


Affective Computing
Prof. Gulshan Sharma
Department of Computer Science and Engineering
Indraprastha Institute of Information Technology, Delhi

Lecture - 15
Tutorial: Emotion Recognition by Speech Signal

Hello everyone. My name is Gulshan Sharma and I am the Teaching Assistant for this NPTEL Affective Computing course. First and foremost I would like to welcome everyone on this very first tutorial of this course. We will attempt to learn emotion recognition through speech in this tutorial.

(Refer Slide Time: 00:40)




Emotion Recognition using Speech

- Advancements in Machine Learning
 - Emotion Emotion Recognition using Speech
- Pipeline
 - Preparation of appropriate dataset
 - Selection of suitable & promising Features
 - Designing Classification Methods

With recent advances in the field of machine learning, emotion recognition via speech signal has dramatically increased. Various theoretical and experimental study have been conducted in order to identify a person's emotional state by examining their speech signals. The speech

emotion system pipeline includes repression of an appropriate dataset, selection of promising feature and design of an appropriate classification method.

(Refer Slide Time: 01:13)



The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS)

- Collection of 7356 files
- **Speech** and Songs by 24 actors (12 male and 12 female)
- Includes different emotions classes such as
 - calm, happy, sad, angry, fearful, surprise, and disgust.
- Two levels of emotional intensity
 - Normal
 - Strong
- Available in three modality formats:
 - **Audio-only (16bit, 48kHz .wav)**
 - Audio-Video (720p H.264, AAC 48kHz, .mp4)
 - Video-only (no sound)

So, in this tutorial, we will be utilizing a publicly available dataset known as RAVDESS. The RAVDESS dataset consists of 7356 files. The database includes speeches and songs from 24 actors, 12 male and 12 female. Emotion classes include calm, happiness, sadness, anger, fear, surprise and disgust. And this dataset is available in 3 formats audio only, video only, and audio and video. So, for our task, we will be used only speech power that is our audio only files.

(Refer Slide Time: 01:57)




Filename Identifiers: 03-01-01-01-01-01-01.wav

- Modality (01 = full-AV, 02 = video-only, 03 = audio-only).
- Vocal channel (01 = speech, 02 = song).
- Emotion (01 = neutral, 02 = calm, 03 = happy, 04 = sad, 05 = angry, 06 = fearful, 07 = disgust, 08 = surprised).
- Emotional intensity (01 = normal, 02 = strong). NOTE: There is no strong intensity for the 'neutral' emotion.
- Statement (01 = "Kids are talking by the door", 02 = "Dogs are sitting by the door").
- Repetition (01 = 1st repetition, 02 = 2nd repetition).
- Actor (01 to 24. Odd numbered actors are male, even numbered actors are female).

Now, moving towards the file name convention in this dataset, the file name in this dataset consists of 7 identifiers where the first identifier tell us about the modality either it is a full audio video file, video only file or audio only file. The second identifier will tell about the vocal channel, it is either a speech file or a song file. The third and the most important identifier is our emotion identifier, which will tell about the class of the emotion, neutral, calm, happy, sad, angry, fearful, disgust or surprised.

The fourth one is the emotional intensity either the emotion is of normal intensity or the strong intensity. Later on, fifth identifier will tell about the statement. And sixth will tell about the repetition of that statement. And the seventh identifier will tell about the actor. Odd number actors are male and even number actors are female.

(Refer Slide Time: 02:55)



Tutorial Overview

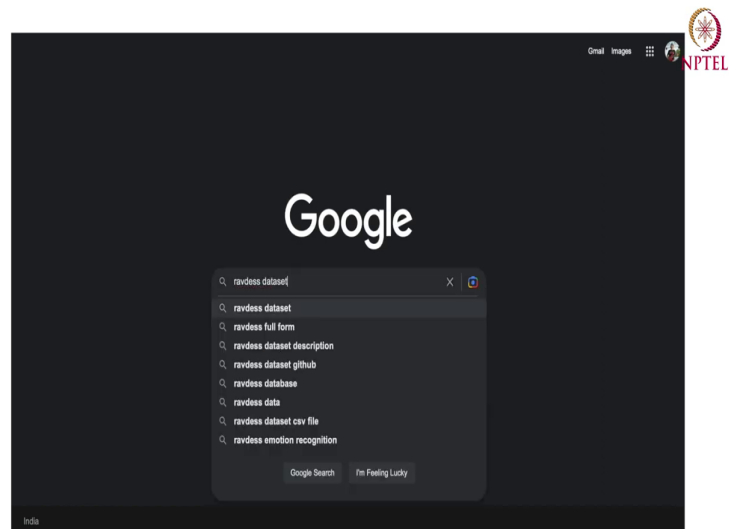
- Downloading Dataset
- Import dataset into Google Drive
- Following Experiments on Google Colab
 - Reading Audio File in Python
 - Feature Extraction in Python
 - Fundamental Frequency
 - Zero Crossing Rate
 - Mel Frequency Cepstral Coefficients (MFCC)
 - Employing ML Classification
 - GNB
 - LDA
 - SVM
 - 1D-CNN on Raw Audio

So, before starting the coding part, let me first give you the complete overview of this tutorial. We will start with downloading the dataset. After downloading the dataset we will import that dataset into a Google Drive. The reason behind importing the dataset into Google Drive is that we will be using Google Colab for our experimentations.

And our experimentation will start with reading audio file in Python. Then, we will extract these fundamental frequency zero cross rates and Mel Frequency Cepstral Coefficient as our features from the audio files. And after that we will be employing some of the classification algorithms like Gaussian Naive Bayes, Linear Discriminant Analysis and Support Vector Machine.

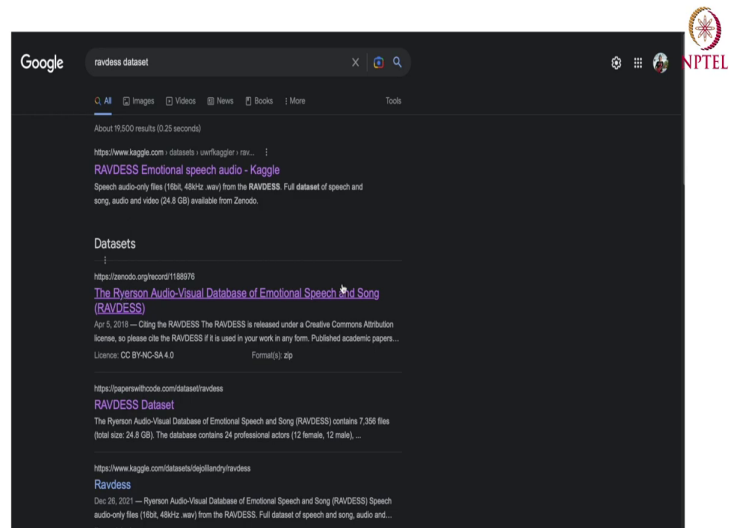
And in the end, we will also try to create a 1-dimensional convolution neural network over the raw audio for the emotion classification.

(Refer Slide Time: 04:06)



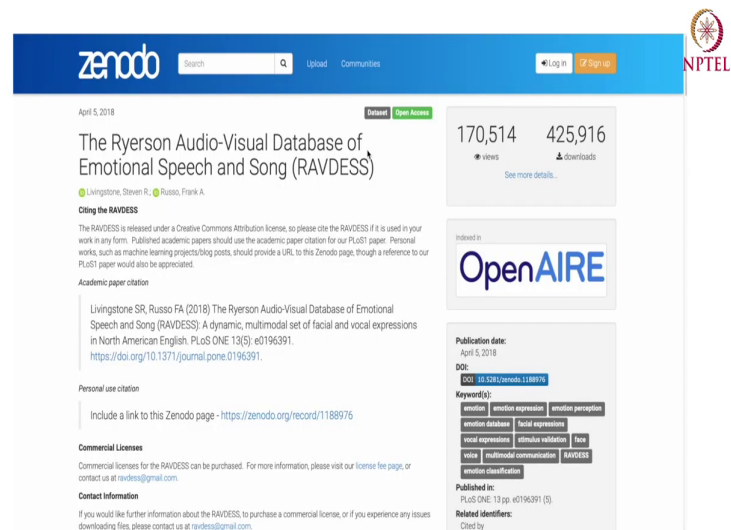
So, starting with our very first exercise which is dataset download, we can download this dataset by simply searching the RAVDESS on Google.

(Refer Slide Time: 04:10)



After putting this RAVDESS keyword on Google, you will find this zenodo link over here. So, basically, zenodo is a general purpose open access repository. Here we can store our data up to 50 GBs.

(Refer Slide Time: 04:29)



The screenshot shows the Zenodo record page for the Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDSS). The page features a blue header with the Zenodo logo, a search bar, and navigation links for 'Upload' and 'Communities'. On the right side of the header, there are 'Log in' and 'Sign up' buttons. The main content area includes the title 'The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDSS)', the authors 'Livingstone, Steven R.' and 'Russo, Frank A.', and a 'Citing the RAVDESS' section. The 'Citing the RAVDESS' section provides information on how to cite the dataset in academic papers and personal use. The 'Academic paper citation' section includes the following text: 'Livingstone SR, Russo FA (2016) The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDSS). A dynamic, multimodal set of facial and vocal expressions in North American English. PLoS ONE 13(5): e0196391. https://doi.org/10.1371/journal.pone.0196391.' The 'Personal use citation' section includes the text: 'Include a link to this Zenodo page - https://zenodo.org/record/1188976'. The 'Commercial licenses' section provides information on how to purchase a commercial license. The 'Contact information' section provides contact details for the dataset. On the right side of the page, there are statistics showing 170,514 views and 425,916 downloads. Below the statistics, there is a 'See more details' link. The page also features an 'OpenAIRE' logo and a 'Publication date' of April 5, 2018. The 'DOI' is 10.5281/zenodo.1188976. The 'Keywords' section includes terms such as 'emotion', 'emotion expression', 'emotion perception', 'emotion database', 'facial expressions', 'vocal expressions', 'stimulus validation', 'face', 'voice', 'multimodal communication', and 'RAVDSS'. The 'Published in' section includes the text: 'PLOS ONE: 13 pp. e0196391 (5)'. The 'Related identifiers' section includes the text: 'Cited by'.

So, we can simply click on this link, and find the dataset.

