

MACHINE LEARNING

(Speech Emotion Recognition)

Summer Internship Report Submitted in partial fulfillment

of the requirement for undergraduate degree of

Bachelor of Technology

In

Computer Science Engineering

By

LIKHITA YANAMADDI

221710315024

Under the Guidance of

Mr. M. Venkateswarlu

Assistant Professor



Department Of Electronics and Communication Engineering

GITAM School of Technology

GITAM (Deemed to be University)

Hyderabad-502329

July 2020

DECLARATION

I submit this industrial training work entitled “**Speech Emotion Recognition**” to GITAM (Deemed To Be University), Hyderabad in partial fulfillment of the requirements for the award of the degree of “**Bachelor of Technology**” in “**Computer Science Engineering**”. I declare that it was carried out independently by me under the guidance of **Mr.** Asst. Professor, GITAM (Deemed To Be University), Hyderabad, India.

The results embodied in this report have not been submitted to any other University or Institute for the award of any degree or diploma.

Place: HYDERABAD

LIKHITA YANAMADDI

Date:

221710315024



GITAM (DEEMED TO
BE UNIVERSITY)

Hyderabad-502329,

India Dated:

CERTIFICATE

This is to certify that the Industrial Training Report entitled “**SPEECH EMOTION RECOGNITION**” is being submitted by LIKHITA YANAMADDI (221710315024) in partial fulfillment of the requirement for the award of **Bachelor of Technology in Computer Science Engineering** at GITAM (Deemed To Be University), Hyderabad during the academic year 2018-19

It is faithful record work carried out by her at the **Computer Science Engineering Department**, GITAM University Hyderabad Campus under my guidance and supervision.

1.1.1 Mr.

Assistant Professor
Department of ECE

Dr.

Professor and HOD
Department of ECE

ACKNOWLEDGEMENT

Apart from my effort, the success of this internship largely depends on the encouragement and guidance of many others. I take this opportunity to express my gratitude to the people who have helped me in the successful completion of this internship.

I would like to thank respected **Dr. N. Siva Prasad**, Pro Vice Chancellor, GITAM Hyderabad and **Dr. CH. Sanjay**, Principal, GITAM Hyderabad

I would like to thank respected **Dr.** Head of the Department of Electronics and Communication Engineering for giving me such a wonderful opportunity to expand my knowledge for my own branch and giving me guidelines to present a internship report. It helped me a lot to realize of what we study for.

I would like to thank the respected faculties **Mr.** who helped me to make this internship a successful accomplishment.

I would also like to thank my friends who helped me to make my work more organized and well-stacked till the end.

Likhita Yanamaddi

221710315024

Nagma

(18071A0546)

ABSTRACT

Emotion plays a significant role in daily interpersonal human interactions and it is an abstract concept for a robot to understand. This is essential to make rational as well as intelligent decisions. It helps to match and understand the feelings of others by conveying our feelings and giving feedback to others. Research has revealed the powerful role that emotion play in shaping human social interaction as well as human robot interaction. Emotional displays convey considerable information about the mental state of an individual. This Project can be used for application in call center systems, interactive teaching, lie detection, etc.

Speech emotion recognition has been a vital topic of research in human-machine interface applications for many years. Our project is a comparison and analysis of up-to-date Speech Emotion Recognition approaches regarding the use of different classification algorithms. The focus is mainly on classifiers like Multilayer Perceptron (MLP), Support Vector Machine, Decision Tree, Random Forest, and Convolutional Neural Network (CNN). First, the RAVDASS dataset will be discussed in detail. Second, the features that were extracted and selected will be addressed. Then, the focus is shifted to these classifier algorithms that categorize the input data into 4 classes of emotions: happy, angry, sad, and neutral. Each algorithm is implemented and its performance (accuracy) is compared with the others. Finally, conclusions about the performance and limitations of each classifier used for Speech Emotion Recognition System are presented.

Table of Contents:

LIST OF FIGURES

CHAPTER 1:MACHINE LEARNING1

1.1 INTRODUCTION	1
1.2 IMPORTANCE OF MACHINE LEARNING	1
1.3 USES OF MACHINE LEARNING	2
1.4 TYPES OF LEARNING ALGORITHMS	3
1.4.1 Supervised Learning	3
1.4.2 Unsupervised Learning	3
1.4.3 Semi Supervised Learning	4
1.5 RELATION BETWEEN DATA MINING,MACHINE LEARNING AND DEEP LEARNING	5

CHAPTER 2: INFORMATION ABOUT DEEP LEARNING

2.1 Importance of deep learning	5
2.2 Uses of deep learning	5
2.3 Relation between data mining, machine learning and deep learning	6

CHAPTER 3:PYTHON

3.1 INTRODUCTOIN TO PYTHON	6
3.2 HISTORY OF PYTHON	6
3.3 FEATURES OF PYTHON	6
3.4 HOW TO SETUP PYTHON	7
3.4.1 Installation(using python IDLE)	7
3.4.2 Installation(using Anaconda)	8
3.5 PYTHON VARIABLE TYPES	9
3.5.1 Python Numbers	10
3.5.2 Python String	10

3.5.3	Python Lists	11
3.5.4	Python Tuples	11
3.5.5	Python Dictionary	12
3.6	PYTHON FUNCTION	12
3.6.1	Defining a Function	12
3.6.2	Calling a Function	13
3.7	PYTHON USING OOP's CONCEPTS	14
3.7.1	Class	14
3.7.2	__init__ method in class	15

CHAPTER 4: CASE STUDY

4.1	PROBLEM STATEMENT	16
4.2	DATA SET	16
4.3	OBJECTIVE OF CASE STUDY	17
4.4	TECHNOLOGY	17
4.5	IMPLEMENTATION	23

CHAPTER 5: MODEL BUILDING

5.1	INSTALLING THE PACKAGES	25
5.2	MOUNTING THE DRIVE	25
5.3	NECESSARY IMPORTS	26
5.3.1	Metplotlib	26
5.3.2	Sklern	26
5.3.3	Numpy	26
5.3.4	Librosa	27
5.3.5	Soundfile	27
5.3.6	Keras	27
5.4	DEFINE A DICTIONARY	28
5.5	DEFINE A FUNCTION	29
5.6	LOAD THE DATA	30
5.7	TRAINING THE DATASET	31

CHAPTER 6: IMPLEMENTING ALGORITHMS

6.1 CNN ALGORITHM

6.1.1 Importing packages	34
6.1.2 training and testing	34
6.1.3 Importing required packages	35
6.1.4 model summary	36
6.1.5 compiling the model	37
6.1.6 model history	37
6.1.7 model history accuracy	37
6.1.8 train and test score	38
6.1.9 CNN accuracy	38

6.2 DECISION TREE ALGORITHM

6.2.1 importing packages	39
6.2.2 classification report	40

6.3 SVM ALGORITHM

6.3.1 importing packages	41
6.3.2 SVM training dataset accuracy	42
6.3.3 SVM testing dataset accuracy	43

6.4 RANDOM FOREST ALGORITHM

6.4.1 importing packages	44
6.4.2 training data accuracy	44
6.4.3 testing data accuracy	45

6.5 MLP CLASSIFIER

6.5.1 required parameters	46
6.5.2 initializing MLP classifier	46

6.5.3 predicting accuracy of trained dataset	47
6.5.4 classification report and confusion matrix of trained set	47
6.5.5 predicting accuracy of tested dataset	48
6.5.6 classification report and confusion matrix of tested set	48

CHAPTER 7: COMPARISION

7.1 comparision of different models	49
-------------------------------------	----

CHAPTER 8: CONCLUSION

8 conclusion and future scope	
8.1 conclusion	52
8.2 future scope	52

TABLE OF FIGURES

Figure 1.2.1: The Process Flow	2
Figure 1.4.2.1: Unsupervised Learning	4
Figure 1.4.3.1: Semi Supervised Learning	5
Figure 3.4.1.1: Python download	12
Figure 3.4.2.1: Anaconda downloads	13
Figure 3.4.2.2: Jupyter notebook	14
Figure 3.7.1.1: Defining a Class	16
Figure 4.5.1: Installing packages	18
Figure 5.1.1: installing soundfile	20
Figure 5.2.1: mounting the drive	20
Figure 5.3.1: necessary import	21
Figure 5.4.1: emotions available	23
Figure 5.5.1: extract features	24
Figure 5.6.1: load the data	25
Figure 5.6.2: dataset	26
Figure 5.6.3: sub-dataset	26
Figure 5.7.1: training the dataset	27
Figure 5.7.2: training samples	27
Figure 6.1.1.1: importing packages	29
Figure 6.1.2.1: training and testing	30
Figure 6.1.2.2: training and testing	30
Figure 6.1.3.1: importing packages	31
Figure 6.1.3.2: importing packages(continue)	31
Figure 6.1.4.1: model summary	32
Figure 6.1.5.1: compiling model	32

Figure 6.1.6.1: model history	33
Figure 6.1.7.1: model history accuracy	33
Figure 6.1.8.1: train and test score	34
Figure 6.1.9.1: CNN accuracy	34
Figure 6.2.1.1: importing packages	36
Figure 6.2.2.1: classification report and confusion matrix	36
Figure 6.3.1.1: importing packages	38
Figure 6.3.2.1: trained data accuracy	39
Figure 6.4.1.1: tested data accuracy	40
Figure 6.4.2.1: importing packages	42
Figure 6.4.3.1: trained set accuracy	42
Figure 6.4.3.2: tested set accuracy	43
Figure 6.5.1.1: required parameters	45
Figure 6.5.2.1: initializing	45
Figure 6.5.3.1: training the model and accuracy	46
Figure 6.5.4.1: classification report of trained dataset	46
Figure 6.5.5.1: classification report of tested dataset	47
Figure 7.1.1: comparing the accuracy of algorithms	48
Figure 7.1.2: comparing all algorithm trained set	49
Figure 7.1.3: bar graph of accuracy of all algorithms for tested dataset	49
Figure 7.2.1: training and testing the dataset	50
Figure 7.2.2: support vector machine	51
Figure 7.2.3 :svm using tested dataset	52
Figure 7.2.4: random forest	53
Figure 7.2.5: random forest using testing dataset	54
Figure 7.2.6: mlp classifier	55
Figure 7.2.7: classification report	56
Figure 7.2.8: accuracy	56
Figure 7.2.9: comparison of accuracy of all algorithms using unknown dataset	57

CHAPTER 1

MACHINE LEARNING

1.1 INTRODUCTION:

Machine Learning(ML) is the scientific study of algorithms and statistical models that computer systems use in order to perform a specific task effectively without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of Artificial Intelligence(AI).

1.2 IMPORTANCE OF MACHINE LEARNING:

Consider some of the instances where machine learning is applied: the self-driving Google car, cyber fraud detection, online recommendation engines—like friend suggestions on Facebook, Netflix showcasing the movies and shows you might like, and “more items to consider” and “get yourself a little something” on Amazon—are all examples of applied machine learning. All these examples echo the vital role machine learning has begun to take in today’s data-rich world.

