfcc shape")
print(mfcc_train1[0].shape)

print("original spec feature")
print((spec_train_clean[0].shape)) # Since the shape is already 1, no need to perform PCA
print("Shape already 1, no need to use PCA")
spec_train1 = spec_train_clean
spec_test1 = spec_test_clean
print("Original chroma feature")
print(chroma_list_clean[0].shape)
scaler = preprocessing.StandardScaler().fit(chroma_train_clean)
chroma_train_clean = scaler.transform(chroma_train_clean)
chroma_test_clean = scaler.transform(chroma_test_clean)

pca = PCA(n_components = 0.95)
```

- Augmented and Cleaned Dataset combined:

```
# Augmented Dataset and Clean Dataset Combined
random_ints = random.sample(range(19999), 4000)
total_ints = list(range(0,20000))
split_ints =[]
for inte in total_ints:
  if inte not in random_ints:
    split_ints.append(inte)

mfcc_clean_test, mfcc_clean_train = splitter(random_ints,mfcc_list,split_ints)
spec_test_clean, spec_train_clean = splitter(random_ints,spectral_centroid,split_ints)
chroma_test_clean, chroma_train_clean = splitter(random_ints,chroma_list,split_ints)
classes_test_clean,classes_train_clean = splitter(random_ints,classes_list,split_ints)
classes_test2 = classes_test_clean
classes_train2 = classes_train_clean

print("original MFCC feature")
print((mfcc_clean_test[0].shape))
from sklearn import preprocessing
scaler = preprocessing.StandardScaler().fit(mfcc_clean_train)
mfcc_clean_train_scaled = scaler.transform(mfcc_clean_train)
mfcc_clean_test_scaled = scaler.transform(mfcc_clean_test)
from sklearn.decomposition import PCA
pca = PCA(n_components = 0.95)
pca.fit(mfcc_clean_train_scaled)
mfcc_train2 = pca.transform(mfcc_clean_train_scaled)
mfcc_test2 = pca.transform(mfcc_clean_test_scaled)
print("After mfcc shape")
print(mfcc_train2[0].shape)
```

- Feature Selection through brute force:

```
#mfcc,spec,chroma,mfcc_spec,mfcc_chroma,chroma_spec,mfccs_spec_chroma

mfcc_chroma=[] #mfcc chroma
mfcc_spec = [] #mfcc spec
chroma_spec =[] # chroma spec
mfcc_spec_chroma=[] # mfcc spec chroma

def maker(mfcc_lister,spec_lister,chroma_lister):
  mfcc_chroma=[] #mfcc chroma
  mfcc_spec = [] #mfcc spec
  chroma_spec =[] # chroma spec
  mfcc_spec_chroma=[] # mfcc spec chroma
  for x in range(len(mfcc_lister)):
    finale=[]
    finale = np.concatenate((mfcc_lister[x],chroma_lister[x]))
    mfcc_chroma.append(finale)
    finale=[]
    finale = np.concatenate((mfcc_lister[x],spec_lister[x]))
    mfcc_spec.append(finale)
    finale=[]
    finale = np.concatenate((chroma_lister[x],spec_lister[x]))
    chroma_spec.append(finale)
    finale=[]
    finale = np.concatenate((mfcc_lister[x],spec_lister[x],chroma_lister[x]))
    mfcc_spec_chroma.append(finale)
  return mfcc_chroma,mfcc_spec,chroma_spec,mfcc_spec_chroma

mfcc_chroma_train1,mfcc_spec_train1,chroma_spec_train1,mfcc_spec_chroma_train1 = maker(mfcc_train1,spec_train1,chroma_train1)
mfcc_chroma_train2,mfcc_spec_train2,chroma_spec_train2,mfcc_spec_chroma_train2 = maker(mfcc_train2,spec_train2,chroma_train2)
mfcc_chroma_test1,mfcc_spec_test1,chroma_spec_test1,mfcc_spec_chroma_test1 = maker(mfcc_test1,spec_test1,chroma_test1)
mfcc_chroma_test2,mfcc_spec_test2,chroma_spec_test2,mfcc_spec_chroma_test2 = maker(mfcc_test2,spec_test2,chroma_test2)
```

- Sklearn implementation of a KNN:

```
from sklearn.metrics import classification_report

def KnnImplementation(X_train,X_test,y_train,y_test):
  knn = (KNeighborsClassifier())
  parameters = {
      'n_neighbors' :[1,3,5,7,9,11,13,15,17,19,21,23,25,27,29],
      'p' :[1,2],
      'algorithm':['auto']
  }

  final = GridSearchCV(estimator = (KNeighborsClassifier()),param_grid=parameters,scoring='accuracy',cv=10)
  final2 = final.fit(X_train,y_train)

  if final2.best_params_['p']==1:
    best_dis = "Manhattan"
  if final2.best_params_['p']==2:
    best_dis = "Euclidean"


  knn = (KNeighborsClassifier (n_neighbors = final2.best_params_['n_neighbors'],p=final2.best_params_['p'],algorithm = 'brute'))
  knn.fit(X_train,y_train)

  predict_list= knn.predict(X_test)

  acc_score = accuracy_score(y_test,predict_list)

  f1_scor = f1_score(y_test,predict_list,average='macro')
  report = classification_report(y_test,predict_list)

  return((acc_score,best_dis,final2.best_params_['n_neighbors'],'KNN',report))
```

- Sklearn implementation of a Logistic Regression:

```
def LogisticRegressionImplementation(X_train,X_test,y_train,y_test):
  parameters = {
      'penalty' :["l1","l2"],
      'random_state':[0],
      'max_iter':[1000],
      'solver':["liblinear"]

  }

  final = GridSearchCV(estimator = LogisticRegression(),param_grid=parameters,scoring='accuracy',cv=5)
  final2 = final.fit(X_train,y_train)
  clf = LogisticRegression(random_state=0,max_iter = 1000,solver="liblinear",penalty = final2.best_params_['penalty']).fit(X_train,y_train)
  list_predict = clf.predict(X_test)
  acc_score1 = accuracy_score(y_test,list_predict)

  parameters2= {
      'penalty':["elasticnet"],
      'l1_ratio': np.linspace(0,1,num=10),
      'random_state':[0],
      'max_iter':[1000],
      'solver':["saga"],
      'n_jobs':[-1]

  }
  final3 = GridSearchCV(estimator = LogisticRegression(),param_grid=parameters2,scoring='accuracy',cv=5)
  final4 = final3.fit(X_train,y_train)
```

```
clf = LogisticRegression(l1_ratio = final4.best_params_['l1_ratio'],random_state=0,max_iter=1000,penalty='elasticnet',solver='saga',n_jobs=-1).fit(X_train, y_train)
list_predict = clf.predict(X_test)
acc_score2 = accuracy_score(y_test,list_predict)
report = classification_report(y_test,list_predict)

if acc_score1>=acc_score2:
  return ((acc_score1,final2.best_params_['penalty'],'Logistic Regression',report))
else:
  return ((acc_score2,final4.best_params_['penalty'],final4.best_params_['l1_ratio'],'Elastic Net Logistic Regression',report))
```

- Sklearn implementation of a Naïve Bayes:

```
def naiveBayesImplementation(X_train,X_test,y_train,y_test):
  gnb = GaussianNB()

  predict_list= gnb.fit(X_train, y_train).predict(X_test)

  acc_score = accuracy_score(y_test,predict_list)
  report = classification_report(y_test,predict_list)
  return((acc_score,'Naive Bayes',report))
```

```
#For the Clean Dataset
result_list = []
for x in features1:
  x_train = x[1]
  x_test = x[2]
  res = naiveBayesImplementation(x_train,x_test,classes_train1,classes_test1)
  result_list.append(res)
#For augmented
result_list2 = []
for x in features2:
  x_train = x[1]
  x_test = x[2]
  res = naiveBayesImplementation(x_train,x_test,classes_train2,classes_test2)
  result_list2.append(res)
```

- Sklearn implementation of a Support Vector Machine (SVM):

```
from sklearn.metrics import classification_report

from sklearn.svm import SVC
def SVMimplementation(X_train,X_test,y_train,y_test):
  parameters= {
    'C': [0.1, 1],
    'gamma': [1, 0.1, 0.01],
    'kernel': ['rbf','linear','sigmoid'],
    'cache_size':[1500],
    'max_iter': [10000]
  }
  final3 = GridSearchCV(estimator = SVC(),param_grid=parameters,scoring='accuracy',cv=5)
  final4 = final3.fit(X_train,y_train)