(Refer Slide Time: 04:32)

Commercial Licenses
Commercial licenses for the RAVeSS can be purchased. For more information, please visit our license fee page, or contact us at ravedes@gmail.com.

Contact Information
If you would like further information about the RAVeSS, to purchase a commercial license, or if you experience any issues downloading files, please contact us at ravedes@gmail.com.

Example Videos
Watch a sample of the RAVeSS speech and song videos.

Emotion Classification Users
If you're interested in using machine learning to classify emotional expressions with the RAVeSS, please see our new RAVeSS Facial Landmark Tracking data set [Zenodo project page].

Construction and Validation
Full details on the construction and perceptual validation of the RAVeSS are described in our PLoS ONE paper - <https://doi.org/10.1371/journal.pone.0196391>.

The RAVeSS contains 7356 files. Each file was rated 10 times on emotional validity, intensity, and genuineness. Ratings were provided by 247 individuals who were characteristic of untrained adult research participants from North America. A further set of 72 participants provided test-retest data: high levels of emotional validity, moderate reliability, and test-retest intrasubject reliability were reported. Validation data is open-access, and can be downloaded along with our paper from PLoS ONE.

Description
The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVeSS) contains 7356 files (total size: 24.8 GB). The database contains 24 professional actors (12 female, 12 male), vocalizing two lexically-matched statements in a neutral North American accent. Speech includes calm, happy, sad, angry, fearful, surprise and disgust expressions, and song contains calm, happy, sad, angry, and fearful emotions. Each expression is produced at two levels of emotional intensity (normal, strong), with an additional neutral expression. All conditions are available in three modality formats: Audio-only (11bit, 48kHz, wav), Audio-Video (720p H.264, AAC 48kHz, mp4), and Video-only (no sound). Note, there are no song files for Actor_18.

Audio-only files
Audio-only files of all actors (01-24) are available as two separate zip files (~200 MB each).

- Speech file (Audio_Speech_Actors_01-24.zip, 215 MB) contains 1440 files: 60 trials per actor x 24 actors = 1440.
- Song file (Audio_Song_Actors_01-24.zip, 198 MB) contains 1012 files: 44 trials per actor x 23 actors = 1012.

Audio-Visual and Video-only files

Navigation: vocal expressions, stimulus validation, face, voice, multimodal communication, RAVeSS, emotion classification

Published in: PLoS ONE: 13 pp. e0196391 (5)

Related identifiers:
Cited by: 10.1371/journal.pone.0196391
Referenced by: 10.5281/zenodo.3255102

Communities:
The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVeSS)
Zenodo

License (for files):
CC Creative Commons Attribution Non-Commercial Share Alike 4.0 International

Versions

Version	Date
Version 1.0.0	Apr 5, 2018
10.5281/zenodo.1188176	

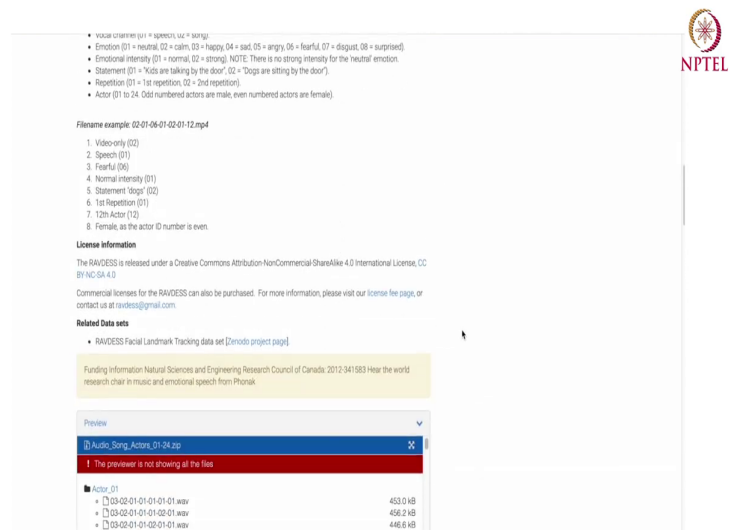
Cite all versions! You can cite all versions by using the DOI 10.5281/zenodo.1188176. This DOI represents all versions, and will always resolve to the latest one. Read more.

Share

Cite as

So, this dataset is basically released under creative common attribute license. So, one can openly use it for the publications.

(Refer Slide Time: 04:42)



• vocal channel (v) = speech, (u) = song
• Emotion (01 = neutral, 02 = calm, 03 = happy, 04 = sad, 05 = angry, 06 = fearful, 07 = disgust, 08 = surprised)
• Emotional intensity (01 = normal, 02 = strong). NOTE: There is no strong intensity for the 'neutral' emotion.
• Statement (01 = "Kids are talking by the door", 02 = "Dogs are sitting by the door")
• Repetition (01 = 1st repetition, 02 = 2nd repetition)
• Actor (01 to 24: Odd numbered actors are male, even numbered actors are female)

Filename example: 02-01-06-01-02-01-12.mp4

1. Video-only (02)
2. Speech (01)
3. Fearful (06)
4. Normal intensity (01)
5. Statement 'dogs' (02)
6. 1st Repetition (01)
7. 12th Actor (12)
8. Female, as the actor ID number is even.

License information
The RAVeSS is released under a Creative Commons Attribution-NonCommercial ShareAlike 4.0 International License, CC BY-NC-SA 4.0
Commercial licenses for the RAVeSS can also be purchased. For more information, please visit our license fee page, or contact us at rauess@gmail.com.

Related Data sets

- RAVeSS Facial Landmark Tracking data set [Emojo project page]

Funding information: Natural Sciences and Engineering Research Council of Canada, 2012-341883 Hear the world research chair in music and emotional speech from Phonak.

Preview

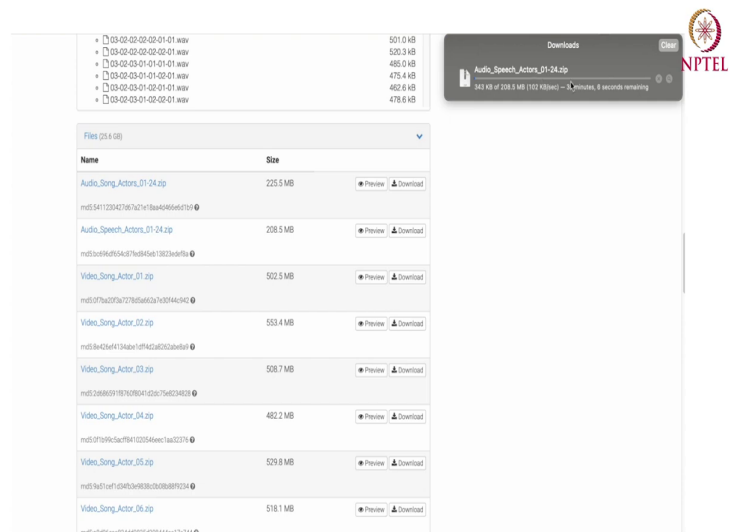
Audio_Song_Actors_01-04.zip

The previewer is not showing at the first

Actor_01	
03-02-01-01-01-01-01.wav	453.0 KB
03-02-01-01-01-02-01.wav	456.2 KB
03-02-01-01-02-01-01.wav	446.6 KB

And to download the dataset, and to download the our exact part which is audio speech actor dataset we can simply click on this link.

(Refer Slide Time: 04:47)



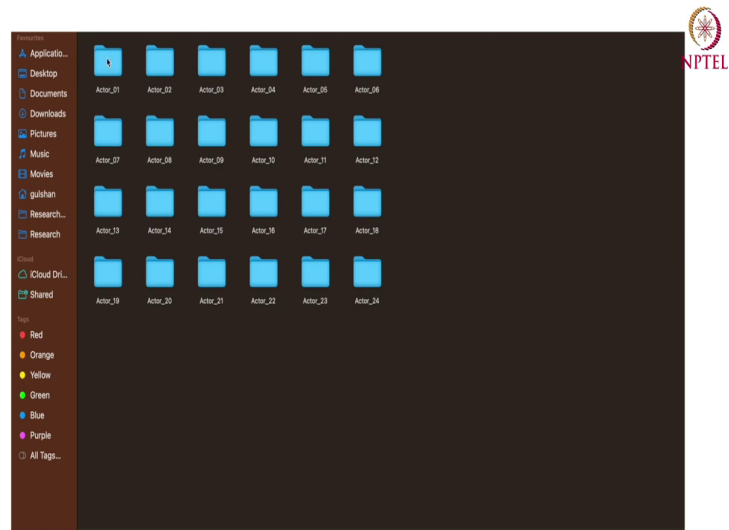
The screenshot displays a file explorer interface with a list of files and a download progress bar. The file list includes:

Name	Size	Actions
Audio_Song_Actors_01-24.zip	225.5 MB	Preview Download
Audio_Speech_Actors_01-24.zip	208.5 MB	Preview Download
Video_Song_Actor_01.zip	502.5 MB	Preview Download
Video_Song_Actor_02.zip	553.4 MB	Preview Download
Video_Song_Actor_03.zip	528.7 MB	Preview Download
Video_Song_Actor_04.zip	482.2 MB	Preview Download
Video_Song_Actor_05.zip	529.9 MB	Preview Download
Video_Song_Actor_06.zip	518.1 MB	Preview Download

The download progress bar at the top right shows the file 'Audio_Speech_Actors_01-24.zip' is being downloaded at 342 KB of 208.5 MB (102 KB/sec) with 8 seconds remaining. The NPTEL logo is visible in the top right corner.

It will take some time to download, but in my case I have already downloaded this dataset. And I can show you.

(Refer Slide Time: 05:05)



After unzipping the downloaded file, this dataset will look something like this. So, there will be 24 folders each belonging to one actor.

(Refer Slide Time: 05:19)



And after going through one folder, we will have a couple of files over here. Or maybe I can just play a couple of files just for your reference.

Dogs are sitting by the door. Dogs are sitting by the door. Dogs are sitting by the door. Dogs are sitting by the door. Dogs are sitting by the door.