Machines can aid in filtering useful pieces of information that help in major advancements, and we are already seeing how this technology is being implemented in a wide variety of industries.

With the constant evolution of the field, there has been a subsequent rise in the uses, demands, and importance of machine learning. Big data has become quite a buzzword in the last few years; that’s in part due to increased sophistication of machine learning, which helps analyze those big chunks of big data. Machine learning has also changed the way data extraction, and interpretation is done by involving automatic sets of generic methods that have replaced traditional statistical techniques.

The process flow depicted here represents how machine learning works



Figure 1.2.1 : The Process Flow

1.3 USES OF MACHINE LEARNING:

Earlier in this article, we mentioned some applications of machine learning. To understand the concept of machine learning better, let's consider some more examples: web search results, real-time ads on web pages and mobile devices, email spam filtering, network intrusion detection, and pattern and image recognition. All these are by-products of applying machine learning to analyze huge volumes of data

Traditionally, data analysis was always being characterized by trial and error, an approach that becomes impossible when data sets are large and heterogeneous. Machine learning comes as the solution to all this chaos by proposing clever alternatives to analyzing huge volumes of data.

By developing fast and efficient algorithms and data-driven models for real-time processing of data, machine learning can produce accurate results and analysis.

1.4 TYPES OF LEARNING ALGORITHMS:

The types of machine learning algorithms differ in their approach, the type of data they input and output, and the type of task or problem that they are intended to solve.

1.4.1 Supervised Learning :

When an algorithm learns from example data and associated target responses that can consist of numeric values or string labels, such as classes or tags, in order to later predict the correct response when posed with new examples comes under the category of supervised learning.

Supervised machine learning algorithms uncover insights, patterns, and relationships from a labelled training dataset – that is, a dataset that already contains a known value for the target variable for each record. Because you provide the machine learning algorithm with the correct answers for a problem during training, it is able to “learn” how the rest of the features relate to the target, enabling you to uncover insights and make predictions about future outcomes based on historical data.

Examples of Supervised Machine Learning Techniques are Regression, in which the algorithm returns a numerical target for each example, such as how much revenue will be generated from a new marketing campaign.

Classification, in which the algorithm attempts to label each example by choosing between two or more different classes. Choosing between two classes is called binary classification, such as determining whether or not someone will default on a loan. Choosing between more than two classes is referred to as multiclass classification.

1.4.2 Unsupervised Learning:

When an algorithm learns from plain examples without any associated response, leaving to the algorithm to determine the data patterns on its own. This type of algorithm tends to restructure the data into something else, such as new features that may represent a class or a new series of uncorrelated values. They are quite useful in providing humans with insights into the meaning of data and new useful inputs to supervised machine learning algorithms.

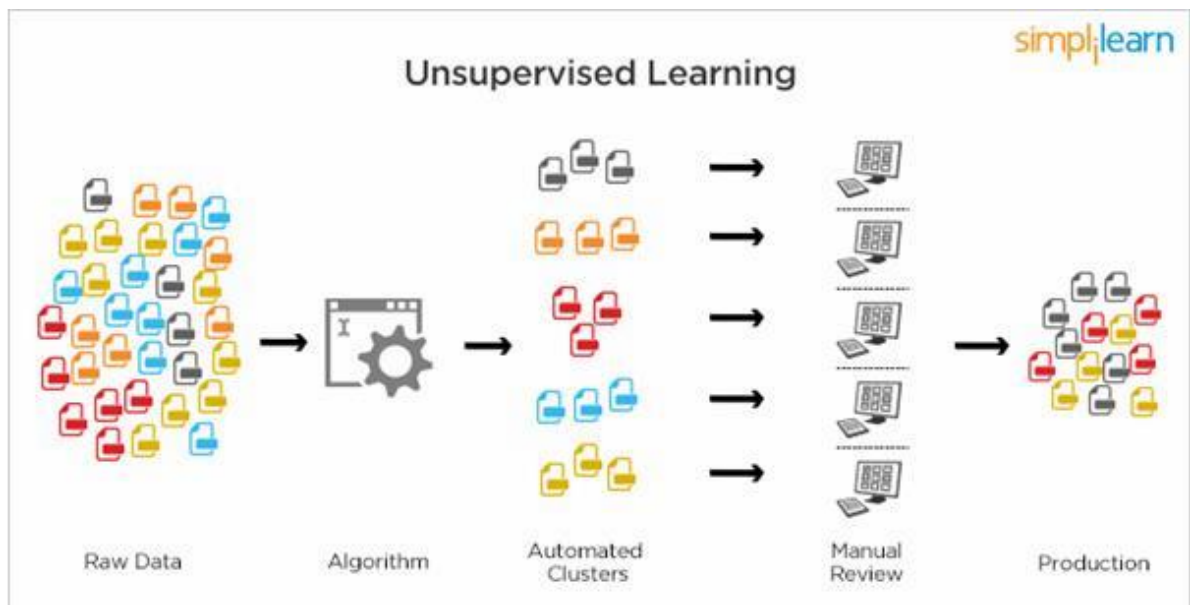


Figure 1.4.2.1 : Unsupervised Learning

Popular techniques where unsupervised learning is used also include self-organizing maps, nearest neighbor mapping, singular value decomposition, and k-means clustering. Basically, online recommendations, identification of data outliers, and segment text topics are all examples of unsupervised learning.

1.4.3 Semi Supervised Learning:

As the name suggests, semi-supervised learning is a bit of both supervised and unsupervised learning and uses both labeled and unlabeled data for training. In a typical scenario, the algorithm would use a small amount of labeled data with a large amount of unlabeled data.

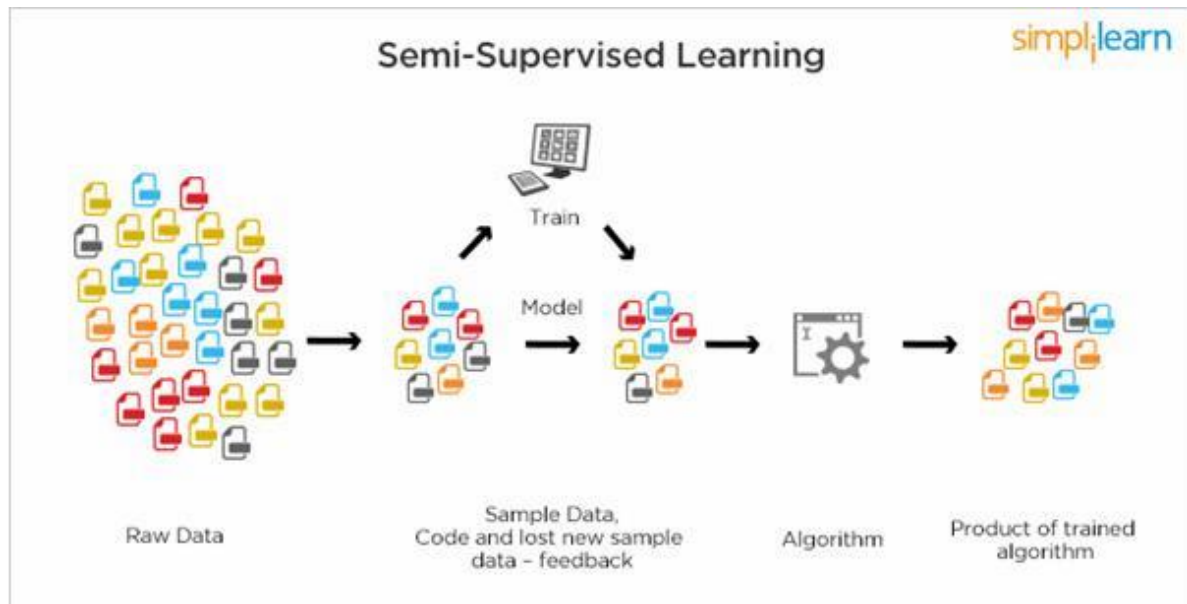


Figure 1.4.3.1 : Semi Supervised Learning

1.5 RELATION BETWEEN DATA MINING, MACHINE LEARNING AND DEEP LEARNING:

Machine learning and data mining use the same algorithms and techniques as data mining, except the kinds of predictions vary. While data mining discovers previously unknown patterns and knowledge, machine learning reproduces known patterns and knowledge—and further automatically applies that information to data, decision-making, and actions.

Deep learning, on the other hand, uses advanced computing power and special

types of neural networks and applies them to large amounts of data to learn, understand, and identify complicated patterns. Automatic language translation and medical diagnoses are examples of deep learning.

CHAPTER 2

INFORMATION ABOUT DEEPLEARNING

2.1 IMPORTANCE OF DEEP LEARNING:

Deep learning recently returned to the headlines when google's AlphaGo program crushed Lee Sedol, one of the highest-ranking Go players in the world. Google has invested heavily in deep learning and AlphaGo is just their latest deep learning project to make the news. Google's search engine, voice recognition system and self-driving cars all rely heavily on deep learning. They've used deep learning networks to build a program that picks out an alternative still from a YouTube video to use as a thumbnail. Late last year Google announced smart reply, a deep learning network that writes short email responses for you. Deep learning is clearly powerful, but it also may seem somewhat mysterious.

2.2 USES OF MACHINE LEARNING:

The value of machine learning technology has been recognized by companies across several industries that deal with huge volumes of data. By leveraging insights obtained from this data, companies are able work in an efficient manner to control costs as well as get an edge over their competitors. This is how some sectors / domains are implementing machine learning –

- **Financial Services:**

Companies in the financial sector are able to identify key insights in financial data as well as prevent any occurrences of financial fraud, with the help of machine learning technology. The technology is also used to identify opportunities for investments and trade. Usage of cyber surveillance helps in identifying those

individuals or institutions which are prone to financial risk, and take necessary actions in time to prevent fraud.

- **Marketing sales:**

Companies are using machine learning technology to analyze the purchase history of their customers and make personalized product recommendations for their next purchase. This ability to capture, analyze, and use customer data to provide a personalized shopping experience is the future of sales and marketing.

- **Government:**

Government agencies like utilities and public safety have a specific need FOR ML, as they have multiple data sources, which can be mined for identifying useful patterns and insights. For example sensor data can be analyzed to identify ways to minimize costs and increase efficiency. Furthermore, ML can also be used to minimize identity thefts and detect fraud.

- **Health care:**

With the advent of wearable sensors and devices that use data to access health of a patient in real time, ML is becoming a fast-growing trend in healthcare. Sensors in wearable provide real-time patient information, such as overall health condition, heartbeat, blood pressure and other vital parameters. Doctors and medical experts can use this information to analyze the health condition of an individual, draw a pattern from the patient history, and predict the occurrence of any ailments in the future. The technology also empowers medical experts to analyze data to identify trends that facilitate better diagnoses and treatment.