  list_predict = final4.predict(X_test)
  acc_score2 = accuracy_score(y_test,list_predict)
  report = classification_report(y_test,list_predict)
  return ((acc_score2,final4.best_params_['C'],final4.best_params_['kernel'],final4.best_params_['gamma'],'Support Vector Machine',report))
```

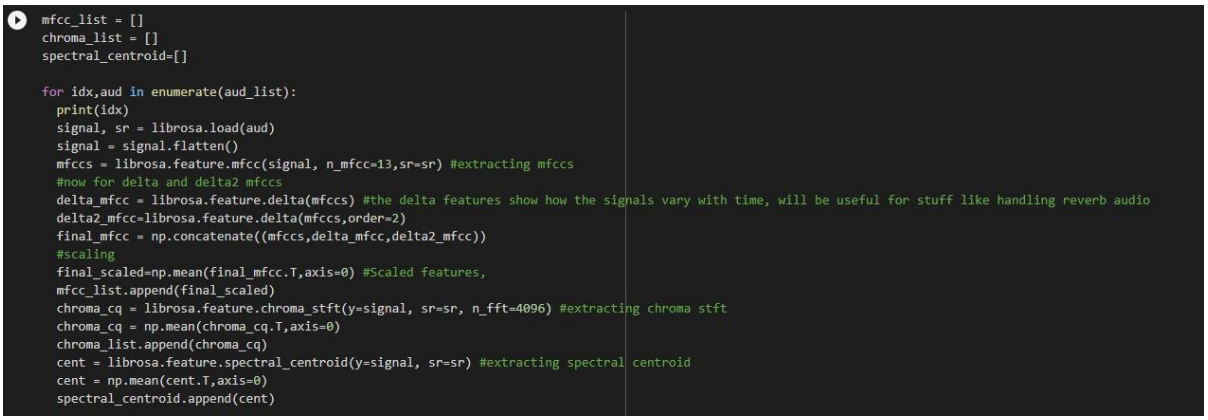
- Sklearn implementation of a Neural Network:

```
[ ] from sklearn.metrics import classification_report
    from sklearn.neural_network import MLPClassifier

    def NeuralNetwork(X_train,X_test,y_train,y_test):
      parameters = {
        'learning_rate_init': [0.05, 0.01, 0.005, 0.001],
        'hidden_layer_sizes': [4, 8, 12],
        'activation': ["relu","logistic", "tanh"],
        'batch_size':[1000],
        'max_iter':[10000]}
      final3 = GridSearchCV(estimator=MLPClassifier(),param_grid=parameters,scoring='accuracy',cv=5)
      final4=final3.fit(X_train,y_train)
      predict_list= final4.predict(X_test)

      acc_score = accuracy_score(y_test,predict_list)
      report = classification_report(y_test,predict_list)
      return((acc_score,final4.best_params_['learning_rate_init'],final4.best_params_['hidden_layer_sizes'],final4.best_params_['activation'],'Neural Network',report
```

- 80 20 split after standardization through PCA implementation:

```
scaler = preprocessing.StandardScaler().fit(mfcc_clean_train)
mfcc_clean_train_scaled = scaler.transform(mfcc_clean_train)
print(mfcc_clean_train_scaled[0])
mfcc_clean_test_scaled = scaler.transform(mfcc_clean_test)
from sklearn.decomposition import PCA
pca = PCA(n_components = 0.95)
pca.fit(mfcc_clean_train_scaled)
mfcc_train1 = pca.transform(mfcc_clean_train_scaled)
mfcc_test1 = pca.transform(mfcc_clean_test_scaled)
print("After mfcc shape")
print(mfcc_train1[0].shape)
```

- Features extraction through librosa:



```
mfcc_list = []
chroma_list = []
spectral_centroid=[]

for idx,aud in enumerate(aud_list):
  print(idx)
  signal, sr = librosa.load(aud)
  signal = signal.flatten()
  mfccs = librosa.feature.mfcc(signal, n_mfcc=13,sr=sr) #extracting mfccs
  #now for delta and delta2 mfccs
  delta_mfcc = librosa.feature.delta(mfccs) #the delta features show how the signals vary with time, will be useful for stuff like handling reverb audio
  delta2_mfcc=librosa.feature.delta(mfccs,order=2)
  final_mfcc = np.concatenate((mfccs,delta_mfcc,delta2_mfcc))
  #scaling
  final_scaled=np.mean(final_mfcc.T,axis=0) #Scaled features,
  mfcc_list.append(final_scaled)
  chroma_cq = librosa.feature.chroma_stft(y=signal, sr=sr, n_fft=4096) #extracting chroma stft
  chroma_cq = np.mean(chroma_cq.T,axis=0)
  chroma_list.append(chroma_cq)
  cent = librosa.feature.spectral_centroid(y=signal, sr=sr) #extracting spectral centroid
  cent = np.mean(cent.T,axis=0)
  spectral_centroid.append(cent)
```