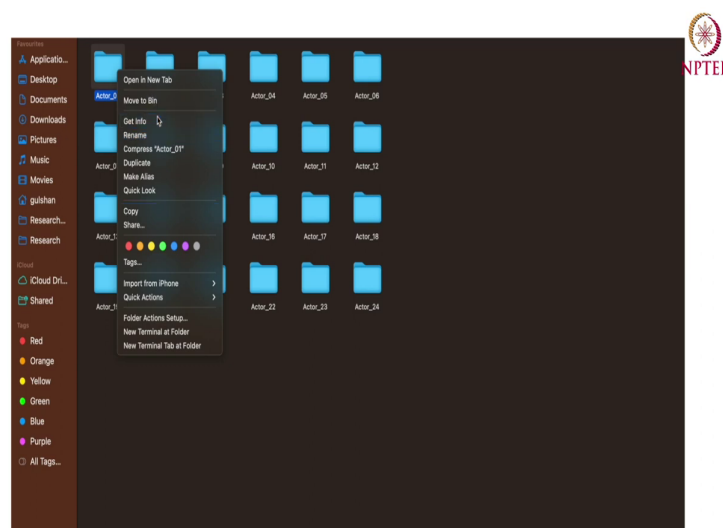
So, as you can see there are multiple emotions saying this line, dogs are sitting on the door. And if I move to some other folder, let us say actor 2 folder and play a couple of files.

Kids are talking by the door. Kids are talking by the door. Kids are talking by the door. Kids are talking by the door.

So, as we can see like there are a couple of variations in this speaking style representing different different type of emotions. So, as we have downloaded this dataset, now our next task is to upload it on Google Drive, so that we can easily access it through a Google Colab. I believe most of us can easily upload a folder on Google Drive. So, I will be skipping that part.

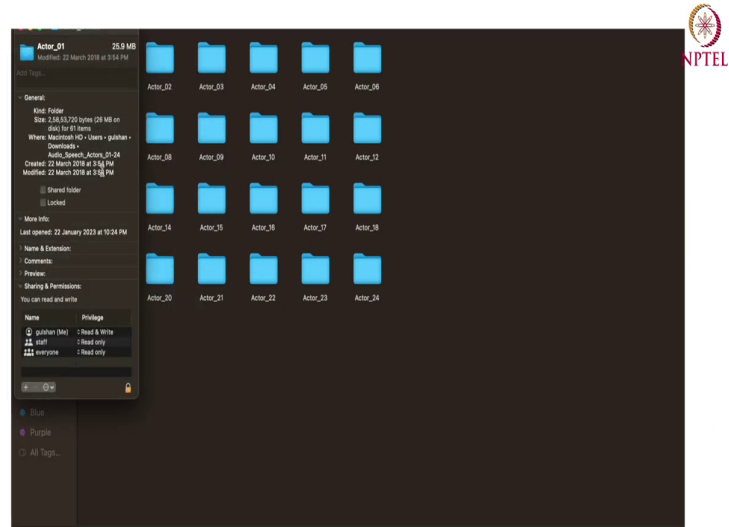
But with some of the participants it could be a situation that they are having a low bandwidth internet connection, these participants can take any of the folder and upload it on a Google Drive. So, let us suppose you are taking folder number 1.

(Refer Slide Time: 07:10)



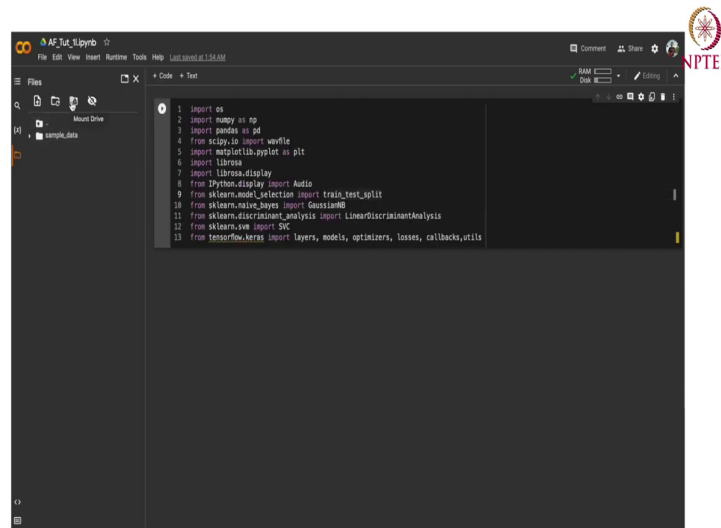
So, folder number 1, I believe is of 25.9 MB. So, it will not be a very big file to upload on a Google Drive.

(Refer Slide Time: 07:14)



So, now, I will shift on the Google Colab and we will start writing a program for emotion recognition.

(Refer Slide Time: 07:33)

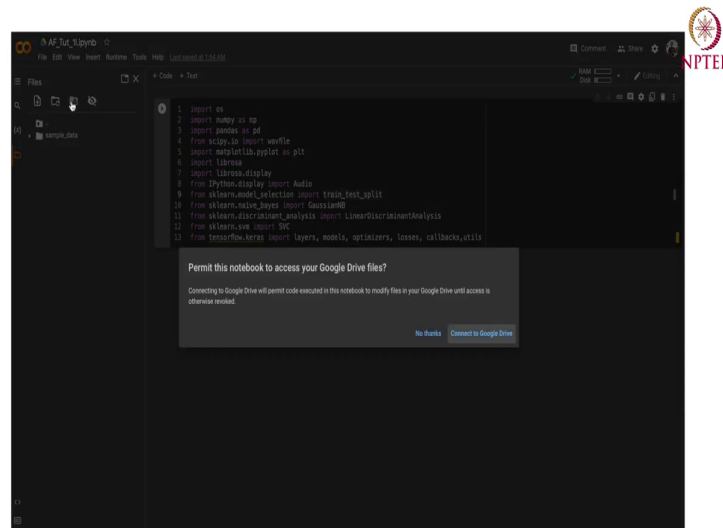


```
1 import os
2 import numpy as np
3 import pandas as pd
4 from scipy.io import wavfile
5 import matplotlib.pyplot as plt
6 import librosa
7 import librosa.display
8 from IPython.display import Audio
9 from sklearn.model_selection import train_test_split
10 from sklearn.naive_bayes import GaussianNB
11 from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
12 from sklearn.svm import SVC
13 from tensorflow.keras import layers, models, optimizers, losses, callbacks, utils
```

So, before starting our programming exercise, so before starting our programming exercise I assume that everyone has some sort of experience with Python programming language and everyone is aware of Google Colab Interface. We will start this exercise with importing couple of libraries and the helping functions. To save some time I have already copied required import code. So, everyone who is programming along with me can pause this video and write this code in their own environment.

So, after importing the required libraries and helping functions, we will start with reading the audio files using Python, but before that we need to mount Google Drive with our Colab Interface. To do so, we will first click on this files icon over here and then select mount drive option.

(Refer Slide Time: 08:44)



The screenshot shows a Jupyter Notebook environment with a dark theme. The top menu bar includes 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. The left sidebar shows a file explorer with a folder named 'sample_data'. The main area contains a code cell with the following Python code:

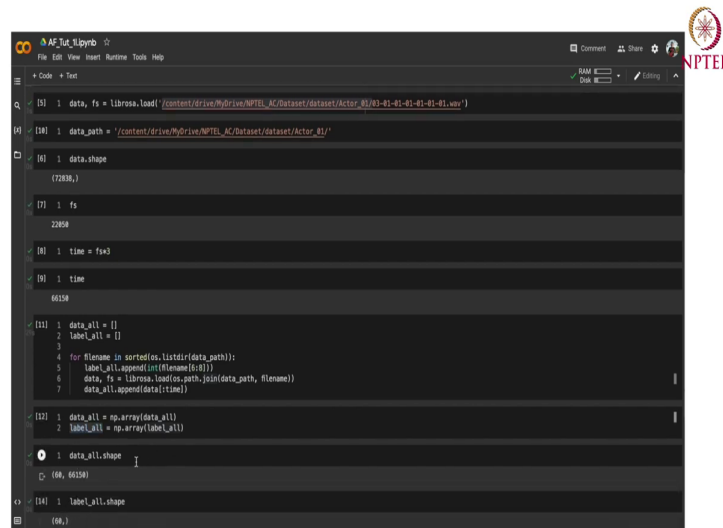
```
1 import os
2 import numpy as np
3 import pandas as pd
4 from sklearn import svm
5 import matplotlib.pyplot as plt
6 import librosa
7 import librosa.display
8 from IPython.display import Audio
9 from sklearn.model_selection import train_test_split
10 from sklearn.metrics import confusion_matrix
11 from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
12 from sklearn.svm import SVC
13 from tensorflow.keras import layers, models, optimizers, losses, callbacks, utils
```

A dialog box is overlaid on the code, asking for permission to access Google Drive files. The dialog text reads: "Permit this notebook to access your Google Drive files? Connecting to Google Drive will permit code executed in this notebook to modify files in your Google Drive and access a database if needed." There are two buttons: "No Thanks" and "Connect to Google Drive".

After pressing this button, you will find a dialog box over here asking for permission to access Google Drive. So, we will simply click on connect to Google Drive. So, after mounting Google Drive with Colab environment, we will now import the data. I am also assuming that some of the participant does not have enough powerful machine or high speed internet connection.

So, to simplify our job, I will be using data from a single subfolder. Since, we are importing audio data, so to read audio data in our Python environment, I will be using librosa dot load function.

(Refer Slide Time: 09:27)



```
AF Tut_11.ipynb
File Edit View Insert Runtime Tools Help
Code Text
Comment Share
RAM
Data

[5] 1 data, fs = librosa.load('/content/drive/MyDrive/NPTEL_AC/dataset/dataset/Actor_01/03-01-01-01-01.wav')

[6] 1 data_path = '/content/drive/MyDrive/NPTEL_AC/dataset/dataset/Actor_01/'

[6] 1 data.shape
(72838,)

[7] 1 fs
22050

[8] 1 time = fs*3

[9] 1 time
66150

[11] 1 data_all = []
2 label_all = []
3
4 for filename in sorted(os.listdir(data_path)):
5     label_all.append(int(filename[0]))
6     data, fs = librosa.load(os.path.join(data_path, filename))
7     data_all.append(data[:time])

[12] 1 data_all = np.array(data_all)
2 label_all = np.array(label_all)

[13] 1 data_all.shape
(66, 66150)

[14] 1 label_all.shape
(66,)
```

I will create two variable called data and sample rate, then I will simply write librosa dot load and inside the brackets I will pass the I will pass the file name. So, as you can say this function has successfully run, and now I will show you the shape of the data. It consists of 72838 sample values and our sample rate is 22,050.