- **Transportation:**

Based on the travel history and pattern of traveling across various routes, machine learning can help transportation companies predict potential problems that could arise on certain routes, and accordingly advise their customers to opt for a different route. Transportation firms and delivery organizations are increasingly using machine learning technology to carry out data analysis and data modeling to make

informed decisions and help their customers make smart decisions when they travel.

- **Oil and gas:**

This is perhaps the industry that needs the application of machine learning the most. Right from analyzing underground minerals and finding new energy sources to streaming oil distribution, ML applications for this industry are vast and are still expanding.

2.3 RELATION BETWEEN DATA MINING MACHINE LEARNING AND DEEP LEARNING:

Machine learning and data mining use the same algorithms and techniques as data mining, except the kinds of predictions vary. While data mining discovered previously unknown patterns and knowledge, machine learning reproduces known patterns and knowledge—and further automatically applies that information to data, decision-making, and actions. Deep learning, on the other hand, uses advanced computing power and special types of neural networks and applies them to large amounts of data to learn, understand, and identify complicated patterns. Automatic language translation and medical diagnoses are examples of deep learning.

CHAPTER 3

PYTHON

Basic programming language used for machine learning is : PYTHON

3.1 INTRODUCTION TO PYHTON:

- Python is a high-level, interpreted, interactive and object-oriented scripting language.
- Python is a general purpose programming language that is often applied in scripting roles
- Python is Interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is like PERL and PHP.
- Python is Interactive: You can sit at a Python prompt and interact with the interpreter directly to write your programs.
- Python is Object-Oriented: Python supports the Object-Oriented style or technique of programming that encapsulates code within objects.

3.2 HISTORY OF PYTHON:

- Python was developed by GUIDO VAN ROSSUM in early 1990's
- Its latest version is 3.7 , it is generally called as python3

3.3 FEATURES OF PYTHON:

- Easy-to-learn: Python has few keywords, simple structure, and a clearly defined syntax,
- This allows the student to pick up the language quickly.
- Easy-to-read: Python code is more clearly defined and visible to the eyes.
- Easy-to-maintain: Python's source code is fairly easy-to-maintaining.
- A broad standard library: Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- Portable: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- Extendable: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- Databases: Python provides interfaces to all major commercial databases.
- GUI Programming: Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

3.4 HOW TO SETUP PYTHON:

- Python is available on a wide variety of platforms including Linux and Mac OS X. Let's understand how to set up our Python environment.
- The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python.

3.4.1 Installation(using python IDLE):

- Installing python is generally easy, and nowadays many Linux and Mac OS distributions include a recent python.
- [Download python from www.python.org](http://www.python.org)
- When the download is completed, double click the file and follow the instructions to install it.
- When python is installed, a program called IDLE is also installed along with it. It provides a graphical user interface to work with python.

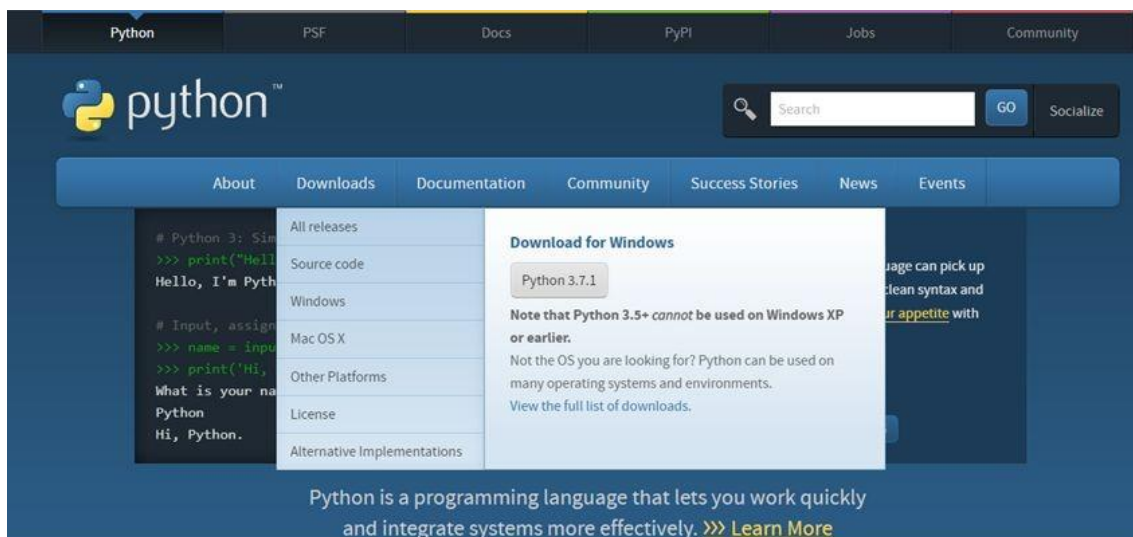


Figure 3.4.1.1 : Python download

3.4.2 Installation(using Anaconda):

- Python programs are also executed using Anaconda.
- Anaconda is a free open source distribution of python for large scale data processing, predictive analytics and scientific computing.
- Conda is a package manager quickly installs and manages packages.

- In WINDOWS:
- In windows
 - Step 1: Open Anaconda.com/downloads in web browser.
 - Step 2: Download python 3.4 version for (32-bitgraphic installer/64 -bit graphic installer)
 - Step 3: select installation type(all users)
 - Step 4: Select path(i.e. add anaconda to path & register anaconda as default python 3.4) next click install and next click finish
 - Step 5: Open jupyter notebook (it opens in default browser)

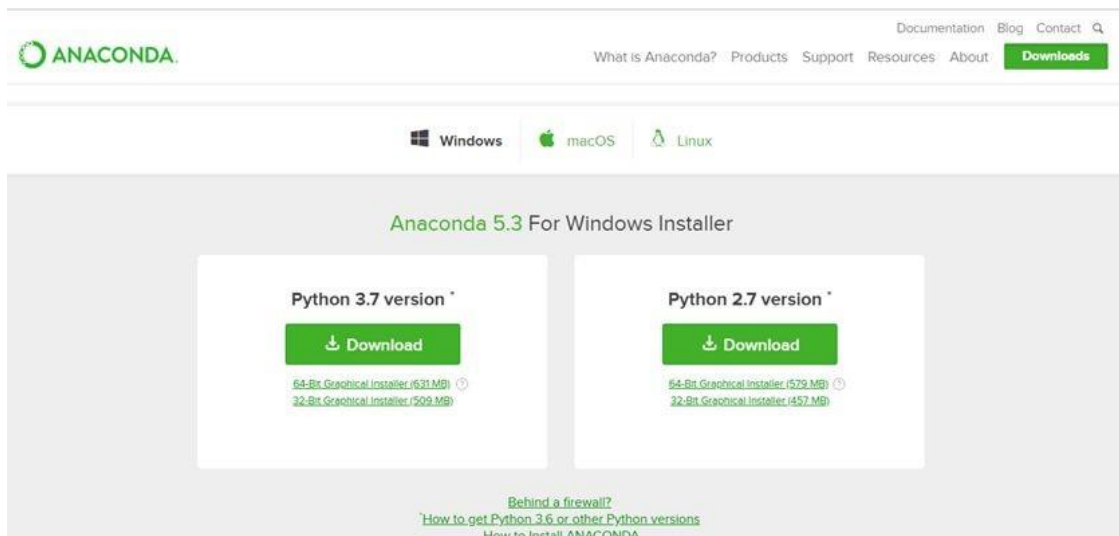


Figure 3.4.2.1 : Anaconda download



Figure 3.4.2.2 : Jupyter notebook

3.5 PYTHON VARIABLE TYPES:

- Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.
- Variables are nothing but reserved memory locations to store values.
- Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.
- Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable.
- Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.
- Python has five standard data types –
 - Numbers
 - Strings
 - Lists

- Tuples
- Dictionary

3.5.1 Python Numbers:

- 3.5.1.1 Number data types store numeric values. Number objects are created when you assign a value to them.
- 3.5.1.2 Python supports four different numerical types – int (signed integers) long (long integers, they can also be represented in octal and hexadecimal) float (floating point real values) complex (complex numbers).

3.5.2 Python Strings:

- Strings in Python are identified as a contiguous set of characters represented in the quotation marks.
- Python allows for either pairs of single or double quotes.
- Subsets of strings can be taken using the slice operator ([] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.
- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

3.5.3 Python Lists:

- Lists are the most versatile of Python's compound data types.
- A list contains items separated by commas and enclosed within square bracket

- To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.
- The values stored in a list can be accessed using the slice operator ([] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1.
- The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

3.5.4 Python Tuples:

- A tuple is another sequence data type that is similar to the list.
- A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.
- The main differences between lists and tuples are: Lists are enclosed in brackets ([]) and their elements and size can be changed, while tuples are enclosed in parentheses (()) and cannot be updated.
- Tuples can be thought of as read-only lists.
- For example – Tuples are fixed size in nature whereas lists are dynamic. In other words, a tuple is immutable whereas a list is mutable. You can't add elements to a tuple. Tuples have no append or extend method. You can't remove elements from a tuple. Tuples have no remove or pop method.

3.5.5 Python Dictionary:

- Python's dictionaries are kind of hash table type. They work like associative array

or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

- Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).
- You can use numbers to "index" into a list, meaning you can use numbers to find out what's in lists. You should know this about lists by now, but make sure you understand that you can only use numbers to get items out of a list.
- What a dict does is let you use anything, not just numbers. Yes, a dict associates one thing to another, no matter what it is.

3.6 PYTHON FUNCTION:

3.6.1 Defining a Function:

You can define functions to provide the required functionality. Here are simple rules to define a function in Python. Function blocks begin with the keyword `def` followed by the function name and parentheses (i.e.()).

Any input parameters or arguments should be placed within these parentheses.

You can also define parameters inside these parentheses

The code block within every function starts with a colon (:) and is indented. The statement `return [expression]` exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as `return None`.

3.6.2 Calling a Function:

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code. Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt.

3.7 PYTHON USING OOP's CONCEPTS:

3.7.1 Class:

- **Class:** A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.
- **Class variable:** A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are.
- **Data member:** A class variable or instance variable that holds data associated with a class and its objects.
- **Instance variable:** A variable that is defined inside a method and belongs only to the current instance of a class.
- **Defining a Class:**
 - We define a class in a very similar way how we define a function.
 - Just like a function, we use parentheses and a colon after the class name(i.e. `():`) when we define a class. Similarly, the body of our class is

- indented like a functions body is.

```
def my_function():  
    # the details of the  
    # function go here
```

```
class MyClass():  
    # the details of the  
    # class go here
```

Figure 3.7.1.1 : Defining a Class

3.7.2 `__init__` method in Class:

- The init method — also called a constructor — is a special method that runs when an instance is created so we can perform any tasks to set up the instance.
- The init method has a special name that starts and ends with two underscores: `init ()`.