So, for more simplification I am planning to use just first 3 seconds of our audio data. So, calculating first 3 seconds of data, I can simply multiply our fs by 3, our sample rate by 3 and we will get the exact time. So, first 3 seconds of time. So, it will be equivalent to 66150 samples.

Now, we have imported just one data. So, we need to import all the data inside this folder. To do so, I need to write a piece of code where I will be sequentially reading all the files and saving them into a Python list. So, let us start with our code, I will name my variable as data

all I will be also extracting all the labels. Since, we have already seen that data file name contains their respective label. So, we need to extract the relevant label also.

So, I will start with a loop where I will be reading all the file names in a sorted function from `os dot list dir` and in this list `dir` I will be importing that data file, I mean the data path of the data folder. So, maybe I can just create a another variable which is data path and this will be treated as a string and I need to write the exact data path for this folder. So, I will simply copy it from this place and paste it over here, ok.

Now, I can simply use this variable wherever I want this data path. So, first I will try to extract all the labels. So, for that I will be pending all the labels, in this label all list and I will simply read the file name and extract the substring, does that sub identify from that particular file name. Maybe I will I also need to write classes to integer as these little bit treated as label, so that that could approach.

Now, I can simply read my data and also my sample rate `limb librosa dot load`. And now, now this line; this line of code will simply read all the file names and append it with the our data path and then the `librosa` function will read that corresponding file. Now, I need to store all those files in our data, all list. So, to do so, I will simply write all append data and also I will be you know simply using first 3 second. So, will I will put a colon, and sorry I will put a colon and type the time over here.

So, let me just run this code. It might take a couple of seconds to run this file completely, ok. So, code has been executed now. So, I will simply convert these list into numpy array. So, to do so, I will simply write data all equal to `np dot array` and label all be become sorry, this is a mistake over here. I need to write underscore not hyphen.

So, after converting these list into numpy arrays I can simply try to see their shape, what are their exact shapes. So, I will simply write data all dot shape, ok. So, now, we have written 60 files each consisting of 66150 sample which corresponding to 3 seconds of initial data. I can

Kids are talking by the door.

So, as you can see using this utility, I can simply play the exact audio file in my Colab. Maybe I can play one more file over here.

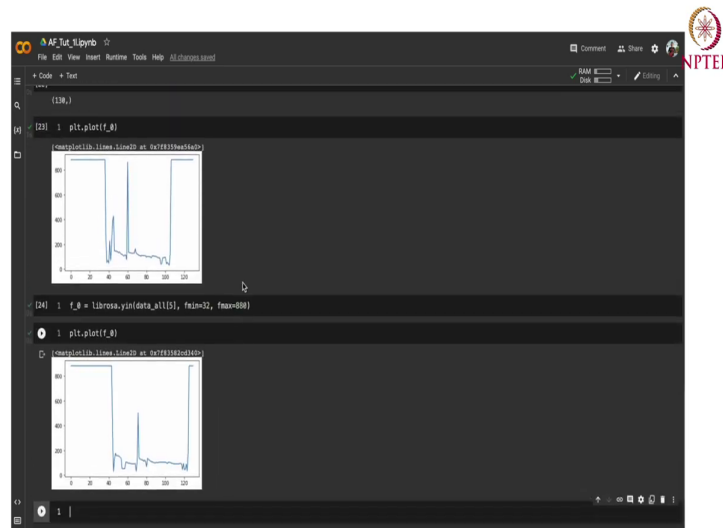
Kids are talking by the door.

Sounds good. So, after importing our data, I will now move towards some sort of a particular feature extraction phase. And in our feature extraction phase, we will be simply using fundamental frequency, and a 0 cost rate, Mel-frequency cepstral coefficients as our basic features.

So, let me show you how to extract a fundamental frequency from this audio files. And to extract the fundamental frequency, I will simply make a I am so sorry; I will simply make a variable f_0 . And I will be using a library function called librosa dot yin. And in this function, I just need to pass a data instance with a range of, range of frequency value like minimum frequency value and the highest frequency value.

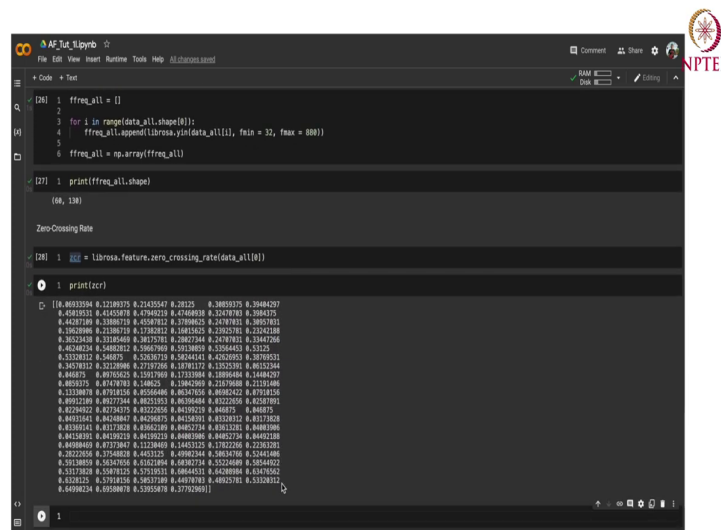
So, let me just extract fundamental frequency for a single instance, then I will show you how to do it for a whole folder, ok.

(Refer Slide Time: 19:36)



So, yeah, as you can see that initial values are somewhat around 882, then there was some sort of a variation in this part and then again it is going to 882 over here. Or, maybe I just need to show you another file, let us say 5 and then I will print another plot over here. So, as you can see over here that there is some sort of a difference between the fundamental frequencies in two different emotions.

(Refer Slide Time: 20:25)



```
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
[26]: 1 ffreq_all = []
      2
      3 for i in range(data_all.shape[0]):
      4     ffreq_all.append(librosa.yin(data_all[i], fmin = 32, fmax = 888))
      5
      6 ffreq_all = np.array(ffreq_all)

[27]: 1 print(ffreq_all.shape)
(60, 138)

Zero-Crossing Rate
[28]: 1 zcr = librosa.feature.zero_crossing_rate(data_all[0])

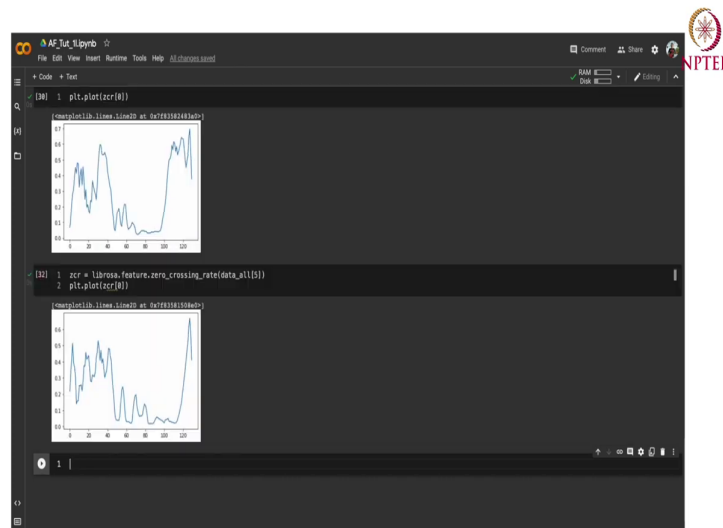
[ ] 1 print(zcr)
[[ 0.06532594  0.12189375  0.21435447  0.28125    0.38859375  0.50484297
  0.49829531  0.45458718  0.47949219  0.47408938  0.32478763  0.39843735
  0.46287389  0.33883789  0.45887812  0.37898625  0.24787961  0.38933803
  0.39289898  0.23388719  0.13828212  0.18825625  0.23825761  0.23242188
  0.35232158  0.23849689  0.38173761  0.28927344  0.24787961  0.29474796
  0.46248234  0.54882812  0.58667869  0.59138693  0.53564451  0.53752777
  0.53288212  0.54882812  0.58262719  0.58484444  0.45232959  0.39789551
  0.36738132  0.35139386  0.27379766  0.17111717  0.12552913  0.86153344
  0.8688735    0.89755823  0.12917869  0.17323984  0.18916484  0.16884297
  0.8955375    0.87187918  0.18625    0.38867869  0.75379688  0.11325486
  0.13338878  0.87328126  0.82566486  0.86387956  0.86834422  0.87318156
  0.89323189  0.89777384  0.89325761  0.86387956  0.83232959  0.52527901
  0.82294932  0.82734375  0.83228556  0.84139219  0.8688735    0.8688735
  0.89323189  0.84168844  0.84396219  0.84168844  0.83232959  0.83232959
  0.83892141  0.83173828  0.83662189  0.84622734  0.83613261  0.84883986
  0.84168844  0.84168844  0.84168844  0.84168844  0.84168844  0.84168844
  0.84888469  0.87377847  0.11288469  0.14433325  0.17822266  0.22363281
  0.28222656  0.37188219  0.4453125    0.49862344  0.58234766  0.54424486
  0.59138693  0.55494926  0.51823984  0.48397734  0.52524688  0.56448492
  0.53733828  0.53878125  0.57519531  0.68644531  0.64238984  0.63471656
  0.5282325    0.52918261  0.58373789  0.4837893    0.48527171  0.51243192
  0.64998234  0.69588878  0.53958878  0.37792891]]
```

Now, let me simply extract the fundamental frequency for all the data. For that I will simply write another list where; to do so, I will be simply using a for loop, where I will iterate over all the data and extract our fundamental frequency. So, I will simply append this. Later I will convert it into a and by later I will; later I will convert it to a numpy array, this exact list. This is copied, see it sometime.

Let me print the shape of our; now, let me print the shape of this variable, ok. So, we have selected the fundamental frequency for each and every file in the folder 1. Now, I will go with another sort of feature known as our 0 cross ratings. So, I will be extracting 0 cross rate over here. So, to do so, I will be again using librosa library, and there is a function in it called 0 crossing rate which will be giving us the exact 0 cross rating corresponding to these audio files.

So, let me use the variable name as zcr and then I can use, then I can write; let me show you with the single file first. (Refer Time: 23:49) it is working. Let me show the print. Let me print the zcr, ok. So, you can see there is some sort of differences over here. Maybe I can give you a better visualization by simply plotting this over the time.

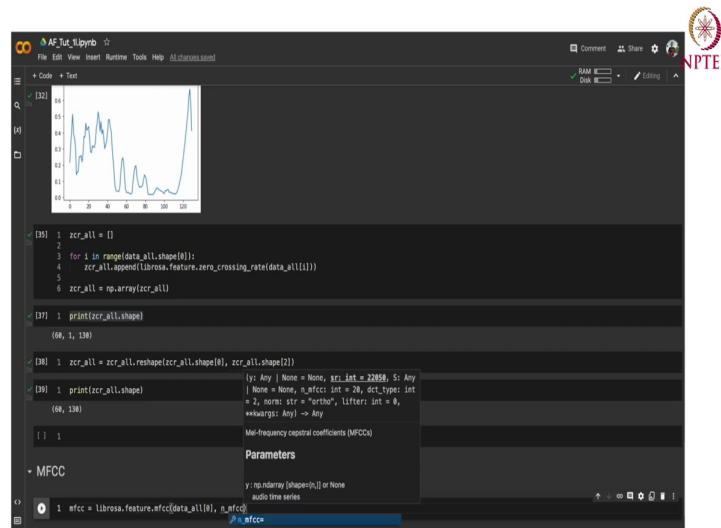
(Refer Slide Time: 24:11)



So, for that I will be writing plot, ok. Maybe I will create the zcr for another emotion. See for emotion number 5 and then plot, ok. There is an issues over here, ok. It is not a cr, it is zcr.

Yeah, here we can also observe that there is some significant amount of differences between these two features. So, I will simply write another code to you know extract the zcr value for all the files.

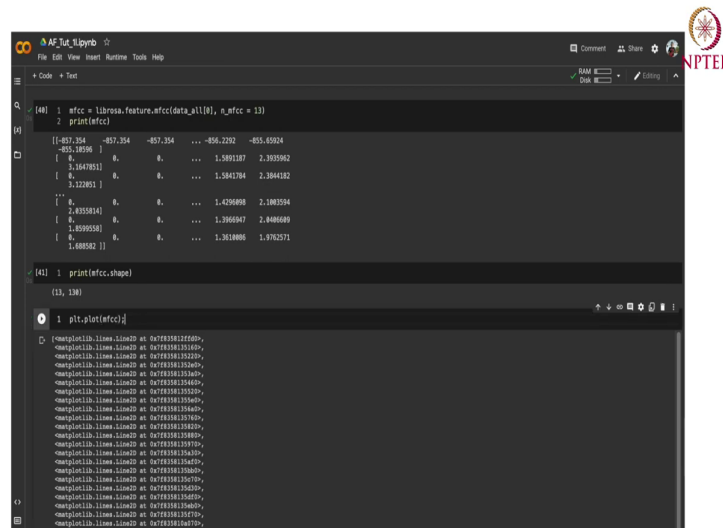
(Refer Slide Time: 25:19)



Now, maybe to just keep it consistent with my previous feature shape which was 16 to 130, you can also reshape this zcr all function. And now, if I run my print function again, so yeah we will got a similar shape over here. This is just to keep back the consistency among the all the feature. Now, I will show you two extract another feature called Mel-frequency cepstral coefficients.

So, for that maybe I just need to let us write MFCC separate over here. And yeah, I can simply extract MFCC from librosa. And yeah, there is a parameter in MFCC, it is called number of MFCC coefficient. In my case, let us say, we will be extracting first 13 coefficients, yeah.

(Refer Slide Time: 26:27)



The screenshot shows a Jupyter Notebook interface with the following code and output:

```
[40] 1 mfcc = librosa.feature.mfcc(data_all10, n_mfcc = 13)
      2 print(mfcc)
```

```
[[-857.354 -857.354 -857.354 ... -856.2202 -855.45024
  [-855.38596 ] 0. 0. ... 1.5891187 2.3035962
  [ 3.1647851 ] 0. 0. ... 1.3843784 2.3844182
  [ 3.122851 ] 0. 0. ... 1.4296898 2.1883594
  [ 2.4338344 ] 0. 0. ... 1.3966947 2.4486689
  [ 1.8209558 ] 0. 0. ... 1.3618806 1.9762571
  [ 1.688582 ] 0. 0. ... 1.3618806 1.9762571]]
```

```
[41] 1 print(mfcc.shape)
```

```
(13, 130)
```

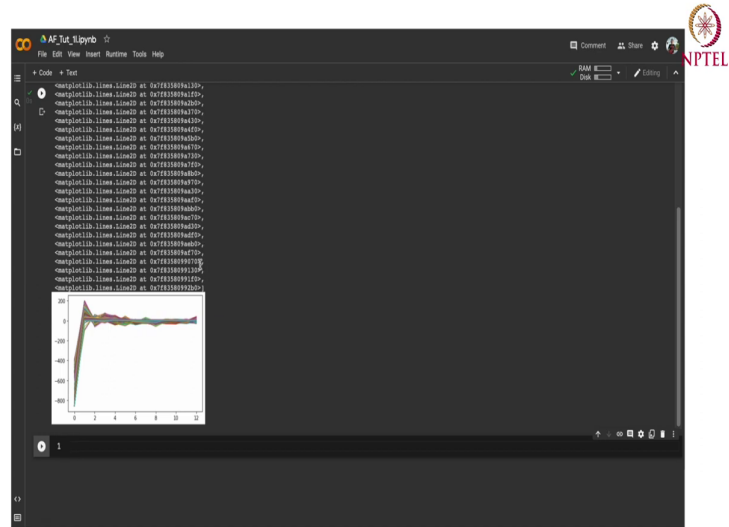
```
0 1 plt.plot(mfcc)
```

The output shows a 13x130 array of MFCC coefficients. The visualization area shows a plot of these coefficients, with the x-axis representing the 130 coefficients and the y-axis representing the 13 rows.

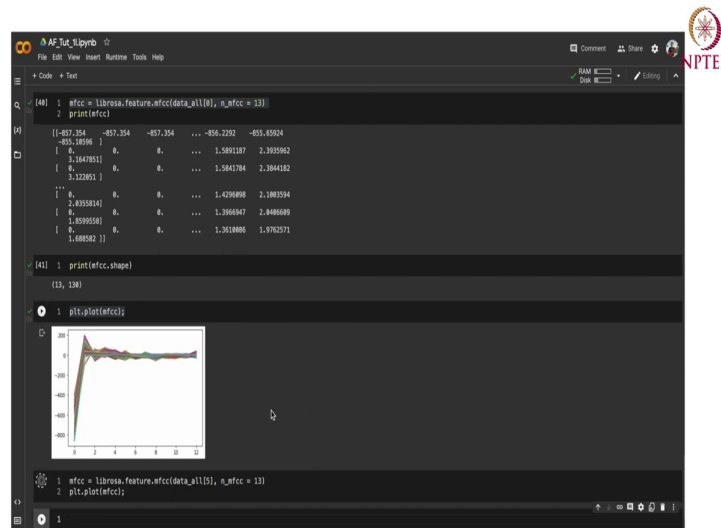
So, these are MFCC coefficients. I will like to print its shape also. So, yeah, there are 13 cross 130 for a single (Refer Time: 26:42). So, in this case, as we are getting 13 rows and 130 columns. So, for each row there are 130 values and each of these 13 values corresponding to one MFCC coefficient.

Now, to visualize this I can simply type and I can simply write MFCC, ok or maybe just to avoid these values I can simply put a semicolon.

(Refer Slide Time: 27:12)

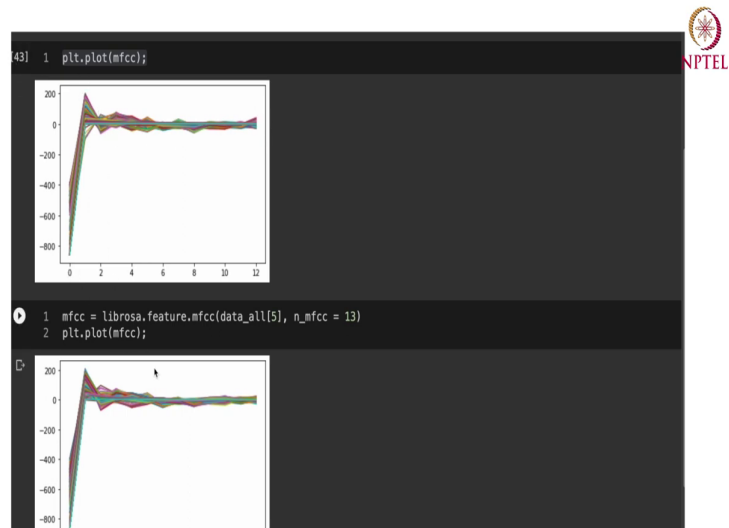


(Refer Slide Time: 27:23)



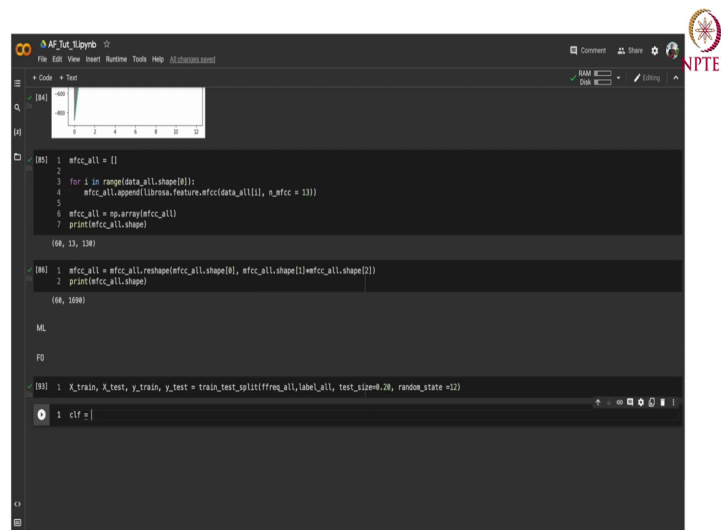
Now, let me plot it for another file. Let us see for file number I mean in the 5th file and our MFCC coefficient will look something like this. So, there is some significant differences over here.