CHAPTER 4

CASE STUDY

4.1 PROBLEM STATEMENT:

To extract emotional feature from speech signals by computer and contrasts and analyses the characteristic parameters and changes the emotion

4.2 DATA SET:

Speech can contain certain parameters that are useful in gaining emotional information. The collected speech samples must preserve legitimacy. It is a fundamental part of Speech Emotion Recognition which determines the quality of the model and the credibility of the results.

The Sampling library of speech data must follow certain standards to be of any relevance:

- The dialogue uttered must not contain any emotional bias that will affect the judgment.
- The dialogues should be such that the same sentence can be expressed in various emotions for us to compare the emotional speech parameters.

Taking these constraints into consideration, we used the Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS) Dataset for the Speech Emotion Recognition System. This dataset has 7356 files rated by 247 individuals 10 times on emotional validity, intensity, and genuineness. Only audio files are considered for this comparative study of classifiers. It contains the speech samples collected from 24 professional actors who vocalized the emotional expressions at two levels of emotional intensity with an additional neutral expression.

4.3 OBJECTIVE OF THE CASE STUDY:

The objective of the project is to build a model to recognize emotion from speech using the librosa and sklearn libraries and the RAVDESS dataset.

4.4 TECHNOLOGY:

- Decision Tree Algorithm
- Random Forest Algorithm
- Support Vector Machine(SVM)
- Multi-Layer Perceptron(MLP)
- Convolutional Neural Networks (CNN)

4.5 IMPLEMENTATION:

Flow Chart of Implementation

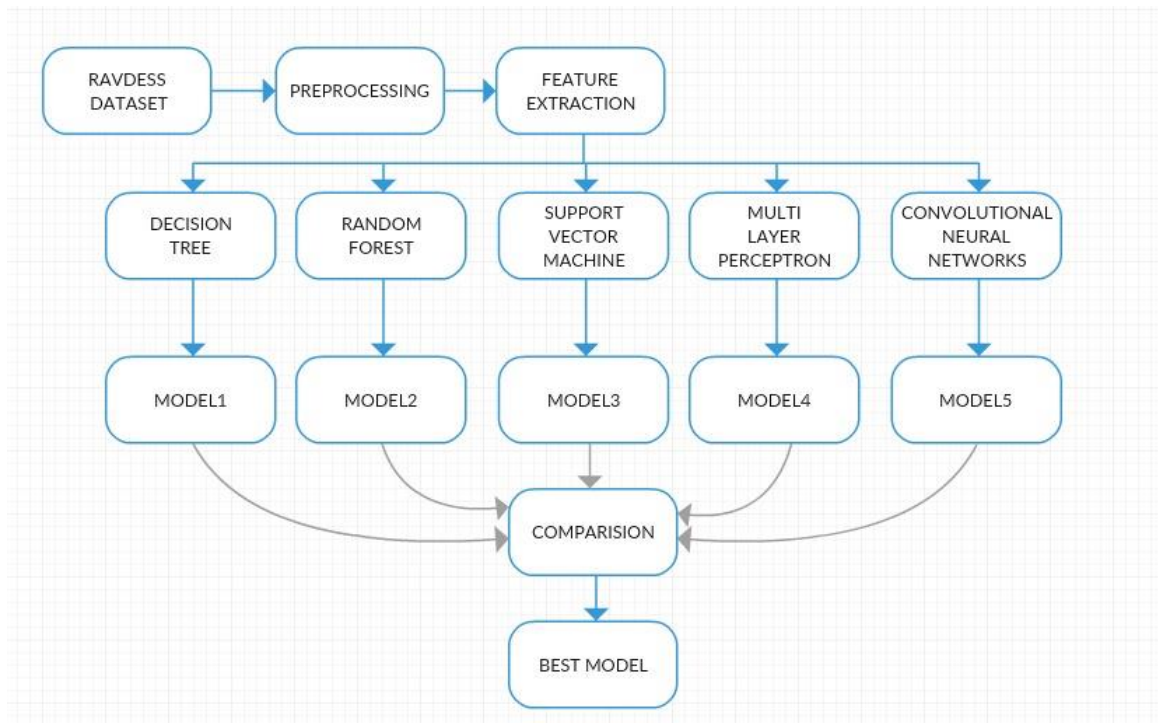


Figure 4.5.1 Flow Chart of Implementation

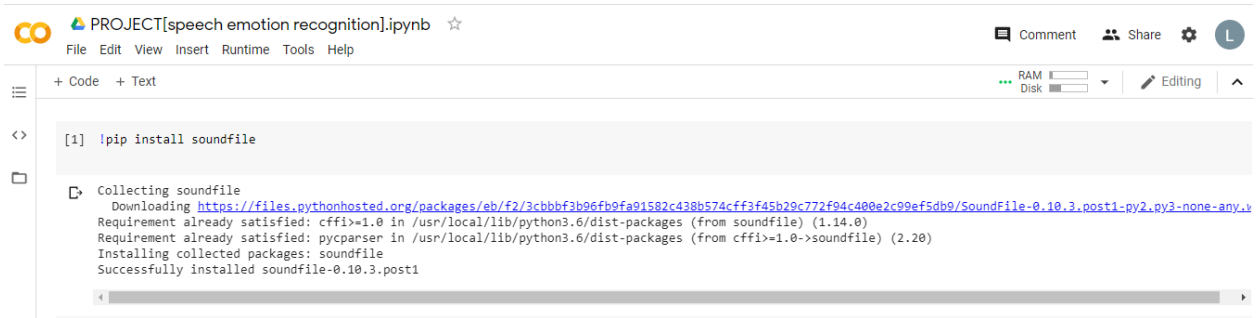
We can define the machine learning work flow in 5 stages.

- Gathering data
- Data pre-processing
- Researching the model that will be best for the type of data
- Training and testing the model
- Evaluation

CHAPTER 4

MODEL BUILDING

5.1 INSTALLING THE PACKAGES:



The screenshot shows a Jupyter Notebook titled "PROJECT[speech emotion recognition].ipynb". The code cell contains the command `!pip install soundfile`. The output shows the process of collecting the package, downloading it from a URL, checking requirements, and successfully installing `soundfile-0.10.3.post1`.

```
[1] !pip install soundfile

Collecting soundfile
  Downloading https://files.pythonhosted.org/packages/eb/f2/3cbbbf3b96fb9fa91582c438b574cff3f45b29c772f94c400e2c99ef5db9/SoundFile-0.10.3.post1-py2.py3-none-any.whl (1.14MB)
Requirement already satisfied: cffi>=1.0 in /usr/local/lib/python3.6/dist-packages (from soundfile) (1.14.0)
Requirement already satisfied: pycparser in /usr/local/lib/python3.6/dist-packages (from cffi>=1.0->soundfile) (2.20)
Installing collected packages: soundfile
Successfully installed soundfile-0.10.3.post1
```

Figure 5.1.1 installing sound file

5.2 MOUNTING THE DRIVE:



The screenshot shows a Jupyter Notebook with the code `from google.colab import drive` and `drive.mount('/content/drive')`. The output shows a URL for authorization, a prompt for an authorization code, and the final message "Mounted at /content/drive".

```
from google.colab import drive
drive.mount('/content/drive')

Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6gk8ndgf4ndg3pfee6491hc0brcdi.apps.googleusercontent.com&redirect_uri=https://colab.research.google.com/&scope=https://www.googleapis.com/auth/drive
Enter your authorization code:
.....
Mounted at /content/drive
```

Figure 5.2.1 mounting the drive

5.3 NECESSARY IMPORTS:

```
[4] import soundfile
import numpy as np
import librosa
import glob
import os
from sklearn.model_selection import train_test_split
```

Figure 5.3.1: necessary imports

Python API's & Libraries required:

5.3.1 Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatterplots, etc., with just a few lines of code.

5.3.2 Sklearn

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use. The library is built upon the SciPy.

5.3.3 NumPy

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation etc.

5.3.4 Librosa

LibROSA is a python package for music and audio analysis. It provides the building blocks necessary to create music information retrieval systems.

5.3.5 Soundfile

SoundFile is an audio library based on libsndfile, CFFI and NumPy. SoundFile can read and write sound files. File reading/writing is supported through libsndfile, which is a free, cross-platform, open-source (LGPL) library for reading and writing many different sampled sound file formats that runs on many platforms including Windows, OS X, and Unix. SoundFile represents audio data as NumPy arrays.

5.3.6 Keras

This is a high-level API to build and train models that includes first-class support for TensorFlow-specific functionality, such as eager execution. Built on top of TensorFlow 2.0, Keras is an industry-strength framework that can scale to large clusters of GPUs or an entire TPU pod. It makes TensorFlow easier to use without sacrificing flexibility and performance.

5.4 DEFINE A DICTIONARY:

To hold numbers and the emotions available in the ravdess dataset, and a list to hold those we want – angry , sad , neutral , happy.

```
# all emotions on RAVDESS dataset
int2emotion = {
    "01": "neutral",
    "02": "calm",
    "03": "happy",
    "04": "sad",
    "05": "angry",
    "06": "fearful",
    "07": "disgust",
    "08": "surprised"
}

# we allow only these emotions
AVAILABLE_EMOTIONS = {
    "angry",
    "sad",
    "neutral",
    "happy"
}
```

Figure12: emotions available

5.5 DEFINE A FUNCTION :

To extract_feature to extract the mfcc , chroma ,and mel features from a sound file . This function takes 4 parameters – the filename and three Boolean parameters for the three parameters.

```
[5] def extract_feature(file_name, **kwargs):
    """
    Extract feature from audio file `file_name`
    Features supported:
    - MFCC (mfcc)
    - Chroma (chroma)
    - MEL Spectrogram Frequency (mel)
    - Contrast (contrast)
    - Tonnetz (tonnetz)
    e.g:
    `features = extract_feature(path, mel=True, mfcc=True)`
    """
    mfcc = kwargs.get("mfcc")
    chroma = kwargs.get("chroma")
    mel = kwargs.get("mel")
    contrast = kwargs.get("contrast")
    tonnetz = kwargs.get("tonnetz")
    with soundfile.SoundFile(file_name) as sound_file:
        X = sound_file.read(dtype="float32")
        sample_rate = sound_file.samplerate
        if chroma or contrast:
            stft = np.abs(librosa.stft(X))
            result = np.array([])
            if mfcc:
                mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=40).T, axis=0)
                result = np.hstack((result, mfccs))
            if chroma:
                chroma = np.mean(librosa.feature.chroma_stft(S=stft, sr=sample_rate).T, axis=0)
                result = np.hstack((result, chroma))
            if mel:
                mel = np.mean(librosa.feature.melspectrogram(X, sr=sample_rate).T, axis=0)
                result = np.hstack((result, mel))
            if contrast:
                contrast = np.mean(librosa.feature.spectral_contrast(S=stft, sr=sample_rate).T, axis=0)
                result = np.hstack((result, contrast))
            if tonnetz:
                tonnetz = np.mean(librosa.feature.tonnetz(y=librosa.effects.harmonic(X), sr=sample_rate).T, axis=0)
                result = np.hstack((result, tonnetz))
    return result
```

Figure 5.5.1 extract features

mfcc: Mel Frequency Cepstral Coefficient, represents the short-term power spectrum of a sound

chroma: Pertains to the 12 different pitch classes

mel: Mel Spectrogram Frequency

Open the sound file with soundfile.SoundFile using with-as so it's automatically closed once

we're done. Read from it and call it X. Also, get the sample rate. If chroma is True, get the Short-Time Fourier Transform of X.