(Refer Slide Time: 27:42)

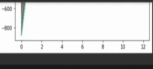


I believe yes, I can see some differences over here. There are some differences over here also. And since, it is a very complex and very tightly bounded lines, but yeah there are some significant differences over these two files. So, now, again we will simply extract all these MFCC for all the files.

(Refer Slide Time: 28:17)



The screenshot shows a Jupyter Notebook interface with the following code and output:

```
[341] 
```

```
[351] 1 mfcc_all = []
2
3 for i in range(data_all.shape[0]):
4     mfcc_all.append(librosa.feature.mfcc(data_all[i], n_mfcc = 13))
5
6 mfcc_all = np.array(mfcc_all)
7 print(mfcc_all.shape)
(68, 13, 1307)
```

```
[361] 1 mfcc_all = mfcc_all.reshape(mfcc_all.shape[0], mfcc_all.shape[1]*mfcc_all.shape[2])
2 print(mfcc_all.shape)
(68, 1698)
```

ML

```
[371] 1 X_train, X_test, y_train, y_test = train_test_split(mfcc_all, label_all, test_size=0.20, random_state =12)
```

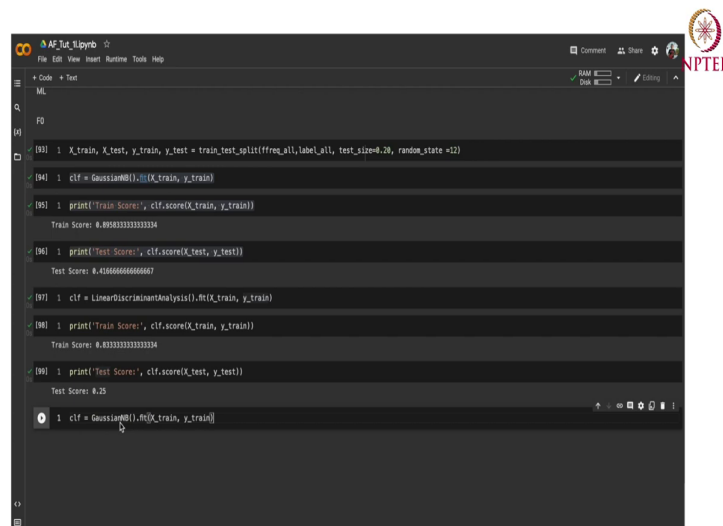
```
[381] 1 clf =
```

So, now yeah it looked consistent with my prior representations. And now, we have extracted all 3 features of each basic features like MFCC fundamental frequency and 0 cost rating. Now, after this I will be using my basic machine learning inferences. In my machine learning algorithms, I will be using Gaussian Naive Bayes, linear discriminant analysis and support vector machines.

So, now, moving towards the one machine learning part, in machine learning part we will take one feature divide it into their respective train and test parts, and then run our classifier over it. So, starting with the our very first feature which is fundamental frequency. So, let me first divide it into their train and test part. So, for this division, I will be using train test split function from sklearn library. So, code will look something like this, ok.

Now, I can simply run my classifier as `clf` equal to see my first classifier which is let me show you, Gaussian Naive Bayes. So, I have already inputted like from as `sklearn dot Naive Bayes import Gaussian Naive Bayes`.

(Refer Slide Time: 29:37)



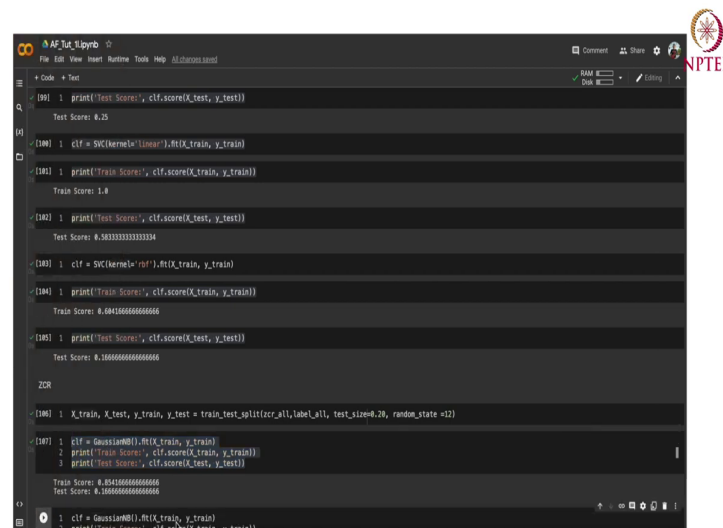
```
File Edit View Insert Runtime Tools Help
+ Code + Text
ML
FO
[93] 1 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2)
[94] 1 clf = GaussianNB().fit(X_train, y_train)
[95] 1 print('Train Score:', clf.score(X_train, y_train))
Train Score: 0.8928333333333334
[96] 1 print('Test Score:', clf.score(X_test, y_test))
Test Score: 0.4166666666666667
[97] 1 clf = LinearDiscriminantAnalysis().fit(X_train, y_train)
[98] 1 print('Train Score:', clf.score(X_train, y_train))
Train Score: 0.8333333333333334
[99] 1 print('Test Score:', clf.score(X_test, y_test))
Test Score: 0.25
100 1 clf = GaussianNB().fit(X_train, y_train)
```

So, I will simply copy it over here and function over here and fit it on my training set, ok. Now, my classifier is fit on our training set. So, let me check about the training accuracy over here, ok. So, we are getting 89 percent of training accuracy. And let me also check for the testing score, training accuracy I will get, ok; so, I am getting 83 percent of training accuracy over here using `lta` which is lesser than our Gaussian Naive Bayes.

And let me also check out my test score, ok. So, yeah test score is also getting down to 0.25 percent. So, I believe Gaussian Naive Bayes is performing better in our fundamental frequency. So, guys let me try my third classifier now which is support vector machine. So,

for that I will also again reuse my code. And instead of Gaussian Naive Bayes, I will be using our SVC function over here.

(Refer Slide Time: 30:53)



```
1 print("Test Score:", clf.score(X_test, y_test))
Test Score: 0.25

1001 | clf = SVC(kernel='linear'), fit(X_train, y_train)
1002 | print("Train Score:", clf.score(X_train, y_train))
Train Score: 1.0

1003 | print("Test Score:", clf.score(X_test, y_test))
Test Score: 0.5833333333333334

1004 | clf = SVC(kernel='rbf'), fit(X_train, y_train)
1005 | print("Train Score:", clf.score(X_train, y_train))
Train Score: 0.6816666666666666

1006 | print("Test Score:", clf.score(X_test, y_test))
Test Score: 0.16666666666666666

ZCR

1007 | X_train, X_test, y_train, y_test = train_test_split(xcr_all, label_all, test_size=0.2, random_state=12)

1008 | clf = GaussianNB(), fit(X_train, y_train)
1009 | print("Train Score:", clf.score(X_train, y_train))
1010 | print("Test Score:", clf.score(X_test, y_test))
Train Score: 0.8516666666666666
Test Score: 0.16666666666666666

1 | clf = GaussianNB(), fit(X_train, y_train)
2 | print("Train Score:", clf.score(X_train, y_train))
```

So, I will replace Gaussian Naive with SVC and inside SVC I need to define my kernel, which kernel I will be using. So, kernel equal to let us say we start with a linear kernel and let me check, ok yeah. So, classifier is set on our training data. Let us see about our train score, ok. So, with unit classifier we are getting 100 percent training accuracy. Let me check the test score over here. We are getting 58 percent of accuracy which is you know higher than all of other classifiers.

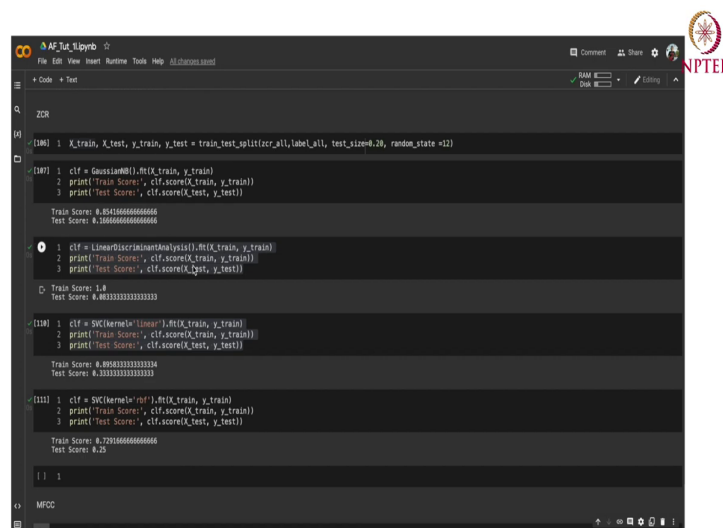
So, maybe I can also check it with another kernel called RBF kernel, ok. RBF kernel is not getting with that good accuracy. And yeah of course, our testing score also decreased towards 16 percent, which I believe is a chance level. So, yeah, in our case for a fundamental

frequency, we can easily see that our support vector machine with linear kernel giving the best results, ok.

Now, let us try a similar classifier using another feature. So, after fundamental frequency, our next feature was 0 cross rates. Let me code similar stuff for zcr. Again, I will be you know simply reusing my code over here. So, instead of f frequency all, I will be using my zcr underscore all and rest of the part will be same, ok. Now, my train and test variable this setting is over the zcr features.

So, again I will simply reuse my code. I will be using Gaussian Naive Bayes over here. And again I have to show my train and test accuracies. So, I will simply use this code, ok. So, for Gaussian Naive Bayes in case of our 0 cross rates, the train accuracy is somewhat around 85 percent, but the test accuracy is around chance level only.

(Refer Slide Time: 33:27)



```
zcr

[106] 1 X_train, X_test, y_train, y_test = train_test_split(zcr_all, label_all, test_size=0.20, random_state=12)

[107] 1 clf = GaussianNB().fit(X_train, y_train)
2 print('Train Score:', clf.score(X_train, y_train))
3 print('Test Score:', clf.score(X_test, y_test))

Train Score: 0.8541666666666666
Test Score: 0.36666666666666666

[108] 1 clf = LinearSVC(random_state=12).fit(X_train, y_train)
2 print('Train Score:', clf.score(X_train, y_train))
3 print('Test Score:', clf.score(X_test, y_test))

Train Score: 1.0
Test Score: 0.8333333333333333

[109] 1 clf = SVC(kernel='linear').fit(X_train, y_train)
2 print('Train Score:', clf.score(X_train, y_train))
3 print('Test Score:', clf.score(X_test, y_test))

Train Score: 0.8958333333333334
Test Score: 0.3333333333333333

[110] 1 clf = SVC(kernel='rbf').fit(X_train, y_train)
2 print('Train Score:', clf.score(X_train, y_train))
3 print('Test Score:', clf.score(X_test, y_test))

Train Score: 0.7916666666666666
Test Score: 0.25

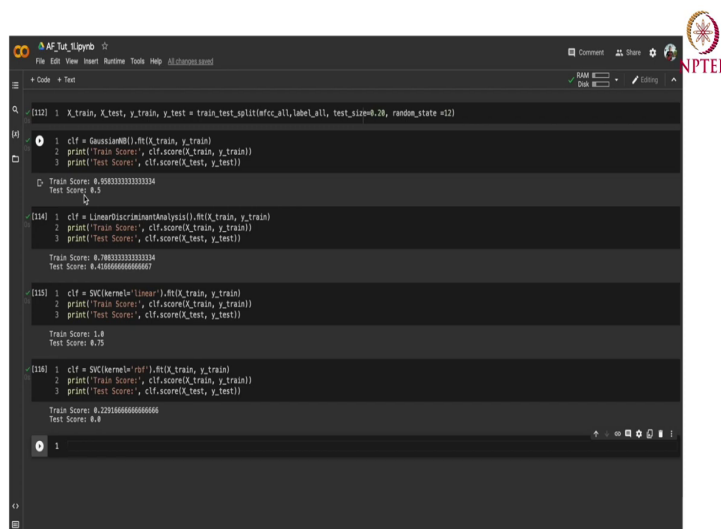
[ ] 1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

So, we will try with another classifier which our linear discriminant analysis. Again, I will simply reuse my code. So, for linear discriminant analysis we are getting a train accuracy of 100 percent and test accuracy is somewhat around 8 percent which is very lower than I guess chance level. And this is a clear example of overfitting in this case. In fact, this is also example of overfitting.

Let me try with support vector machine now. So, you simply change for a function to SVC, SVC and then we use kernel equal to linear, ok. Some problem over here, ok. I forgot to put equal to. In this case results look little bit better than, I mean Gaussian Naive Bayes and linear discriminant analysis, but still there is a huge variance between train score and a test score. So, it is another example of overfitting only. Let me try with the RBF kernel, same case overfitting.

So, now, moving towards our final feature, final manual feature that we have extracted MFCC, let us try to run similar code using MFCC. Again, in re-usability of code.

(Refer Slide Time: 35:24)



```
1121 X_train, X_test, y_train, y_test = train_test_split(mfcc_all, label_all, test_size=0.20, random_state=42)
1122
1123 clf = GaussianNB().fit(X_train, y_train)
1124 print('Train Score:', clf.score(X_train, y_train))
1125 print('Test Score:', clf.score(X_test, y_test))
Train Score: 0.9503333333333334
Test Score: 0.5
1126
1127 clf = LinearDiscriminantAnalysis().fit(X_train, y_train)
1128 print('Train Score:', clf.score(X_train, y_train))
1129 print('Test Score:', clf.score(X_test, y_test))
Train Score: 0.7043333333333334
Test Score: 0.4106666666666667
1130
1131 clf = SVC(kernel='linear').fit(X_train, y_train)
1132 print('Train Score:', clf.score(X_train, y_train))
1133 print('Test Score:', clf.score(X_test, y_test))
Train Score: 1.0
Test Score: 0.75
1134
1135 clf = SVC(kernel='rbf').fit(X_train, y_train)
1136 print('Train Score:', clf.score(X_train, y_train))
1137 print('Test Score:', clf.score(X_test, y_test))
Train Score: 0.22166666666666668
Test Score: 0.4
```

Now, training over Gaussian Naive Bayes classifier for MFCC feature, ok again, ok. These are comparatively better result. We are getting trained accuracy of 95 percent and test accuracy of 50 percent. Now, let us try with linear discriminant analysis, ok. 70, 41, and in case of now in case of our support vector machines, 1 and 75 which was a good result for linear as a linear kernel.

And in case of RBF kernel, ok (Refer Time: 36:34) simply you know working over here. So, yeah, as we can see that support vector machine with linear kernel is giving best results. Now, as we all can see that we have used our basic feature on our basic classifier. After this, I maybe I can do one more exercise where we will be using a raw audio data over a one-dimensional convolution neural network.

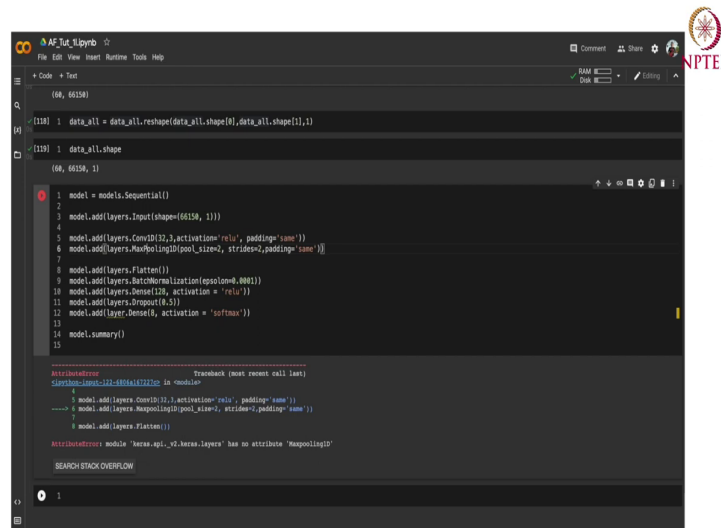
a shape will be a tuple consist of 66150 cross 1. So, I will simply copy it. And after inputting the data of this shape, I will add a convolution layer over it.

So, the code will look something like model dot add. For activation function we will be using relu and padding will be same. After a convolution layer maybe I will try another layer called max pooling or average pooling. Let us say I will use max pool, ok. Maybe I will just use single convolution layer and try to see how my result changes with this network.

Maybe I can put a batch normalization layer, then a dense layer (Refer Time: 39:53) layer dot add see number of neuron equal to 128, with activation equal to relu, ok. Now, maybe we can also add a dropout layer you know just to avoid any overfitting case. And as a final layer, we will simply add a dense layer with number of neuron equal to 8 which is equivalent over number of classes and activation will be Softmax, just for the classification purpose, yeah.

Now, maybe I can just present the summary of this model, ok. There is some error syntax error over here, ok. I forgot to put a equal to over here, ok. Another error activation I again forgot put the equal to over here, no attribute max pooling 1D. Let me check, ok.

(Refer Slide Time: 41:14)



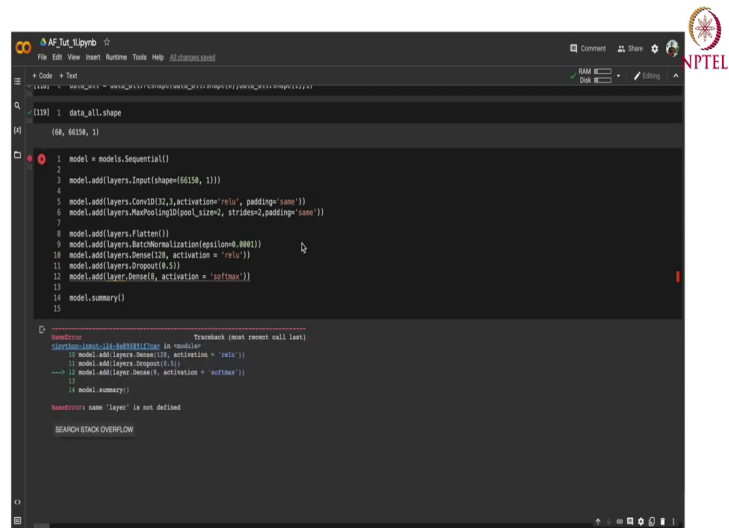
```
AF Tut_11.pyrb
File Edit View Insert Runtime Tools Help
+ Code + Text
(04, 66158)
~/1181 1 data_all = data_all.reshape(data_all.shape[0], data_all.shape[1], 1)
(04, 66158) 1 data_all.shape
(04, 66158, 1)
1 model = models.Sequential()
2
3 model.add(layers.Input(shape=(66158, 1)))
4
5 model.add(layers.Conv2D(32, activation='relu', padding='same'))
6 model.add(layers.MaxPooling2D(pool_size=2, strides=2, padding='same'))
7
8 model.add(layers.Flatten())
9 model.add(layers.BatchNormalization(epsilon=0.001))
10 model.add(layers.Dense128, activation = 'relu')
11 model.add(layers.Dropout(0.5))
12 model.add(layers.Dense64, activation = 'softmax')
13
14 model.summary()
15

AttributeError: Traceback (most recent call last)
~/keras/layers-121-48946132270c in <module>
4
5 model.add(layers.Conv2D(32, activation='relu', padding='same'))
----> 6 model.add(layers.MaxPooling2D(pool_size=2, strides=2, padding='same'))
7
8 model.add(layers.Flatten())

AttributeError: module 'keras.api._v2.keras.layers' has no attribute 'MaxPooling2D'
SEARCH STACK OVERFLOW
```

Another error attribute max pooling, ok. The P in max pooling will be in capital, ok. One more error, ok epsilon spelling is wrong.

(Refer Slide Time: 41:34)



The screenshot shows a Jupyter Notebook interface with a code cell containing the following Python code:

```
1 data_all.shape
2 (68, 68158, 1)
3
4 model = models.Sequential()
5 model.add(layers.Input(shape=(68158, 1)))
6
7 model.add(layers.Conv2D(32, activation='relu', padding='same'))
8 model.add(layers.MaxPooling2D(pool_size=2, strides=2, padding='same'))
9
10 model.add(layers.Flatten())
11 model.add(layers.BatchNormalization(epsilon=0.001))
12 model.add(layers.Dense(128, activation='relu'))
13 model.add(layers.Dropout(0.5))
14 model.add(layers.Dense(9, activation='softmax'))
15
16 model.summary()
```

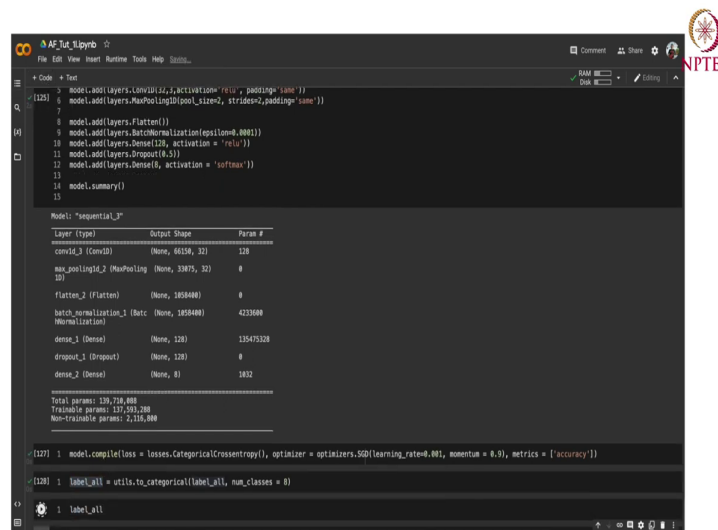
Below the code, a red error message is displayed:

```
RuntimeError: Traceback (most recent call last):
  File <ipython-input-11-9d88881411>: in <module>
    10 model.add(layers.Dense(128, activation='relu'))
    11 model.add(layers.Dropout(0.5))
--> 12 model.add(layers.Dense(9, activation='softmax'))
    13
    14 model.summary()
RuntimeError: name 'layer' is not defined
```

The error message indicates that the name 'layer' is not defined, which is a common mistake when using the `softmax` activation function in Keras. The correct activation function is `softmax`, not `softmax` with a trailing 's'.

One more error, I forgot to put s over here.

(Refer Slide Time: 41:45)



```
model.add(layers.Conv2D(32, activation='relu', padding='same'))
model.add(layers.MaxPooling2D(pool_size=2, strides=2, padding='same'))
model.add(layers.Flatten())
model.add(layers.BatchNormalization(epsilon=0.001))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(8, activation='softmax'))
model.summary()

Model: "sequential_3"
Layer (type)                 Output Shape              Param #
-----
conv1d_1 (Conv2D)            (None, 64, 64, 32)        128
max_pooling1d_1 (MaxPooling (None, 32, 32, 32)        0
flatten_1 (Flatten)          (None, 1024)              0
batch_normalization_1 (Batc (None, 1024)              423360
dense_1 (Dense)              (None, 128)              131072
dropout_1 (Dropout)          (None, 128)              0
dense_2 (Dense)              (None, 8)                 882

Total params: 139,728,000
Trainable params: 137,593,280
Non-trainable params: 2,134,720

model.compile(loss = losses.CategoricalCrossentropy(), optimizer = optimizers.SGD(learning_rate=0.001, momentum = 0.9), metrics = ['accuracy'])
label_all = utils.to_categorical(label_all, num_classes = 8)
```

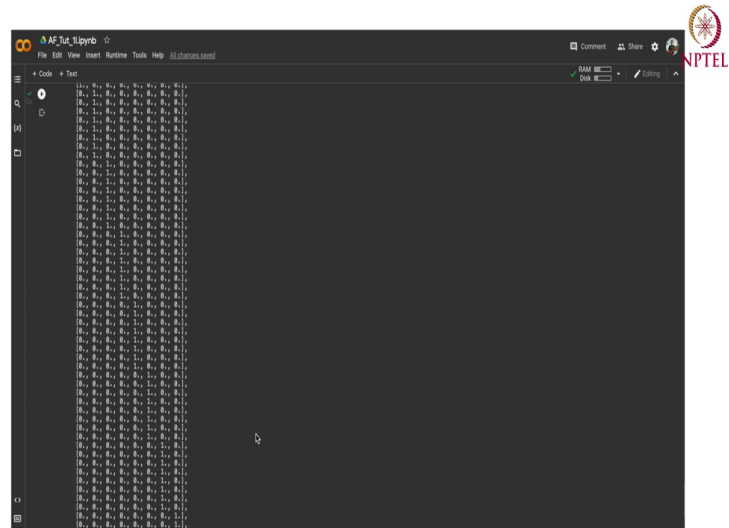
Yeah, now it is working. And this is the summary of our model and these are total number of parameter that the model will be tuning. Out of these parameter, this much will be trainable parameter and others will be non-trainable parameters. Now, we have defined our the structure of our network. Now, we can simply compile this model using model dot compile.

Now, in compilation of model we have to define a loss function, but exact loss function we will be using and a optimizer, but sort of optimizer we will be using to you know optimize that loss function, ok. There is a error over here, sequential model has no attribute compile, ok. I have written the wrong spelling. So, after compilation of my model, I just need to fit my model over a training data.

But before dividing our data into train and test split, I just need to convert my labels into categorical classes in terms of one hot encoded vector, since we are using categorical curve

course entropy loss. So, for that I will be using in build function in tensor for chaos, ok. This has changed my labels into one hot code encoded vectors. Let me show you, yeah.

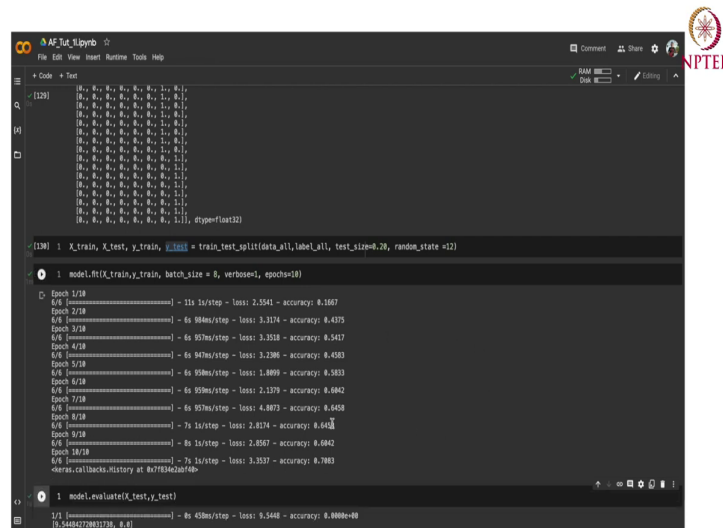
(Refer Slide Time: 43:23)



The image shows a screenshot of a code editor window titled "AE_Mat_11.pyrb". The editor displays a large list of one-hot encoded vectors, each represented as a list of 10 binary values (0s and 1s). The vectors are arranged in a grid-like pattern, with each row representing a different input or output state. The editor interface includes a menu bar (File, Edit, View, Insert, Runtime, Tools, Help), a search bar, and a status bar at the bottom. A small logo with the text "NPTEL" is visible in the top right corner of the editor window.

So, instead of a single value over here, we have this vectors.

(Refer Slide Time: 43:31)



```
AF Tut_11.ipynb
File Edit View Insert Runtime Tools Help
Code + Test
[129] 10, 0., 0., 0., 0., 0., 1., 0.,
10, 0., 0., 0., 0., 0., 1., 0.,
10, 0., 0., 0., 0., 0., 1., 0.,
10, 0., 0., 0., 0., 0., 1., 0.,
10, 0., 0., 0., 0., 0., 1., 0.,
10, 0., 0., 0., 0., 0., 1., 0.,
10, 0., 0., 0., 0., 0., 1., 0.,
10, 0., 0., 0., 0., 0., 1., 0.,
10, 0., 0., 0., 0., 0., 1., 0.,
10, 0., 0., 0., 0., 0., 1., 0.,
10, 0., 0., 0., 0., 0., 1., 0.,
10, 0., 0., 0., 0., 0., 1., 0.,
10, 0., 0., 0., 0., 0., 1., 0.,
10, 0., 0., 0., 0., 0., 1., 0.,
10, 0., 0., 0., 0., 0., 1., 0.,
dtype=float32)

[130] 1 X_train, X_test, y_train, y_test = train_test_split(data_all, label_all, test_size=0.20, random_state=12)

1 model.fit(X_train, y_train, batch_size=8, verbose=1, epochs=10)

Epoch 1/10 [-----] - 11s 1s/step - loss: 2.5541 - accuracy: 0.1667
Epoch 2/10 [-----] - 6s 980ms/step - loss: 3.3174 - accuracy: 0.4375
Epoch 3/10 [-----] - 6s 937ms/step - loss: 3.3518 - accuracy: 0.5417
Epoch 4/10 [-----] - 6s 947ms/step - loss: 3.2386 - accuracy: 0.4583
Epoch 5/10 [-----] - 6s 958ms/step - loss: 1.8099 - accuracy: 0.5833
Epoch 6/10 [-----] - 6s 959ms/step - loss: 2.1379 - accuracy: 0.6842
Epoch 7/10 [-----] - 6s 957ms/step - loss: 4.8873 - accuracy: 0.6458
Epoch 8/10 [-----] - 7s 1s/step - loss: 2.8174 - accuracy: 0.6417
Epoch 9/10 [-----] - 8s 1s/step - loss: 2.8567 - accuracy: 0.6942
Epoch 10/10 [-----] - 7s 1s/step - loss: 3.3537 - accuracy: 0.7083
<-----calbacks.history at 0x77834c220f4b>

1 model.evaluate(X_test, y_test)

1/1 [-----] - 0s 430ms/step - loss: 9.5468 - accuracy: 0.0000e+00
[0.544842728031738, 0.8]
```

Now, I can use my train and test splitting code to you know divide this data into train and test split. I will simply copy my previous code and increased it over here. And this time my training data will be our raw audio data, ok. So, after dividing to train and test split, we have to simply fit our model, but we have already defined and compiled through model dot fit. It might take some time to you know as we are, right now we are just using our CPU you know Google Colab. So, it might take some time to learn, ok.

We can see that our accuracy is improving over here, ok. So, my model has now completed all its epochs. So, let us try to evaluate what test accuracy is over here. We are getting training accuracy of somewhere around 70 percent and test accuracy is 0 percent, ok. So, this, my this network architecture is not learning anything as of now.

Maybe I can you know start with some sort of a hyper parameter tuning, start adding a couple of layers in it or maybe using different sort of activation functions or decreasing or increasing the number of neuron in dense layers. All that sort of hyper parameter tuning we I can do to make this network a better classifier.

So, guys, this was all about this tutorial. Hope I was able to give you a basic idea about programming on these sort of networks. And if you have any sort of doubt, feel free to put a question in discussion form.

Thank you.