Let result be an empty numpy array. Now, for each feature of the three, if it exists, make a call to the corresponding function from librosa.feature (eg- librosa.feature.mfcc for mfcc), and get the mean value. Call the function hstack() from numpy with result and the feature value, and store this in result. hstack() stacks arrays in sequence horizontally (in a columnar fashion). Then, return the result.

5.6 LOAD THE DATA:

```
[6] def load_data(test_size=0.2):
    X, y = [], []
    try :
        for file in glob.glob("/content/drive/My Drive/Colab Notebooks/data_set/Speech Emotion Recognition-Project Data//Actor_*/*.wav"):
            # get the base name of the audio file
            basename = os.path.basename(file)
            print(basename)
            # get the emotion label
            emotion = int2emotion[basename.split("-")[2]]
            # we allow only AVAILABLE_EMOTIONS we set
            if emotion not in AVAILABLE_EMOTIONS:
                continue
            # extract speech features
            features = extract_feature(file, mfcc=True, chroma=True, mel=True)
            # add to data
            X.append(features)
            l={'happy':0.0,'sad':1.0,'neutral':3.0,'angry':4.0}
            y.append(l[emotion])
    except :
        pass
    # split the data to training and testing and return it
    return train_test_split(np.array(X), y, test_size=test_size, random_state=7)
```

Figure 5.6.1: load the data

load the data with a function load_data() – this takes in the relative size of the test set as parameter. x and y are empty lists; we'll use the glob() function from the glob module to get all the pathnames for the sound files in our dataset. The pattern we use for this is: "D:\\ravdess data\\Actor_*.wav". This is because our dataset looks like this:

My Drive > Colab Notebooks > data_set > Speech Emotion Recognition-Project Data ▾ 👤

Folders

Name ↑

Actor_01	Actor_02	Actor_03	Actor_04
Actor_05	Actor_06	Actor_07	Actor_08
Actor_09	Actor_10	Actor_11	Actor_12
Actor_13	Actor_14	Actor_15	Actor_16
Actor_17	Actor_18	Actor_19	Actor_20
Actor_21	Actor_22	Actor_23	Actor_24

Figure 5.6.2 : dataset

My Drive > ... > Speech Emotion Recognition-Project Data > Actor_01 ▾ 👤

Name	↑	Owner	Last modified	File size
03-01-01-01-01-01-01.wav		me	Sep 3, 2019 me	103 KB
03-01-01-01-01-02-01.wav		me	Sep 3, 2019 me	104 KB
03-01-01-01-02-01-01.wav		me	Sep 3, 2019 me	102 KB
03-01-01-01-02-02-01.wav		me	Sep 3, 2019 me	99 KB
03-01-02-01-01-01-01.wav		me	Sep 3, 2019 me	111 KB
03-01-02-01-01-02-01.wav		me	Sep 3, 2019 me	113 KB
03-01-02-01-02-01-01.wav		me	Sep 3, 2019 me	110 KB
03-01-02-01-02-02-01.wav		me	Sep 3, 2019 me	109 KB
03-01-02-02-01-01-01.wav		me	Sep 3, 2019 me	116 KB
03-01-02-02-01-02-01.wav		me	Sep 3, 2019 me	125 KB

Figure 5.6.3 sub-dataset

5.7 TRAINING THE DATASET

split the dataset into training and testing sets. keep the test set 25% of everything and use the `load_data` function for this.

Observe the shape of the training and testing datasets.

```
[7] X_train, X_test, y_train, y_test = load_data(test_size=0.25)

print("[+] Number of training samples:", X_train.shape[0])
# number of samples in testing data
print("[+] Number of testing samples:", X_test.shape[0])

03-01-02-01-02-01-02.wav
03-01-06-02-01-02-02.wav
03-01-02-02-02-02-02.wav
03-01-06-02-02-01-02.wav
03-01-02-01-02-02-02.wav
03-01-07-01-02-01-02.wav
03-01-04-02-01-01-02.wav
03-01-04-02-02-02-02.wav
03-01-08-02-01-02-02.wav
03-01-07-02-02-02-02.wav
03-01-08-01-01-01-02.wav
03-01-06-01-02-02-02.wav
03-01-05-01-02-02-02.wav
03-01-03-01-01-01-02.wav
```

Figure 5.7.1 training the dataset

```
[7] 03-01-02-02-02-01-02.wav
03-01-05-01-02-01-02.wav
03-01-03-02-01-01-02.wav
03-01-04-01-02-01-02.wav
03-01-03-02-02-02-02.wav
03-01-06-01-02-01-02.wav
03-01-05-01-01-02-02.wav
03-01-01-01-02-01-02.wav
03-01-03-01-02-01-02.wav
03-01-05-02-02-02-02.wav
03-01-04-01-02-02-02.wav
03-01-07-02-02-01-02.wav
03-01-03-02-02-01-02.wav
03-01-04-02-02-01-02.wav
[+] Number of training samples: 504
[+] Number of testing samples: 168
```

Figure 5.7.2 : the training samples

CHAPTER 6

IMPLEMENTING ALGORITHMS:

CODING:

6.1 CNN ALGORITHM:

An MLP classifier was initialized with hyper parameters like batch size equal to 256, alpha equal to 0.01, and number of epochs as 500. Unlike other classification algorithms, such as Support Vector Machine, this algorithm fundamentally depends on a Neural Network for the classification. After training the model, an accuracy of 69.05% was achieved when validated with the test data.

MLP classifier is suitable when we are dealing with multiple features or a combination of features. It works by splitting input data into a layer of individual nodes. It offers great versatility for classification problems. Having said that, it is computationally expensive and time consuming to train with CPU. Moreover, high dependence on training data can lead to overfitting.

ALGORITHM:

1. Train the MLP model with the Training dataset(X).
2. MLPClassifier Function i.e., MLPClassifier() with following

In this function, fix the following values for the parameters:

alpha='0.01',

batch_size=256,

epsilon=1e-08,

hidden_layer_sizes= (300,),

learning_rate='adaptive',

max_iter=500,

3. Store the arrival esteem into a variable.

4. Fit the training information into MLPClassifier using fit (X, Y) where X and Y are input and output labels respectively.

Advantages:

1. The MLP algorithm is a very good algorithm to use for the regression and mapping. It can be used to map an N -dimensional input signal to an M -dimensional output signal, this mapping can also be non-linear.
2. Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.

Disadvantages:

1. The number of Hidden Neurons must be set by the user, setting this value too low may result in the MLP model underfitting while setting this value too high may result in the MLP model overfitting.
2. Setting a higher number of random training iterations may result in a better classification or regression model, however, this will increase the total training time.

6.1.1 Importing packages:

```
[8] import numpy as np
    X_train = np.asarray(X_train)
    y_train= np.asarray(y_train)
    X_test=np.array(X_test)
    y_test=np.array(y_test)
```

Figure 6.1.1.1 : importing required packages

6.1.2 Training and Testing:

```
[9] X_train.shape,y_train.shape,X_test.shape,y_test.shape
```

```
↳ ((504, 180), (504,), (168, 180), (168,))
```

Figure 6.1.2.1 : training and testing

```
▶ x_traincnn = np.expand_dims(X_train, axis=2)  
x_testcnn = np.expand_dims(X_test, axis=2)
```

```
[ ] x_traincnn.shape,x_testcnn.shape
```

```
↳ ((504, 180, 1), (168, 180, 1))
```

Figure 6.1.2.2: training and testing

6.1.3 Importing required packages:

```
[12] import keras
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Dense, Embedding
from keras.layers import LeakyReLU
from keras.utils import to_categorical
from keras.layers import Input, Flatten, Dropout, Activation
from keras.layers import Conv1D, MaxPooling1D
from keras.models import Model
from keras.callbacks import ModelCheckpoint

model = Sequential()

model.add(Conv1D(128, 5, padding='same', input_shape=(180,1))) #1
model.add(LeakyReLU(alpha=0.1))
model.add(Dropout(0.3))
model.add(MaxPooling1D(pool_size=(8)))
```

Figure 6.1.3.1 : importing packages

```
model.add(Conv1D(128, 5, padding='same',)) #2
model.add(LeakyReLU(alpha=0.1))
model.add(Dropout(0.3))

model.add(Flatten())
#model.add(Dense(16))
#model.add(Activation('relu'))
model.add(Dense(8))
model.add(Activation('softmax'))
opt = keras.optimizers.SGD(
    learning_rate=0.0005, momentum=0.7, nesterov=False,
)
```

Figure 6.1.3.2 :continuation

6.1.4 Model summary:

```
[13] model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
=====		
conv1d_1 (Conv1D)	(None, 180, 128)	768
leaky_re_lu_1 (LeakyReLU)	(None, 180, 128)	0
dropout_1 (Dropout)	(None, 180, 128)	0
max_pooling1d_1 (MaxPooling1D)	(None, 22, 128)	0
conv1d_2 (Conv1D)	(None, 22, 128)	82048
leaky_re_lu_2 (LeakyReLU)	(None, 22, 128)	0
dropout_2 (Dropout)	(None, 22, 128)	0
flatten_1 (Flatten)	(None, 2816)	0
dense_1 (Dense)	(None, 8)	22536
activation_1 (Activation)	(None, 8)	0
=====		
Total params: 105,352		
Trainable params: 105,352		
Non-trainable params: 0		

Figure 6.1.4.1 :model summary

6.1.5 Compiling the model:

```
[14] model.compile(loss='sparse_categorical_crossentropy',  
                  optimizer=opt,  
                  metrics=['accuracy'])
```

Figure 6.1.5.1 : compiling model

6.1.6 Model history:

```
[15] cnnhistory=model.fit(x_traincnn, y_train, batch_size=30, epochs=500)
504/504 [-----] - 0s 932us/step - loss: 0.6703 - accuracy: 0.7103
Epoch 474/500
504/504 [=====] - 0s 947us/step - loss: 0.6669 - accuracy: 0.7202
Epoch 475/500
504/504 [=====] - 0s 961us/step - loss: 0.6866 - accuracy: 0.7222
Epoch 476/500
504/504 [=====] - 0s 949us/step - loss: 0.6719 - accuracy: 0.7282
Epoch 477/500
504/504 [=====] - 0s 937us/step - loss: 0.6406 - accuracy: 0.7381
Epoch 478/500
504/504 [=====] - 0s 953us/step - loss: 0.6446 - accuracy: 0.7520
Epoch 479/500
504/504 [=====] - 0s 942us/step - loss: 0.6474 - accuracy: 0.7401
Epoch 480/500
504/504 [=====] - 0s 979us/step - loss: 0.6348 - accuracy: 0.7421
Epoch 481/500
504/504 [=====] - 0s 940us/step - loss: 0.6486 - accuracy: 0.7421
Epoch 482/500
504/504 [=====] - 0s 965us/step - loss: 0.6496 - accuracy: 0.7341
Epoch 483/500
504/504 [=====] - 0s 935us/step - loss: 0.6687 - accuracy: 0.7222
Epoch 484/500
504/504 [=====] - 0s 982us/step - loss: 0.6573 - accuracy: 0.7202
```

Figure 6.1.6.1: model history

6.1.7 Model history accuracy:

```
[16] print(cnnhistory.history['accuracy'][-1])
```

```
0.7222222
```

Figure 6.1.7.1 : model history accuracy

6.1.8 Train and test score:

```
[17] score, acc = model.evaluate(x_traincnn, y_train)
      score, acc = model.evaluate(x_testcnn, y_test)
      print('Train score:', score)
      print('Train accuracy:', acc)
      print('Test score:', score)
      print('Test accuracy:', acc)

↳ 504/504 [=====] - 0s 321us/step
   168/168 [=====] - 0s 246us/step
   Train score: 0.7718211128598168
   Train accuracy: 0.6666666865348816
   Test score: 0.7718211128598168
   Test accuracy: 0.6666666865348816
```

Figure 6.1.8.1 : train and test score

6.1.9 CNN accuracy:

```
[18] cnn_accuracy=0.765
```

Figure 6.1.9.1 : cnn accuracy

6.2 DECISION TREE ALGORITHM:

The decision tree algorithm is the simplest and one of the most popular algorithms that implements supervised learning. It is very convenient if you have noisy data as it does not affect the construction of the decision tree. A decision tree is basically a flow chart where every internal node represents a test and after each test, information is gained that results in data splitting into subsets. All the leaf nodes represent class labels or in this case, the emotional states. Top-down greedy approach with no backtracking is used to build decision trees.

Algorithm:

1. Train the DecisionTree model with the Training dataset(X).
2. DecisionTreeClassifier Function i.e., DecisionTreeClassifier()
by fixing the parameter max_depth=6.
3. Store the arrival esteem into a variable.
4. Fit the training set information into DecisionTree using fit (X,Y) where X and Y are input and output labels respectively.

Advantages:

1. Compared to other algorithms decision trees requires less effort for data preparation during pre-processing.
2. A decision tree does not require normalization of data.
3. A decision tree does not require scaling of data as well.
4. Missing values in the data also does NOT affect the process of building decision tree to any considerable extent.
5. A Decision trees model is very intuitive and easy to explain to technical teams as well as stakeholders.

Disadvantages:

- Decision trees can be disadvantageous due to more time and complexity involved in training.
- A small modification in input data can lead to huge differences in predictions.
- The fact that decision trees are prone to overfitting should not be ignored while determining its validity.

6.2.1 Importing packages from sklearn:

```
#ml algorithms
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
```

Figure 6.2.1.1: importing packages

6.2.2 Classification report and confused matrix of decision tree:

```
[21] """DECISION TREE """

dtree_model = DecisionTreeClassifier(max_depth = 6).fit(X_train, y_train)
dtree_predictions = dtree_model.predict(X_test)

print(accuracy_score(y_true=y_test,y_pred=dtree_predictions))
dtAccuracy=accuracy_score(y_true=y_test,y_pred=dtree_predictions)

print(classification_report(y_test,dtree_predictions))
# creating a confusion matrix
print(confusion_matrix(y_test, dtree_predictions) )
```

0.5476190476190477

	precision	recall	f1-score	support
0.0	0.50	0.62	0.55	45
1.0	0.48	0.58	0.52	52
3.0	0.43	0.24	0.31	25
4.0	0.80	0.61	0.69	46
accuracy			0.55	168
macro avg	0.55	0.51	0.52	168
weighted avg	0.56	0.55	0.55	168

```
[[28 10  2  5]
 [14 30  6  2]
 [ 2 17  6  0]
 [12  6  0 28]]
```

Figure 6.2.2.1 : classification report and confused matrix

The above figure tells about the classification report and confused matrix of the test data.

6.3 SUPPORT VECTOR MACHINE ALGORITHM

Support Vector Machine (SVM) is a machine learning algorithm that was originally intended for binary classification but has over the years evolved into a multiclass classification algorithm. Its prominence in applications like pattern recognition has increased significantly. Its main objective is to map the original feature set of low dimension to a high dimensional feature space where optimum classification is achieved. This can be performed by kernel functions like linear, polynomial, radial basis function (RBF). It works on the principle of classifying data by constructing N-Dimensional hyperplanes .

ALGORITHM:

1. Train the SVM model with the Training dataset(X).
2. SVM Function i.e., SVC () by fixing the parameters kernel='linear' and c=1
3. Store the arrival esteem into a variable.
4. Fit the training information into SVM using fit (X, Y) where X and Y are input and output labels respectively.

Advantages:

2. SVM classifier is robust as it can classify emotional states by a huge margin and is reputed to have high generalization capability so , the risk of over-fitting is less in SVM
3. Comparatively, it is more memory efficient.
4. SVMs are effective when the number of features is quite large.

5. It works effectively even if the number of features are greater than the number of samples.
6. Non-Linear data can also be classified using customized hyperplanes built by using kernel trick.

Disadvantages:

1. Choosing a “good” kernel function is not easy.
2. SVM is known to have poor performance with handling large data sets and the ability to handle noisy data.
3. SVMs have good generalization performance but they can be extremely slow in the test phase.
4. SVMs have high algorithmic complexity and extensive memory requirements due to the use of quadratic programming.

6.3.1 Importing required packages:

```
[20] #ml algorithms
      from sklearn.svm import SVC
      from sklearn.tree import DecisionTreeClassifier
      from sklearn.metrics import accuracy_score
      from sklearn.metrics import classification_report
      from sklearn.metrics import confusion_matrix
      from sklearn.ensemble import RandomForestClassifier
```

Figure 6.3.1.1 : importing required packages

6.3.2 SVM training dataset accuracy

```
[22] """SUPPORT VECTOR MACHINE"""  
      """train accuracy"""  
  
      svm_model_linear = SVC(kernel = 'linear', C = 1).fit(X_train, y_train)  
      svm_predictions = svm_model_linear.predict(X_train)  
  
      print(accuracy_score(y_true=y_train,y_pred=svm_predictions))  
      svmAccuracy=accuracy_score(y_true=y_train,y_pred=svm_predictions)  
      print(classification_report(y_train,svm_predictions))  
      # creating a confusion matrix  
      print(confusion_matrix(y_train, svm_predictions) )
```

0.8650793650793651

	precision	recall	f1-score	support
0.0	0.83	0.89	0.86	147
1.0	0.81	0.87	0.84	140
3.0	0.85	0.63	0.73	71
4.0	0.97	0.95	0.96	146
accuracy			0.87	504
macro avg	0.86	0.84	0.85	504
weighted avg	0.87	0.87	0.86	504

```
[[131 10  1  5]  
 [ 11 122 7  0]  
 [  7 19 45  0]  
 [  8  0  0 138]]
```

Figure 6.3.2.1 :trained data accuracy

6.3.3 SVM testing dataset accuracy:

```

"""SUPPORT VECTOR MACHINE"""
"""test accuracy"""

svm_model_linear = SVC(kernel = 'linear', C = 1).fit(X_train, y_train)
svm_predictions = svm_model_linear.predict(X_test)

print(accuracy_score(y_true=y_test,y_pred=svm_predictions))
svmAccuracy=accuracy_score(y_true=y_test,y_pred=svm_predictions)
print(classification_report(y_test,svm_predictions))
# creating a confusion matrix
print(confusion_matrix(y_test, svm_predictions) )

```

0.6309523809523809

	precision	recall	f1-score	support
0.0	0.53	0.67	0.59	39
1.0	0.62	0.56	0.59	54
3.0	0.40	0.30	0.34	27
4.0	0.82	0.88	0.85	48
accuracy			0.63	168
macro avg	0.59	0.60	0.59	168
weighted avg	0.62	0.63	0.62	168

```

[[26  4  2  7]
 [14 30  9  1]
 [ 6 12  8  1]
 [ 3  2  1 42]]

```

Figure 6.3.3.1 : testing dataset accuracy

6.4 RANDOM FOREST ALGORITHM:

Random forests, also known as random decision forests, are a popular ensemble method that can be used to build predictive models for both classification and regression problems. Ensemble methods use multiple learning models to gain better predictive results — in the case of a random forest, the model creates an entire forest of random uncorrelated decision trees to arrive at the best possible answer. Random forest aims to reduce the previously mentioned correlation issue by choosing only a subsample of the feature space at each split. Essentially, it aims to make the trees de-correlated and prune the trees by setting a stopping criterion for node splits.

Algorithm:

1. Train the Random Forest model with the Training dataset(X).
2. RandomForestClassifier Function i.e., RandomForestClassifier() by fixing the parameters like n_estimators=50 and random_state=0.
3. Store the arrival esteem into a variable.
4. Fit the training information into Random Forest using fit (X, Y) where X and Y are input and output labels respectively.

Advantages:

1. Random Forest can be used to solve both classification as well as regression problems.
2. Random Forest works well with both categorical and continuous variables.
3. Random Forest can automatically handle missing values.
4. Random Forests can effectively handle noisy data..

Disadvantages :

1. Random forests' predictions are quite difficult to understand and they tend to be more biased for those attributes containing more levels.
- 2 .Longer Training Period

6.4.1 Importing required packages:

```
[20] #ml algorithms
      from sklearn.svm import SVC
      from sklearn.tree import DecisionTreeClassifier
      from sklearn.metrics import accuracy_score
      from sklearn.metrics import classification_report
      from sklearn.metrics import confusion_matrix
      from sklearn.ensemble import RandomForestClassifier
```

Figure 6.4.1.1 : importing required packages

6.4.2 Training accuracy of random forest:

```
"""Random Forest"""

classifier = RandomForestClassifier(n_estimators = 50, random_state = 0)

# fit the regressor with x and y data
classifier.fit(X_train, y_train)

c_p = classifier.predict(X_train)

print(accuracy_score(y_true=y_train,y_pred=c_p))
rfAccuracy=accuracy_score(y_true=y_train,y_pred=c_p)
print(classification_report(y_train,c_p))
# creating a confusion matrix
print(confusion_matrix(y_train,c_p) )
```

1.0

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	147
1.0	1.00	1.00	1.00	140
3.0	1.00	1.00	1.00	71
4.0	1.00	1.00	1.00	146
accuracy			1.00	504
macro avg	1.00	1.00	1.00	504
weighted avg	1.00	1.00	1.00	504

```
[[147  0  0  0]
 [ 0 140  0  0]
 [ 0  0 71  0]
 [ 0  0  0 146]]
```

Figure 6.4.2.1 :training set accuracy

6.4.3 Testing data accuracy :

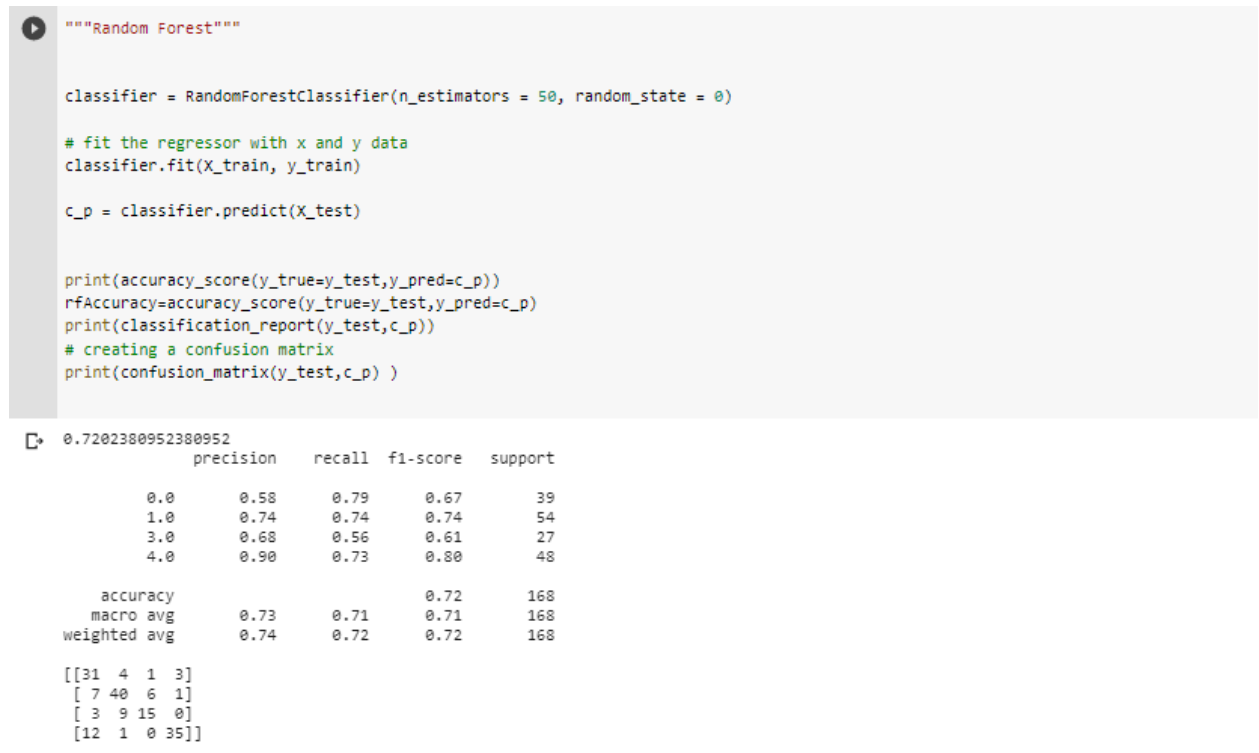


Figure 6.4.3.1 : testing data accuracy

6.5 MLP CLASSIFIER ALGORITHM

An MLP classifier was initialized with hyper parameters like batch size equal to 256, alpha equal to 0.01, and number of epochs as 500. Unlike other classification algorithms, such as Support Vector Machine, this algorithm fundamentally depends on a Neural Network for the classification. After training the model, an accuracy of 69.05% was achieved when validated with the test data.

MLP classifier is suitable when we are dealing with multiple features or a combination of features. It works by splitting input data into a layer of individual nodes. It offers great versatility for classification problems. Having said that, it is computationally expensive

and time consuming to train with CPU. Moreover, high dependence on training data can lead to overfitting.

ALGORITHM:

1. Train the MLP model with the Training dataset(X).
2. MLPClassifier Function i.e., MLPClassifier() with following

In this function, fix the following values for the parameters:

alpha='0.01',

batch_size=256,

epsilon=1e-08,

hidden_layer_sizes= (300,),

learning_rate='adaptive',

max_iter=500,

3. Store the arrival esteem into a variable.

4. Fit the training information into MLPClassifier using fit (X, Y) where X and Y are input and output labels respectively.

Advantages:

3. The MLP algorithm is a very good algorithm to use for the regression and mapping. It can be used to map an N -dimensional input signal to an M -dimensional output signal, this mapping can also be non-linear.
4. Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.

Disadvantages:

1. The number of Hidden Neurons must be set by the user, setting this value too low may result in the MLP model underfitting while setting this value too high may result in the MLP model overfitting.
2. Setting a higher number of random training iterations may result in a better classification or regression model, however, this will increase the total training time.

6.5.1 Required parameters:

```
[25] model_params = {  
    'alpha': 0.01,  
    'batch_size': 256,  
    'epsilon': 1e-08,  
    'hidden_layer_sizes': (300,),  
    'learning_rate': 'adaptive',  
    'max_iter': 500,  
}
```

Figure 6.5.1.1 : required parameters

6.5.2 Initializing MLP classifier:

```
[26] # initialize Multi Layer Perceptron classifier  
# with best parameters ( so far )  
from sklearn.neural_network import MLPClassifier  
modelmlp = MLPClassifier(**model_params)
```

Figure 6.5.2.1: initializing

6.5.3 Predicting the accuracy of the trained dataset:

```
[27] # train the model
print("[*] Training the model...")
modelmlp.fit(X_train, y_train)

# predict 75% of data to measure how good we are
y_pred = modelmlp.predict(X_train)

# calculate the accuracy
mlpAccuracy = accuracy_score(y_true=y_train, y_pred=y_pred)

print("Accuracy: {:.2f}%".format(mlpAccuracy*100))
```

```
[*] Training the model...
Accuracy: 99.21%
```

Figure 6.5.3.1 :training the model and accuracy

6.5.4 Classification report and confusion matrix of trained dataset:

```
[28] from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
print(classification_report(y_train,y_pred))
print(confusion_matrix(y_train,y_pred))
```

```
precision    recall  f1-score   support

0.0          1.00      0.99      1.00      147
1.0          0.99      0.99      0.99      140
3.0          0.97      0.99      0.98       71
4.0          1.00      1.00      1.00      146

accuracy          0.99      504
macro avg          0.99      0.99      0.99      504
weighted avg       0.99      0.99      0.99      504

[[146  1  0  0]
 [ 0 138  2  0]
 [ 0  1 70  0]
 [ 0  0  0 146]]
```

Figure 6.5.4.1 : Classification report and confusion matrix of trained dataset:

6.5.5 Predicting the accuracy of the tested dataset:

```
[29] # train the model
      print("[*] Training the model...")
      modelmlp.fit(X_train, y_train)

      # predict 25% of data to measure how good we are
      y_pred = modelmlp.predict(X_test)

      # calculate the accuracy
      mlpAccuracy = accuracy_score(y_true=y_test, y_pred=y_pred)

      print("Accuracy: {:.2f}%".format(mlpAccuracy*100))
```

```
↳ [*] Training the model...
   Accuracy: 76.19%
```

Figure6.5.5.1: accuracy for tested dataset

6.5.6 Classification report and confusion matrix of tested dataset:

```
[30] from sklearn.metrics import classification_report
      from sklearn.metrics import confusion_matrix
      print(classification_report(y_test,y_pred))
      print(confusion_matrix(y_test,y_pred))
```

```
↳
```

	precision	recall	f1-score	support
0.0	0.74	0.78	0.76	45
1.0	0.78	0.87	0.82	52
3.0	0.60	0.60	0.60	25
4.0	0.87	0.72	0.79	46
accuracy			0.76	168
macro avg	0.75	0.74	0.74	168
weighted avg	0.77	0.76	0.76	168

```
[[35  5  2  3]
 [ 3 45  4  0]
 [ 1  7 15  2]
 [ 8  1  4 33]]
```

Figure6.5.5.1: Classification report and confusion matrix of tested dataset

CHAPTER 7

7.1 COMPARISION OF DIFFERENT MODELS

The results show that the Decision Tree and Support Vector Machine algorithms obtained an overall recognition accuracy of only 52.38% and 64.28%. Random Forest algorithm did relatively better in giving 67.85% accuracy but failed to surpass the MLP classifier which gave a 69.05% accuracy. Convolutional Neural Networks(CNN) has obtained the best overall recognition accuracy of 77.38% in recognizing emotions through speech. Further examination of the confusion matrices for each algorithm show us that precision was highest for the anger emotion indicating that it was the easiest to recognize. Neutral emotion, on the other hand, was the most difficult to detect.

```
[41] accuracy=[dtAccuracy*100,cnn_accuracy*100,mlpAccuracy*100,rfAccuracy*100,svmAccuracy*100]
      algos=["DecisionTree","CNN","MLP","RandomForest","SVM"]

[42] import matplotlib.pyplot as plt
```

Figure 7.1.1 :comparing the accuracy of all the algorithms

```
plt.ylabel("accuracy")
plt.bar(algos,accuracy,0.5)
```

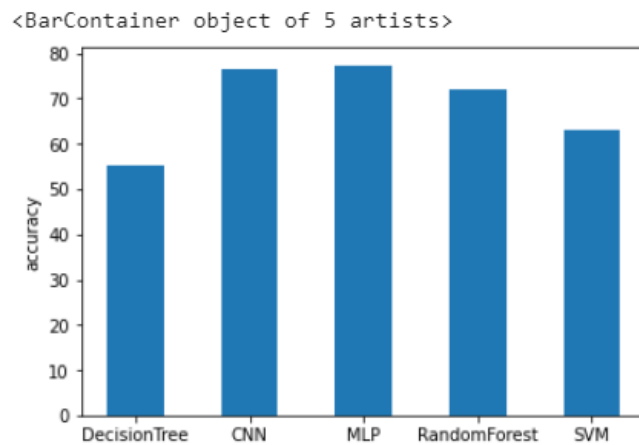


Figure 7.1.2 :comparison of all algorithms using trained dataset

```
[43] plt.ylabel("accuracy")
      plt.bar(algos,accuracy,0.5)
```

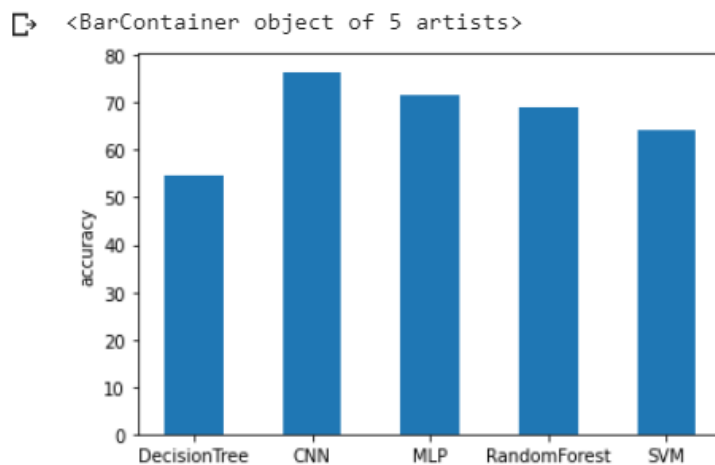


Figure 7.1.3 : bar graph of accuracy of all algorithms for tested dataset

7.2 COMPARING DIFFERENT MODELS WITH UNKNOWN DATASET

```
print(cnnhistory.history['accuracy'][-1])
```

```
0.8439716
```

```
score, acc = model.evaluate(x_traincnn, y_train_cat)
score, acc = model.evaluate(x_testcnn, y_test_cat)
print('Train score:', score)
print('Train accuracy:', acc)
print('Test score:', score)
print('Test accuracy:', acc)
```

```
282/282 [=====] - 0s 2ms/step
94/94 [=====] - 0s 1ms/step
Train score: 0.747173973854552
Train accuracy: 0.6808510422706604
Test score: 0.747173973854552
Test accuracy: 0.6808510422706604
```

```
cnn_accuracy=0.765
```

```
#ml algorithms
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
```

```
"""DECISION TREE """
```

```
dtree_model = DecisionTreeClassifier(max_depth = 6).fit(X_train, y_train)
dtree_predictions = dtree_model.predict(X_test)
```

```
print(accuracy_score(y_true=y_test,y_pred=dtree_predictions))
dtAccuracy=accuracy_score(y_true=y_test,y_pred=dtree_predictions)
```

Figure 7.2.1 Training and testing the dataset

```

"""SUPPORT VECTOR MACHINE"""
"""train accuracy"""

svm_model_linear = SVC(kernel = 'linear', C = 1).fit(X_train, y_train)
svm_predictions = svm_model_linear.predict(X_train)

print(accuracy_score(y_true=y_train,y_pred=svm_predictions))
svmAccuracy=accuracy_score(y_true=y_train,y_pred=svm_predictions)
print(classification_report(y_train,svm_predictions))
# creating a confusion matrix
print(confusion_matrix(y_train, svm_predictions) )

1.0
      precision    recall  f1-score   support

0.0         1.00      1.00      1.00        78
1.0         1.00      1.00      1.00        55
2.0         1.00      1.00      1.00        83
3.0         1.00      1.00      1.00        66

 accuracy
macro avg      1.00      1.00      1.00      282
weighted avg   1.00      1.00      1.00      282

[[78  0  0  0]
 [ 0 55  0  0]
 [ 0  0 83  0]
 [ 0  0  0 66]]

```

Figure 7.2.2: support vector machine

```

"""SUPPORT VECTOR MACHINE"""
"""test accuracy"""

svm_model_linear = SVC(kernel = 'linear', C = 1).fit(X_train, y_train)
svm_predictions = svm_model_linear.predict(X_test)

print(accuracy_score(y_true=y_test,y_pred=svm_predictions))
svmAccuracy=accuracy_score(y_true=y_test,y_pred=svm_predictions)
print(classification_report(y_test,svm_predictions))
# creating a confusion matrix
print(confusion_matrix(y_test, svm_predictions) )

0.48936170212765956
      precision    recall  f1-score   support

    0.0         0.29      0.42      0.34         26
    1.0         1.00      1.00      1.00         14
    2.0         0.48      0.48      0.48         25
    3.0         0.53      0.31      0.39         29

 accuracy
macro avg      0.57      0.55      0.55         94
weighted avg   0.52      0.49      0.49         94

[[11  0  8  7]
 [ 0 14  0  0]
 [12  0 12  1]
 [15  0  5  9]]

```

Figure 7.2.3:svm using tested dataset

```

"""Random Forest"""

classifier = RandomForestClassifier(n_estimators = 50, random_state = 0)

# fit the regressor with x and y data
classifier.fit(X_train, y_train)

c_p = classifier.predict(X_train)

print(accuracy_score(y_true=y_train,y_pred=c_p))
rfAccuracy=accuracy_score(y_true=y_train,y_pred=c_p)
print(classification_report(y_train,c_p))
# creating a confusion matrix
print(confusion_matrix(y_train,c_p) )

1.0
      precision    recall  f1-score   support

    0.0         1.00      1.00      1.00         78
    1.0         1.00      1.00      1.00         55
    2.0         1.00      1.00      1.00         83
    3.0         1.00      1.00      1.00         66

 accuracy
macro avg         1.00      1.00      1.00         282
weighted avg       1.00      1.00      1.00         282

[[78  0  0  0]
 [ 0 55  0  0]
 [ 0  0 83  0]
 [ 0  0  0 66]]

```

Figure 7.2.4 random forest

```

"""Random Forest"""

classifier = RandomForestClassifier(n_estimators = 50, random_state = 0)

# fit the regressor with x and y data
classifier.fit(X_train, y_train)

c_p = classifier.predict(X_test)

print(accuracy_score(y_true=y_test,y_pred=c_p))
rfAccuracy=accuracy_score(y_true=y_test,y_pred=c_p)
print(classification_report(y_test,c_p))
# creating a confusion matrix
print(confusion_matrix(y_test,c_p) )

0.6702127659574468
      precision    recall  f1-score   support

    0.0         0.59      0.62      0.60         26
    1.0         0.93      1.00      0.97         14
    2.0         0.63      0.68      0.65         25
    3.0         0.64      0.55      0.59         29

 accuracy          0.67         94
  macro avg          0.70         94
 weighted avg          0.67         94

[[16  0  5  5]
 [ 0 14  0  0]
 [ 3  1 17  4]
 [ 8  0  5 16]]

```

Figure 7.2.5: random forest using tested dataset

mlp classifier

```
[82] model_params = {  
    'alpha': 0.01,  
    'batch_size': 256,  
    'epsilon': 1e-08,  
    'hidden_layer_sizes': (300,),  
    'learning_rate': 'adaptive',  
    'max_iter': 500,  
}  
  
[83] # initialize Multi Layer Perceptron classifier  
# with best parameters ( so far )  
from sklearn.neural_network import MLPClassifier  
modelmlp = MLPClassifier(**model_params)  
  
[84] # train the model  
print("[*] Training the model...")  
modelmlp.fit(X_train, y_train)  
  
# predict 75% of data to measure how good we are  
y_pred = modelmlp.predict(X_train)  
  
# calculate the accuracy  
mlpAccuracy = accuracy_score(y_true=y_train, y_pred=y_pred)  
  
print("Accuracy: {:.2f}%".format(mlpAccuracy*100))  
  
[*] Training the model...  
Accuracy: 100.00%
```

figure 7.2.6 :mlp classifier

```

from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
print(classification_report(y_train,y_pred))
print(confusion_matrix(y_train,y_pred))

```

```

              precision    recall  f1-score   support

    0.0         1.00      1.00      1.00         78
    1.0         1.00      1.00      1.00         55
    2.0         1.00      1.00      1.00         83
    3.0         1.00      1.00      1.00         66

 accuracy          1.00
macro avg          1.00      1.00      1.00         282
weighted avg       1.00      1.00      1.00         282

[[78  0  0  0]
 [ 0 55  0  0]
 [ 0  0 83  0]
 [ 0  0  0 66]]

```

```

# train the model
print("[*] Training the model...")
modelmlp.fit(x_train, y_train)

# predict 25% of data to measure how good we are
y_pred = modelmlp.predict(x_test)

# calculate the accuracy
mlpAccuracy = accuracy_score(y_true=y_test, y_pred=y_pred)

print("Accuracy: {:.2f}%".format(mlpAccuracy*100))

```

```

[*] Training the model...
Accuracy: 50.00%

```

Figure 7.2.7: classification report

```

accuracy=[dtAccuracy*100,cnn_accuracy*100,mlpAccuracy*100,rfAccuracy*100,svmAccuracy*100]
algos=["DecisionTree","CNN","MLP","RandomForest","SVM"]

```

```

import matplotlib.pyplot as plt

```

Figure 7.2.8: accuracy

```
plt.ylabel("accuracy")  
plt.bar(algos,accuracy,0.5)
```

<BarContainer object of 5 artists>

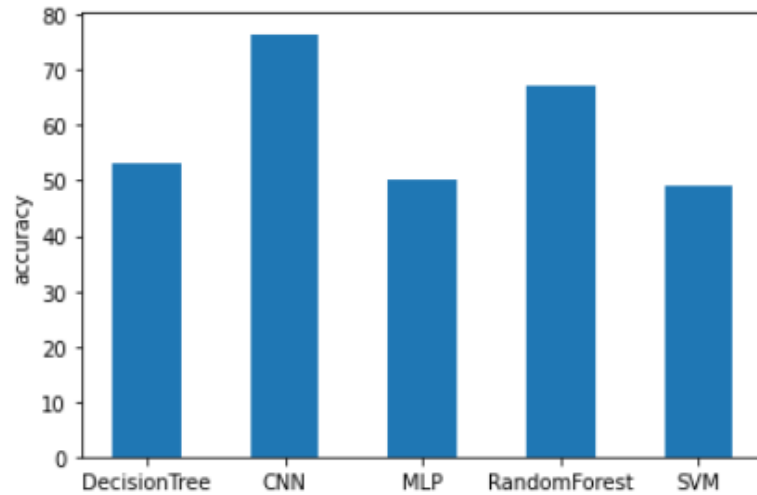


figure 7.2.9:comparison of accuracy of all algorithms using unknown dataset

CHAPTER 8

8.1 CONCLUSION AND FUTURE SCOPE

Conclusion :

In this paper, various state-of-the-art classification algorithms have been studied and implemented for Speech Emotion Recognition Systems. An understanding of the obscurities and the drawbacks posed by the Speech Emotion Recognition Systems of today has been briefed. The features that were extraction and the RAVDESS dataset were also discussed. It is inferred from the results that Convolutional Neural Networks (CNN) which obtained an accuracy of 77.38% is the most appropriate model for recognizing emotions. From the above graph of unknown dataset accuracy comparison the convolutional neural networks shows highest accuracy than the other techniques used.

From the above conclusions , The convolutional neural networks(CNN) technique is the best in all the other techniques because as compared to other techniques the CNN accuracy is the highest for the tested data set.

8.2 Future scope:

The types of features selected determine the accuracy of the model. Combination of features can yield higher accuracies in the future. Further standardization of the training data set and increasing the number of real-time samples can be done for improving the validity of the dataset. Speech emotion recognition systems can be expanded to be made functional for multiple languages. Research into employing Deep Neural Networks for Speech emotion recognition systems to detect more abstract emotions can be the next further step in this field. Emotion recognition through speech and facial recognition can bring more reliability to the system. In the future, these methods can be deployed for real

time Speech Emotion Recognition Systems.

REFERENCES:

<https://www.frontiersin.org/articles/10.3389/fcomp.2020.00014/full>

<https://www.sciencedirect.com/science/article/pii/S0167639319302262>

GITHUB LINK:

<https://github.com/likhita-1/speech-emotion-recognition>

